Design, Implementation, and Evaluation of Scalable Content-Based Image Retrieval Techniques

WONG, Yuk Man

A Thesis Submitted in Partial Fulfilment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

©The Chinese University of Hong Kong
August 2007

Thesis/Assessment Committee Members

Professor Irwin King (Chair)
Professor Michael R. Lyu (Thesis Supervisor)
Professor Leo J. Jia (Committee Member)
Professor L. Quan (External Examiner)

Abstract of thesis entitled:

Design, Implementation, and Evaluation of Scalable Content-Based Image Retrieval Techniques

Submitted by WONG, Yuk Man

for the degree of Master of Philosophy

at The Chinese University of Hong Kong in August 2007

Content-based image retrieval (CBIR) has been a very active research topic in multimedia community since the early 1990s. A lot of CBIR systems have also been built in both industry and academia. However, most of the previously proposed CBIR systems are not scalable to very large database. They suffer from performance problem when trying to scale up to a large database. The main goal of this thesis is to address the performance problem and investigate some ways to further improve the performance of the CBIR system.

In this thesis, we first present a novel scalable CBIR scheme using an emerging data indexing technique, locality-sensitive hashing (LSH), which is shown to be scalable to high dimensional data. With this indexing technique, our CBIR system can retrieve image without much degradation ($\sim 5\%$) on accuracy when comparing with exhaustive linear search with Euclidean distance measure. We propose a parallel and distributed scheme which not only can address the problems of applying LSH in CBIR but also can significantly improve the responsiveness of the CBIR system.

We then present a comprehensive empirical performance evaluation of the proposed CBIR system over one million images. To the best of our knowledge, there are very limited empirical studies on such large-scale CBIR evaluation. Our empirical results show that our proposed solution is able to scale for a million of images, which is promising for building Web-scale CBIR systems.

Next, we propose a novel image near-duplicate detection system for efficient near-duplicate image retrieval. This system adopts the Scale Invariant Feature Transform (SIFT) descriptor to describe image features and Exact Euclidean Locality-Sensitive Hashing (E^2LSH) to index the descriptors. To further improve the performance of the system, we propose a new matching strategy and a new verification process that are not adopted by other image near-duplicate detection system. The resulting accuracy is high and the system's response time is short even for databases of millions of keypoints.

Furthermore, we study the invariant local feature descriptors which have recently been widely employed in many computer vision applications, including image retrieval and object recognition. To realize their real performance, we have performed an empirical performance evaluation on some state-of-the-art descriptors. Then, we propose a new feature descriptor which extends the SIFT descriptor to achieve background and object color invariance. It commonly happens that images of the semantic topic differ with background and object color. Thus our feature descriptor is particularly useful for semantic search in CBIR. The performance evaluation shows that our descriptor performs better than others on datasets that capture changes in background and object color.

摘要

基於內容的圖像檢索(CBIR)是從 90 年代初期開始在多媒體社區非常活躍的研究題目。到現在爲止，已經有很多 CBIR 系統在工業界和學術界被建立了。 然而，大多以前提出的 CBIR 系統並不適用於非常大的資料庫。當它們被用於大資料庫時，它們會有很大的效能問題。這份論文的主要目標是設法解決 CBIR 系統的效能問題和調查一些方法進一步改進 CBIR 系統表現。

在這份論文，我們首先提出一個新穎的、適用於大資料庫的 CBIR 解決方案，這個方案使用了一個嶄新的、被證明就算被用於高維資料庫也能保持高效能的資料索引技術，LSH。透過這個資料索引技術，我們的 CBIR 系統可以以不差於線性搜索 5 個百分比的準確性來進行圖像檢索。我們提出了一個不僅能解決 LSH 的使用問題，而且可以明顯地加快 CBIR 系統的並聯和分佈式的解決方案。

然後我們會對我們提出的CBIR系統在一百萬個圖像資料庫上運行的表現進行了一個全面的效能評估。就我們所知，學界裡很少有對 CBIR 系統在這樣大型的圖像資料庫上運行作全面的效能評估報告。我們實驗結果表示，我們的提議解決方案能爲成千上萬的圖像作快速圖像檢索。這爲我們將要建造的、用來檢索互聯網上的圖像的 CBIR 系統帶來希望。

其次，我們提出一個新穎的高效率圖像近複製檢測系統。這個系統採取 SIFT 圖像特點描述技術來描述圖像特點和使用高效能資料索引技術 LSH 來索引這些圖像特點。爲進一步改進系統的表現，我們提議了一個未曾在其他近複製圖像檢測系統中找到的配比度量和結果核實過程。我們的系統既有高度的準確性，就算在成千上萬的圖像特點資料庫上亦有很短的反應時間。

除此之外，我們研究了最近在許多電腦視覺應用程序(包括圖像檢索和物件辨認)廣泛使用了的特點描述技術。 爲了知道他們真正的表現，我們對一些最新的描述技術進行了一個表現評估。然後，我們提議擴大 SIFT 描述技術以達到背景和物件顏色不變性。在同一語意題目之下的圖像裡面有形狀一樣而僅著色不同的物件與背景是很常見的。因而我們的特點描述技術對語義查尋是特別有用的。表現評估表示，我們的描述技術比其他執行在擁有形狀一樣而僅著色不同的物件與背景的圖像的資料集更好。

Acknowledgement

First of all, I would like to give wholehearted thanks to my supervisor, Prof. Michael R. Lyu, who has given me a lot of support, guidance, and advices throughout the past three years.

I would also like to sincere thanks to my fellow research partner, Steven Hoi, who always kindly gives me helping hand to my research.

Moreover, I would like to thanks Prof. Irwin King and Prof. Leo Jia for being my internal examiners.

Last but not least, I want to give thanks to my dear friends and fellow colleagues, Pat Chan, Alan Chu, Albert Lam, Oscar Leung, Stephen Leung, KK Lo, Edith Ngai, Brian Tsui, Peng Xiang, Eric Yu, and many others. The days we study together are happy and full of joy. I really enjoy the time with them.

This work is dedicated to my family for their continuous encouragement.

Contents

Chapter 1

Introduction

1.1 Overview

Content-based image retrieval (CBIR) has been a very active research topic in multimedia community since the early 1990s. In general, a CBIR system at least consists of four modules: data acquisition and processing, feature representation, data indexing, query feedback processing. In the past decades, many techniques for feature representation, relevance feedback, and multidimensional indexing have been proposed. A lot of CBIR systems have also been built in both industry and academia. These includes IBM's QBIC [41], Virage [2], MIT's Photobook [42], etc. Some of them [42] are designed to work in small image database ($100 - 10,000$) only and are thus not scalable to database as large as 1 million images. In those systems, simple exhaustive linear search suffices to provide prompt searching performance. Other systems [9, 11, 26] employ dimension reduction techniques like Karhunen Lòeve Transform (KLT) to reduce the dimension of the high dimensional feature vector prior to applying traditional multidimensional indexing techniques on the database. Although these systems can perform similarity search in fast speed, the dimension reduction step has already significantly deteriorated the accuracy of the system. Therefore, most of the previously proposed CBIR systems are not scalable to very large databases. They suffer from performance problems when trying to scale up to a large database. Recently, a promising indexing technique, Locality-Sensitive Hashing (LSH), was proposed for solving the near neighbor search in high dimensional spaces efficiently and accurately [6, 19]. However, there have been no research studies employing LSH in CBIR systems for indexing image contents in the previous years.

In a web-scale CBIR system that provides web search service, one of the important issue is on filling the database using the World Wide Web as a logical repository. A common way to obtain the web images from the World Wide Web is to use web image crawlers. During web image crawling, it is not difficult to encounter some web pages that contain images that are nearly the same. For instance, you may easily find multiple sizes of the same image in some photo sharing web sites. The contents of near-duplicate images are very similar and thus these images are likely to be returned together in CBIR search. If the images crawled from the Internet are not filtered before they are used to build the image database of the CBIR system, users of CBIR systems may easily get spammed with near-duplicate images when they submit search queries. This is not desirable since users of CBIR system would not like to see the same image appearing repeatedly when they are searching for images of certain semantics. In other words, users will not be interested in seeing the multiple versions of the same image unless they are interested in that image. Thus, removing the near-duplicate images is critical in improving the quality of CBIR search. An image near-duplicate (IND) detection system [5, 23, 57] is a kind of system that can achieve this purpose.

Previously developed INDs are mostly designed to detect copyrighted images but not in improving the quality of CBIR search. Berrani [5] proposed an IND detection system employing local differential descriptors and approximate similarity search. Yan Ke [23] proposed to use PCA-SIFT invariant local feature descriptors and LSH. According to our performance evaluation, SIFT-based descriptors [32] outperform other local descriptors in matching tasks in terms of both the recall and precision rates. Therefore, previously proposed IND detection systems can be further improved by using this more powerful feature descriptor. On the other hand, locality-sensitive hashing has been proved [23] to be effective in finding near neighbors both in accuracy and speed. However, the LSH algorithm employed by Yan Ke assumes L1 (Manhattan) distance in the analysis of near neighbors, which is not as effective as L2 distance, as shown in [23]. In short, these previously proposed systems can be further improved by considering SIFT and LSH technique.

Recently, local invariant feature [45, 32, 37] is becoming more and more popular and there are increasing interests in applying local invariant feature in CBIR systems. According to our performance evaluation, SIFT local descriptor performs the best in matching tasks among the others. However, SIFT cannot be directly applied in CBIR which searches for images with a specific semantic topic. This is because SIFT descriptor is variant to change in background and object color. For example, two images of the same object on two backgrounds with different colors may have very different SIFT descriptors. Also, two images of the same kind of object but with different color appearance will have very different SIFT descriptors. Thus, it will be interesting if we can separate some sharp dependent components from SIFT so that it is more invariant to changing background and object color. When needed, these shape dependent components can be integrated back to the descriptor to enhance the accuracy of CBIR search for exact individual images, rather than images under the same semantic topic.

1.2 Contribution

This thesis focuses on designing, implementing, and evaluating a large-scale content-based image retrieval system. Our main contributions are summarized in the following aspects:

One of our major contributions is to propose a novel scalable CBIR scheme using an emerging data indexing technique, locality-sensitive hashing, which is shown to be scalable to high dimensional data. With this indexing technique, the CBIR system can achieve fast image retrieval by sampling only a very small subset of data points in the database. In addition to being fast, our CBIR system can retrieve image without much degradation ($\sim 5\%$) on accuracy when comparing with exhaustive linear search with Euclidean distance measure. Applying this technique is not trivial, since there are problems in applying LSH to index large database. Thus we propose a parallel and distributed scheme which not only can overcome the problems of LSH but also can significantly improve the responsiveness of the CBIR system.

The second important contribution in this thesis is a comprehensive empirical performance evaluation of the proposed CBIR system over one million images. To the best of our knowledge, there are very limited empirical studies on such large-scale CBIR evaluation. Our empirical results show that our proposed solution is able to scale for hundreds of thousands of images, which is promising in building web-scale CBIR systems.

The third contribution in this thesis is to propose a novel image near-duplicate detection system. This system adopts the state-of-the-art SIFT feature descriptor with fast LSH retrieval method that makes the system accurate, fast, and practical. A new verification process, called orientation verification, on the matched local feature is introduced to improve the recall and precision of the system. A new empirical distance metric, called K-NNRatio, that integrates the k Nearest Neighbor algorithm with distance ratio, is also introduced to improve the system performance.

Furthermore, this thesis proposes a new invariant local descriptor, Shape-SIFT (SSIFT), which extends Scale Invariant Feature Transform (SIFT) descriptor to achieve background and object color invariance. Experiments show that our descriptor performs better than other state-of-the-art descriptors on data sets that capture changes in background and object color.

Besides the above contributions, the thesis also contributes to an empirical performance evaluation of the state-of-the-art local invariant descriptors. This evaluation leads to our choice of feature descriptor in the proposed IND detection system as our target to be extended to background and object color invariance.

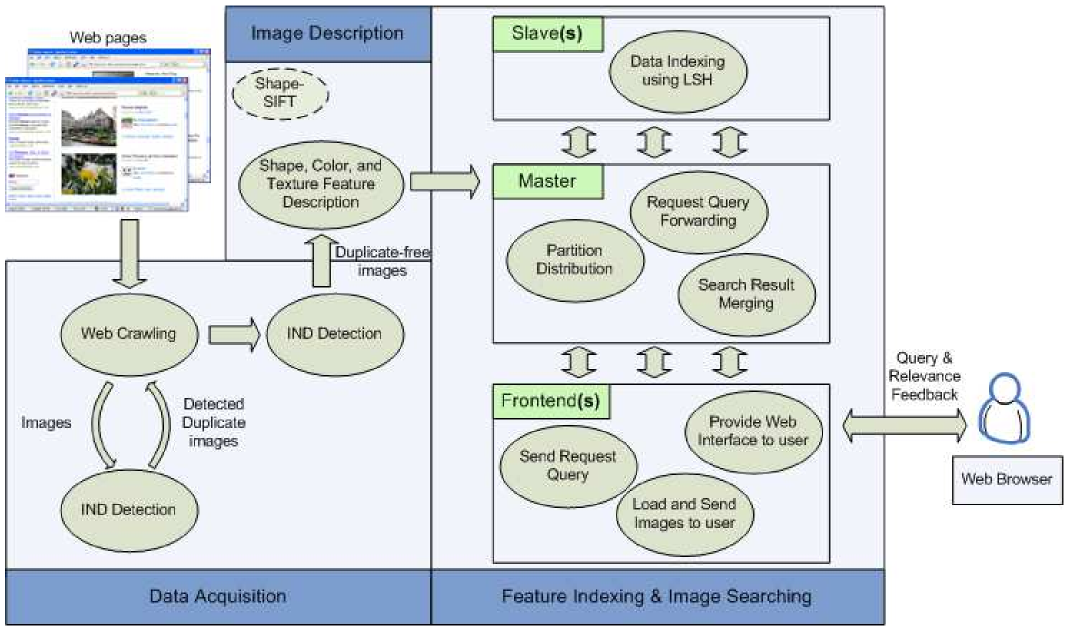


Figure 1.1: Overview of contribution.

Figure 1.1 shows the overview of our contribution in form of a system diagram. We contribute to improve the performance of CBIR system in three major aspects: data acquisition, image description, and feature indexing. For data acquisition, we have built a web image crawler which crawls web pages from the World Wide Web and downloads the images linked to the pages. It first converts the HTML web pages to XML documents and then it parses the documents for certain XML nodes such as <img> to locate the URLs of the target images. Image properties such as *alternate text*, *hyperlink URL*, and *image file name* can aim image search and thus they are also recorded down. They can be found by parsing the XML nodes and attributes around the <img> node. We have also built an novel IND detection system which can filter duplicated images out of the crawled web images. This system can either be applied on the images retrieved from the same web site or be applied on images retrieved from a set of related sites. For image description, we propose to represent the image contents of the duplicate-free images using their shape, color, and texture features. Alternatively, we can adopt Shape-SIFT invariant local descriptor to represent the image contents of the images. For feature indexing, we built a parallel and distributed system which distributes indexing tasks over a cluster of machines and performs similarity search using LSH indexing technique. Through the web-based frontend system, user can do search query and provide relevance feedback simply through a web browser in the client machine.

1.3 Organization of This Work

Chapter 2, "Literature Review", surveys the research work on CBIR and reviews the feature detection, description, and matching techniques for invariant local feature. It also presents a performance evaluation on some of the state-of-the-art feature descriptors to find an outstanding technique.

Chapter 3, "A Distributed Scheme for Large-Scale CBIR", proposes a scalable content-based image retrieval scheme that uses LSH as the indexing technique. An extensive performance evaluation on a large image testbed of one million images is also presented.

Chapter 4, "Image Retrieval System for IND Detection", presents a novel IND detection system built using the state-of-the-art invariant local feature detector, descriptor, and indexing techniques. A new matching strategy and a new verification process are also proposed to improve the performance of the IND detection system.

Chapter 5, "Shape-SIFT Feature Descriptor", suggests a new invariant local descriptor, SSIFT, which extends the SIFT descriptor to achieve background and object color invariance.

The thesis concludes in Chapter 6 with discussion of futher research directions on web-scale CBIR.

□ **End of chapter.**

Chapter 2

Literature Review

2.1 Content-based Image Retrieval

Along with the increasing popularity of digital imaging, web blogging, and photo sharing, image retrieval in web-scale image databases has attracted more and more attention in the research community. Through the extensive studies in both academia and industry in the previous years, two main types of approaches have been proposed to attack the image retrieval problem, that is, the problem of retrieving digital images from large databases. The first type of approach is the metadata-based approach. The metadata-based approach is a traditional method of image retrieval that makes use of the metadata of image such as the textual descriptions, captioning, or keywords to search and retrieve images. However, this method is not practical for large databases because it relies on manual image annotation which is expensive and time-consuming. Because of this limitation, there is a growing interest in solving the image retrieval problem using the second type of approach, which is the content-based approach [27]. The content-based approach performs the search based on the analysis of the contents of an image. Since the contents of the image can be automatically derived from the image itself, the content-based approach has the advantage of not relying on any manual image annotation.

2.1.1 Query Technique

Query to content-based image retrieval system (CBIRS) is usually made by providing an example image. This query technique is called *query by example (QBE)*. The example image can either be supplied by the user or chosen from a random set provided by the system. Given the example image, the system then engages its contents on performing similarity search to search for images that share the same low-level features with the provided example. In CBIRS, the contents of an image are represented by low-level global image features. The most commonly used features include color [52], shape [21], and texture [33]. Before image retrieval can be performed, these features are extracted from every image in the database. Usually, the features of each image are then combined into a feature vector which is used as the index of the corresponding image in the database. During an image retrieval, the feature vector of the provided example will be first computed. Then, based on certain similarity measures, a similarity search can be performed over the feature vectors of all images in the database. One of the most commonly used similarity measure is Euclidean (L_2) distance. It is often used because of its simplicity and its robustness. Since the database can be very large, a fast indexing technique is usually employed to speed up the search.

2.1.2 Relevance Feedback

One of the major limitations of CBIR is the semantic gap between high-level concepts of human and low-level features extracted from images. CBIR bases its search upon the extracted low-level features which may not fully capture the high-level concepts specified by the user. Eventually, the user may not get satisfactory result from the search query. Due to this limitation, relevance feedback is introduced [48] into CBIRS to narrow down the semantic gap between high-level concepts and low-level features. Relevance feedback is an interactive mechanism for the user to progressively refine the search results by marking the images in the results as “relevant” or “irrelevant” to the search query and then repeating the search using this additional information. Through the relevance feedback, the user can obtain her desired images by interacting with the system in a round-by-round basis.

2.1.3 Previously Proposed CBIR systems

Many CBIR systems have been proposed in the previous years. Some representative samples of the systems include IBM’s QBIC [41], Virage [2], MIT’s Photobook [42], etc. Many surveys on these systems have also been published. Three of the comprehensive surveys on CBIR research are [27], [47], and [50]. Readers can refer to them for more information about CBIR. Some of these systems, such as Photobook [42], are suitable for searching small database ($100 - 10,000$) only and not scalable to database as large as 1 million images. Since the size of the database used by these systems is small, a simple exhaustive linear search is adequate to provide prompt searching performance. Other systems, such as QBIC [11] and the CBIR systems proposed by Lew [26] and Egas [9], employ some dimension reduction techniques such as principal component analysis (PCA) to reduce the dimension of feature vectors to below 20 before applying a multidimensional indexing technique on the dimension-reduced feature vectors. Many traditional multidimensional indexing techniques are shown to be applicable for these applications, including k-d tree [9], R*-tree [11]. Although these indexing techniques are fast and accurate, the dimension reduction step significantly deteriorate the system performance.

2.2 Invariant Local Feature

Invariant local features refer to the representations of image contents, at some particular interesting regions on the images of scenes or objects. These features are local as they are related to small regions on objects instead of the whole object. The local property makes feature-based recognition inherently robust to occlusion and clutter, which are the two serious problems in recognition using global features. They are usually solved by image segmentation techniques. However, since the performance of current image segmentation techniques are still limited, the performance of recognition using global features is limited, too. On the other hand, local features can solve these problems easily. Since images of the same object can be taken in different environmental and instrumental conditions, they are most likely different but related in content. Differences between these images include image noise level, change in illumination, scaling, rotation and change in viewing angle. In order to match two different images of the same object, the local features should be invariant to these differences. Invariance of a local feature refers to its ability to tolerate these differences. The extent of invariance depends on how its representation is designed. A good local feature should be highly distinctive, which means it should allow for correct object identification with high probability. However, the more invariance a feature has, the less distinctive it is. Therefore, there are trade-offs between *invariance* and *distinctiveness*.

Three keys processes involved in feature-based recognition are *feature detection*, *feature description* and *feature matching*. These three processes have been actively investigated and continuously improved in the last decade. We will discuss some of the state-of-the-art techniques proposed to improve these three processes in the following sections. Then we will perform performance evaluation on feature descriptors in describing features so as to investigate which techniques outperform the others.

2.3 Invariant Local Feature Detector

Since the resolution of an object's image can be very high, it is not practical in efficiency, storage and accuracy to take every pixel of the image as a feature and describe it by a vector. It is more practical to extract only a subset of pixels from an image to be described. We call this subset of pixels the interest points. There are two main requirements on a feature detector. First, corresponding interest points on the object should be repeatedly detected by the feature detector over different images of the same object. Second, detected interest points should be distinctive. 2D image windows where there is some form of 2D texture like corners, are the most distinctive image patch comparing with other types of image windows. A number of feature detectors have been proposed to detect 2D windows, which include Harris corner detector [13], DOG extrema detector [32], Harris-Laplacian detector [39] and affine covariant region detector [40].

2.3.1 Harris Corner Detector

Harris corner detector [13] is widely used in many image matching tasks to select regions that have significant gradient change in all directions.

The Auto-Correlation Matrix

This detector analyzes the auto-correlation matrix **M** of every location in an image that is computed from image derivatives:

$$\mathbf{M} = g(\sigma_I) * \begin{bmatrix} I_x^2(\mathbf{x}) & I_xI_y(\mathbf{x}) \\ I_xI_y(\mathbf{x}) & I_y^2(\mathbf{x}) \end{bmatrix} \tag{2.1}$$

where $\mathbf{x}$ is the pixel location vector, $I_x(\mathbf{x})$ is the x-gradient at location $\mathbf{x}$, $I_y(\mathbf{x})$ is the y-gradient at location $\mathbf{x}$ and $g(\sigma_I)$ is the gaussian kernel of scale σ_I.

Eigenspace Analysis

A point is located at a corner if its corner response is large. The corner response **R** can be computed from matrix **M** by the following equation:

$$\begin{aligned}\mathbf{R} &= Det(\mathbf{M}) - K \times Trace(\mathbf{M})^2 \\ &= I_x^2I_y^2 - (I_xI_y)^2 - K \times (I_x + I_y)^2\end{aligned}$$

where K is an empirical constant ranging from 0.04 to 0.06.

Non-Maximal Suppression

To reduce the amount of corners detected, a corner should not be captured by more than one interest point. This objective can be achieved by non-maximal suppression which removes candidate points that are not the local maxima of **R** within its local neighborhood:

$$\mathbf{R}(\mathbf{x}) > \mathbf{R}(\mathbf{x}_w) \forall \mathbf{x}_w \in W \wedge \mathbf{R}(\mathbf{x}) > \mathit{threshold}$$

where W denotes the 8-neighborhood of the point x.

2.3.2 DOG Extrema Detector

DOG Extrema Detector is proposed by Lowe [32, 31] to detect SIFT features. It extracts interest points with a cascade filtering approach in which the more expensive operations are applied only at locations that pass all prior tests. The major steps of generating interest point from an image are discussed in the following sections.

Scale-Space Extrema Detection

DOG Extrema detection identifies the locations and scales of the interest point that can be repeatedly detected under different views of the same object. As the interest point can be repeatedly detected, we will call it stable features. Detecting stable features that are invariant to locations is achieved by searching for most of the locations over the image. To extend its invariance to scales, all possible scales of the image are searched instead of one scale only.

The scale space of an image which is defined as a function, $L(x, y, \sigma)$, can be prepared by repeatedly convolving the initial image with a variable-scale Gaussian function $G(x, y, \sigma)$:

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2} L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

To efficiently detect stable interest point locations in scale space, Lowe proposed [31] using scale-space extrema in the difference-of-Gaussian function, $D(x, y, \sigma)$, which can be computed from the difference of two nearby scales of smoothed images, $L(x, y, \sigma)$, separated by a multiplicative factor k.

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) = L(x, y, k\sigma) - L(x, y, \sigma)$$

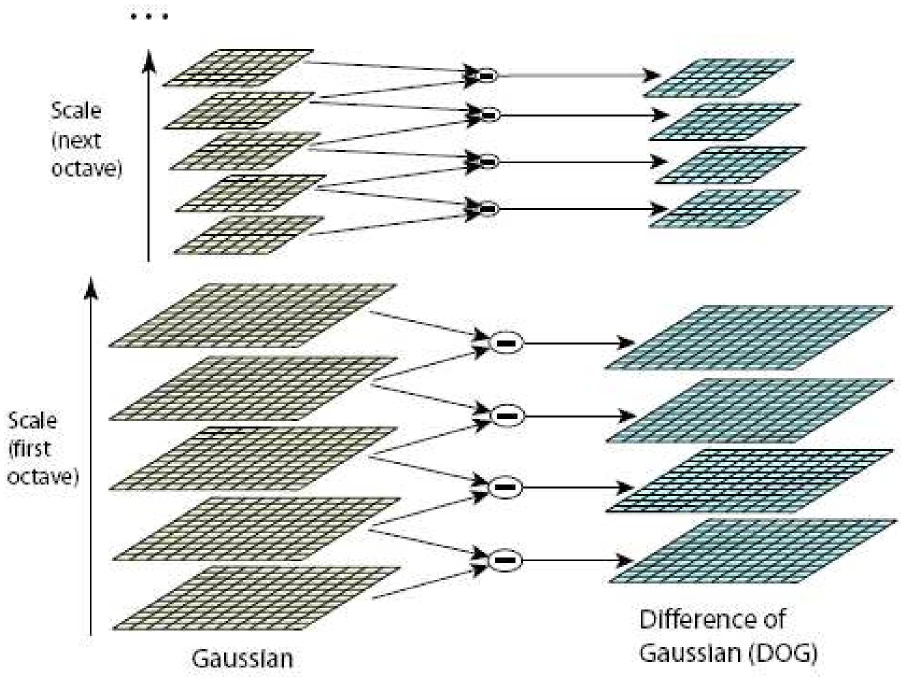


Figure 2.1: A diagram illustrating how differences of gaussian images are prepared from the initial image. The initial image is repeatedly smoothed by Gaussian function, which is shown on the left. Adjacent Gaussian images are subtracted to produce the difference-of-Gaussian images, which is shown on the right.

The scale space of the input image is prepared in the way illustrated by Figure 2.1. The difference-of-Gaussian function has been proved to be a close approximation to the scale-normalized Laplacian of Gaussian. Therefore, finding extrema in difference-of-Gaussian space is approximately equivalent to finding extrema in Laplacian space. After the scale space has been prepared, each sample point is compared to its eight neighbors in the current image and nine neighbors in the scale above and below in order to detect the extrema of $D(x, y, \sigma)$.

The advantage of searching interest points over a complete range of scales is that both small interest points and large interest points are detected. Small interest points help solving occlusion problem while large interest points contribute to the robustness of the system toward noise and image blur.

Interest Point Localization

The second step is to reject the interest points that have low contrast or are localized along an edge. Low contrast interest points are rejected because they are sensitive to noise. Interest points localized along an edge are also rejected because they in general do not make significant differences with nearby points.

To reject interest points with low contrast, the scale-space function value at each extremum, $D(\hat{x})$, is examined:

$$D(\hat{x}) = D + \frac{1}{2}\frac{\delta D}{\delta x}^T \hat{x}$$

For the experiments done by Lowe in [32], all extrema with a value of $|D(\hat{x})|$ less than 0.03 were discarded.

To reject interest points on edges, Hessian edge detector is applied. The difference-of-Gaussian function, D, will have a large principal curvature across the edge but a small one in the perpendicular direction. Hessian matrix, $\mathbf{H}$, can be computed at the location and scale of the interest point by:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

The derivatives, D_{xx}, D_{xy} and D_{yy}, can be estimated by taking differences of neighboring points around the sampling interest point.

The eigenvalues of $\mathbf{H}$ are proportional to the principal curvatures of D. Thus, the ratio of the two eigenvalues reflects whether the interest point is on the edge or not. The solution can be simplified by just checking the following condition:

$$\frac{Tr(\mathbf{H})^2}{Det(\mathbf{H})} < \frac{(r+1)^2}{r}$$

For the experiments done by Lowe in [32], all extrema having a ratio between the principal curvatures greater than 10 are discarded.

2.3.3 Harris-Laplacian Corner Detector

Mikolajczyk et al. [39] proposed another detector for detecting scale invariant interest points. It is called Harris-Laplacian corner detector. This detector first computes a set of images represented at different levels of resolutions (pyramid) for Harris corner detector. It then selects points at which the normalized Laplacian is maximal over scales. Mikolajczyk et al. observed that the amplitude of spatial image derivatives decreases with scale. Thus the derivative function must be normalized according to the scale of the observation. They modify the Harris corner detector such that it can be applied over the scale-space.

Auto-Correlation Matrix for Scale-Space

The detector analyzes the auto-correlation matrix $\mathbf{M}$ of every location in an image that is computed from normalized image derivatives:

$$\mathbf{M} = \sigma_D^2 g(\sigma_I) * \begin{bmatrix} I_x^2(\mathbf{x}, \sigma_D) & I_x I_y(\mathbf{x}, \sigma_D) \\ I_x I_y(\mathbf{x}, \sigma_D) & I_y^2(\mathbf{x}, \sigma_D) \end{bmatrix} \tag{2.2}$$

Equation 2.2 differs from Equation 2.1 by the differentiation scale σ_D. $I_x(\mathbf{x}, \sigma_D)$ and $I_y(\mathbf{x}, \sigma_D)$ represents the image derivative computed over an image obtained by convolving the full-size image with Gaussian kernels of scale σ_D. The image derivatives are normalized by multiplying with σ_D^2 that is proportional to the scale of the target image.

Scale Selection

After localizing points in 2D space using Harris corner detector, the candidate points are subjected to scale maxima detection. For each level of the scale-space, the detector applies the non-maximal suppression to reduce the amount of candidate points. Then for each of the candidate points found on different levels, it is verified if it is the maximum in Laplacian in the scale direction. The Laplacian $\mathbf{F}$ of a point $\mathbf{x}$ is defined by:

$$\mathbf{F}(\mathbf{x}, \sigma_D) = |\sigma_D^2(L_{xx}(\mathbf{x}, \sigma_D) + L_{yy}(\mathbf{x}, \sigma_D))|$$

Candidate point $\mathbf{x}$ at scale σ_{Dn} is the maximum in Laplacian in the scale direction if the following condition is satisfied:

$$\mathbf{F}(\mathbf{x}, \sigma_{Dn}) > \mathbf{F}(\mathbf{x}, \sigma_{Dn-1}) \wedge \mathbf{F}(\mathbf{x}, \sigma_{Dn}) > \mathbf{F}(\mathbf{x}, \sigma_{Dn+1})$$

where σ_{Dn-1} is a sampled scale just smaller than σ_{Dn} and σ_{Dn+1} is a sampled scale just larger than σ_{Dn}.

2.3.4 Harris-Affine Covariant Detector

Harris-affine covariant detector is an advance of the Harris-Laplacian detector. This detector can detect the same elliptical regions on images even if the object in the images is taken with significant different viewpoints. this makes feature description later in the recognition process invariant to changes of viewpoint. The detected regions are covariant to the affine transformation of object and thus this detector is called affine covariant detector.

Harris-affine covariant detector is based on affine normalization around Harris points. After a set of interest points are detected by Harris-Laplacian detector, iterative estimation of elliptical affine regions around the interest points are carried out. The estimation is done by determining the transformation that converts the interest region to the one with equal eigenvalues. The transformation can be computed by the square root of the auto-correlation matrix $\mathbf{M}^{1/2}$. Points $\mathbf{x}$ inside the interest region can then be normalized by the transformation:

$$\mathbf{x}' = \mathbf{M}^{1/2}\mathbf{x}$$

After projecting every point inside the interest region to a new position, the auto-correlation matrix is computed again and transformation of interest region to the one with equal eigenvalues is carried out again. This process proceeds until the auto-correlation matrix has equal eigenvalues. When all interest regions are normalized, corresponding regions differ only by a simple rotation. Thus, regions detected from an image are now invariant to the affine transformation.

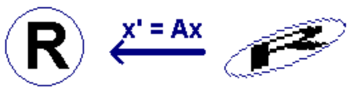


Figure 2.2: This figure shows an example of the elliptical affine region and the normalized region. The transformation matrix $A = M^{1/2}$ projects x to x' such that the eigenvalues of the auto-correlation matrix are equal.

2.4 Invariant Local Feature Descriptor

Given the interest points detected by the feature detector, the remaining task is to describe them for matching and recognition later. Distribution-based descriptors are shown [22] to be superior to other types of descriptors such as differential descriptors in recognition task. A distribution-based descriptor is a histogram representing in the form of a feature vector to capture the distribution of the image context such as pixel intensity, edge point, gradient location and orientation. In this section, five state-of-the-art descriptors are discussed. They are SIFT [31, 32], shape context [4], PCA-SIFT [45], GLOH [38] and GIH descriptors [29]. SIFT descriptor is a 3D histogram of gradient location and orientation direction. Shape context descriptor is a 2D histogram of edge points' locations. Schmid et al. [38] improved shape context to include also the distribution of orientations. PCA-SIFT descriptor is a vector of coefficients of the base image gradient patches obtained by PCA. GLOH descriptor is an extension of SIFT descriptor and is reduced in dimension by PCA. GIH is a geodesic-intensity histogram that is invariant to non-affine deformation.

2.4.1 Scale Invariant Feature Transform (SIFT)

The most important considerations of a feature descriptor are invariance and distinctiveness. SIFT descriptor is a carefully designed representation of image patch that is highly invariant to changes in scale, orientation and illumination, and is partially invariant to 3D viewpoints. SIFT descriptor is originally designed to use DOG extrema detector to detect interest points such that the descriptor is invariant to scale changes. SIFT descriptor allows feature positions to shift significantly without large changes in the descriptor and thus it can achieve partial invariance to affine distortion and changes in 3D viewpoints. Schmid et al. [38] further enhances its invariance to changes in 3D viewpoints by replacing the DOG extrema detector by harris-affine covariant detector. Although the average recall rate is lower, the descriptor showed significant improvement in detecting affine features under large affine distortion.

Orientation Assignment

For each interest point of each image sample $L(x, y)$ in a particular scale, the gradient magnitude $m(x, y)$ and orientation $\theta(x, y)$ are obtained using pixel differences:

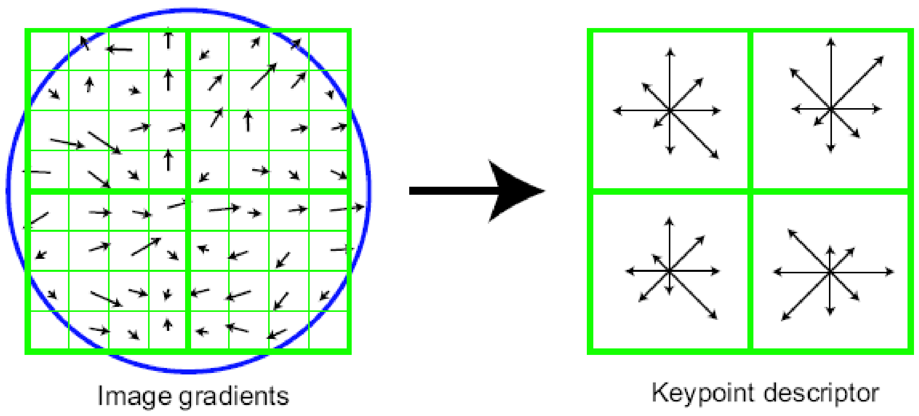


Figure 2.3: Computation of a feature descriptor based on the gradient and orientation of each image sample point in a region around the feature.

$$m(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2}$$
$$\theta(x, y) = \tan^{-1} \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)}$$

The gradient and orientation information of each interest point can then be used to construct the feature descriptor.

Descriptor Representation

The computation of the feature descriptor is illustrated in Figure 2.3. The approach is to create orientation histograms over 4×4 sample regions around the interest locations. Each histogram contains 8 orientation bins which is the Gaussian-weighted average of the gradient vectors over the corresponding region. For the case illustrated in Figure 2.3, a 32-element feature vector can be obtained for each interest point. Lowe has shown in experiments that a 4×4 array of histogram with 8 orientation bins in each would yield the best result. Since the orientation histograms are created over 4×4 regions instead of over every pixel, the descriptor is robust against significant changes in gradient position and thus it is partially invariant to changes in 3D viewpoints.

To make the descriptor further invariant to illumination changes, the descriptor is normalized to unit length. This totally cancels the effect of

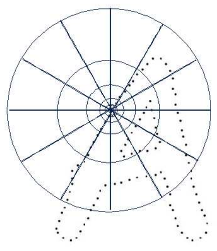


Figure 2.4: Figure illustrating how the bins of shape context is distributed around a given edge point. Belongie et al. [4] use five bins for quantizing distance between the rest of the edge points from the given edge point and 12 bins for quantizing the angle between them.

affine changes in illumination.

$$I(\mathbf{x}) = aI'(\mathbf{x}) + b \tag{2.3}$$

Equation 2.3 shows how an original pixel's intensity $I'(\mathbf{x})$ at position $\mathbf{x}$ is changed by affine illumination. Assume the constants a and b are the same within a small local region of an image, then the image derivative I_x will not be affected by the inter-reflection light term b. That is,

$$\widehat{I}(\mathbf{x}) = \frac{aI_x(\mathbf{x})}{a\sum_{x\in W} I_x(\mathbf{x})} = \frac{I_x(\mathbf{x})}{\sum_{x\in W} I_x(\mathbf{x})} \tag{2.4}$$

where W is the set of points within the concerned local region. Equation 2.4 showed that the image derivative does not depend on the constant a. Since the SIFT descriptor is created solely using image derivative, it is invariant to affine changes in illumination.

2.4.2 Shape Context

Shape context is a shape descriptor that describes the distribution of the rest of the shape with respect to a given edge points on the shape. It is a histogram of the relative positions of all other edge points in the image. Edge points here refer to a set of points sampled from the shape contours of the target object using edge detector. Shape context uses bins that are uniform in log-polar space to emphasize close-by, local structure as shown in Figures 2.4 and 2.5. In the original design of shape context, a histogram h_i is computed by simply counting the number of edge points within a bin:

$$h_i = \#\{q \neq p_i : (q - p_i \in bin(k)\}$$

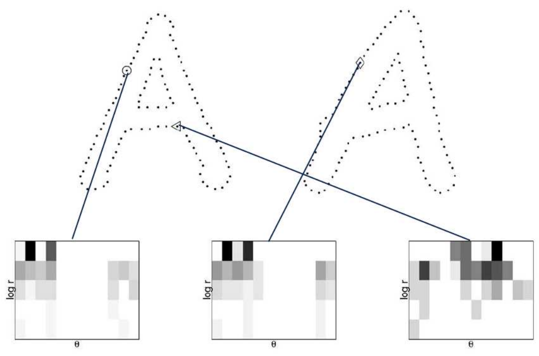


Figure 2.5: Figure showing the shape context histograms of three edge points. Darker bins indicate larger number of edge points are located inside the bins. The first and second histograms are very similar because the edge points they represent are similar while the third histogram is very different.

In the modified design by Schmid et al. [22], weight is assigned to the contribution of each point based on its gradient magnitude and orientation of edge points, which are also captured into the histogram. This makes shape context descriptor very similar to SIFT and GLOH descriptors.

Since shape context is a histogram computed from edge points, it is invariant to changes in illumination. To make shape context descriptor invariant to orientation, the feature detector has to help aligning the dominant orientation of the local patch to a canonical direction.

2.4.3 PCA-SIFT

PCA-SIFT descriptor is a vector of coefficients of the base image gradient patches obtained by PCA. It can be created in the following steps:

For each interest point,

1. Locate the 41×41 image patch around the point at the correct scale.
2. Rotate the patch to align its dominant orientation to a canonical direction in the same manner as SIFT.
3. Compute the Image gradients of the patch.
4. Create a vector by concatenating both horizontal and vertical gradient maps.

5. Normalize the vector to unit magnitude and make it invariant to changes in illumination.

6. Project the vector into a pre-computed eigenspace to derive a feature vector. The eigenspace can be pre-computed by applying PCA to the gradient patches in a set of training images.

Although creating PCA-SIFT descriptor is much simpler than creating SIFT, PCA-SIFT has been shown to have similar accuracy with SIFT in recognition [22] and run a lot faster than SIFT because of its much lower dimension. The success of PCA-SIFT lies in the fact that the patches surrounding the interest points all share some characteristics such as centering at the local extremum in the scale-space and orientated to the canonical direction. However, since the dimension of PCA-SIFT is very small (dim = 20), it is worthwhile to evaluate its performance when the database of features increases significantly.

As implied by the steps of creating PCA-SIFT, PCA-SIFT descriptor is invariant to orientation and changes in illumination in the same way as SIFT.

2.4.4 Gradient Location and Orientation Histogram (GLOH)

GLOH is an extension of the SIFT descriptor and is an advance version of PCA-SIFT and shape context. The same as SIFT, GLOH describes the gradient orientations of the image patches. Instead of sampling gradient orientations in a rectangular grid, GLOH samples them in a log-polor location grid like the one used in shape context descriptor. The histogram of each interest point consists of 17 location bins with 16 orientation bins in each. This gives a 272-bin histogram. PCA is then applied to reduce the dimension of GLOH descriptor to 128.

2.4.5 Geodesic-Intensity Histogram (GIH)

GIH is a novel local descriptor that is invariant to deformation based on the fact that the pixel intensity and geodesic distance are invariant to deformation. Geodesic distance is the distance of the shortest path between two points on the embedded surfaces. It is defined as:

$$d = \int_a^b \sqrt{(1-\alpha)^2 x_t^2 + (1-\alpha)^2 y_t^2 + \alpha^2 I_t^2 dt}$$

where a and b represent the coordinates of the two points on the embedded surfaces and the subscripts denote partial derivatives, e.g., $x_t = dx/dt$. Ling proved [29] that the geodesic distance of two points remains unchanged after deformation when $\alpha \longrightarrow \infty$. Geodesic distance for 1-D image is illustrated in Figure 2.6. GIH descriptor is created in the following steps:

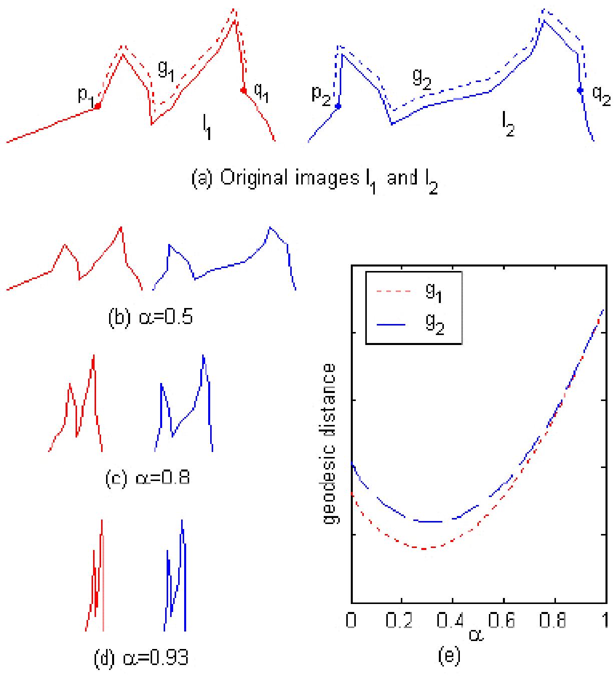


Figure 2.6: Deformation invariance for 1-D images (Figure from [29]).

For each interest point $p_0 = (x_0, y_0)$,

1. Apply fast marching algorithm to compute the points with identical geodesic distances from p_0 at intervals of δ. The aggregate of these points are called *level curve.*

2. Sample points from each level curve at intervals of δ.

3. Create a 2D intensity-geodesic distance space with intensity and geodesic distance as the two dimensions.

4. Insert all sampled points into the histogram according to its intensity and geodesic distance.

5. Normalize the geodesic distance dimension and then normalize the histogram as a whole.

Ling has made a good attempt to enhance local descriptor for deformation invariance. However, since images are defined on discrete grids, pixels in between two points can merge together to be a few pixels only. In this case, the discrete geodesic distance will vary a lot due to deformation. Refer to figure 2.7.

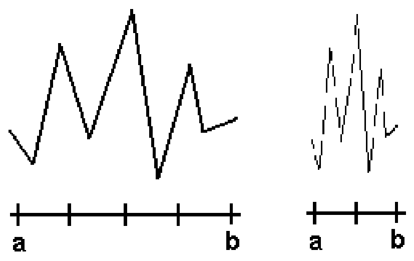


Figure 2.7: Figure illustrating discrete geodesic distance can fail. Due to discrete sampling of image pixels, the geodesic distance between points a and b is large in the image on the left and small in the deformed image on the right.

2.4.6 Experiment

In this experiment, we aim to compare the performances of the top three local descriptors: SIFT, PCA-SIFT and GLOH. In each experiment, each descriptor will describe both the Harris-affine covariant region and the Harris-Laplacian region. This allows us to compare the performance of Harris-affine covariant detector and Harris-Laplacian detector in matching at the same time. Our experiment only evaluates the accuracy but not the computational time of the local descriptors. At last, we will rank the descriptors based on the experiment we carried out.

Data Set

The data set used in our experiment is obtained from Visual Geometry Group[1]. We employed this data set to evaluate the performance of the four descriptors. Shape context is quite similar to GLOH and thus we evaluated the performance of GLOH only. For each set of images of the same scene, we selected two images as the image pair. The images we have used are shown in Figure 2.8

[1]http://www.robots.ox.ac.uk/ vgg/research/affine

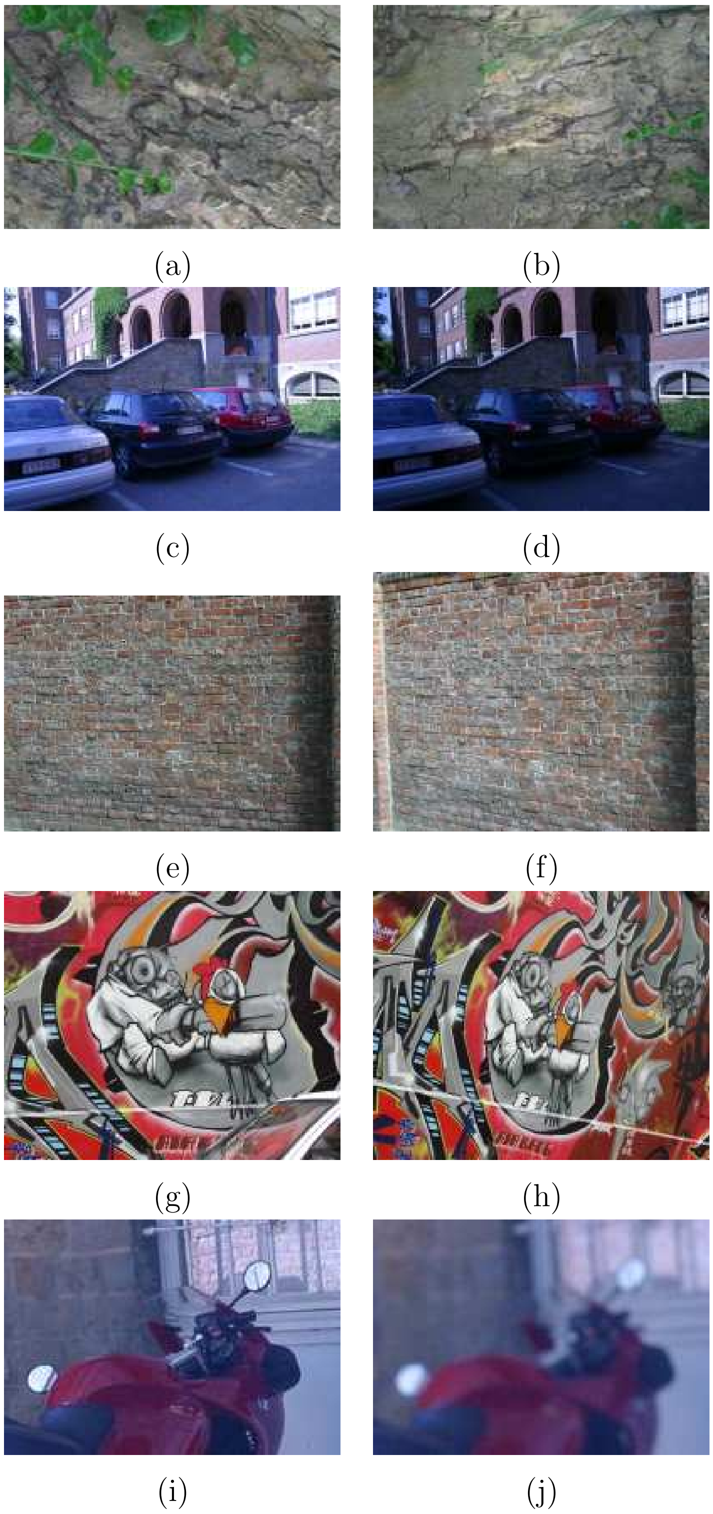


Figure 2.8: (a) & (b) Bark image sequence. (c) & (d) Leuven image sequence. (e) & (f) Wall image sequence. (g) & (h) Graf image sequence. (i) & (j) Bikes image sequence (A small portion).

Evaluation Criterion

We adopted the evaluation criterion used in [22, 45]. It counts the number of correct matches and the number of false matches obtained for an image pair. We want a local descriptor to have a large number of correct positives and a small number of false positives. Two regions are matched if the Euclidean distance between the two descriptors are below a threshold t. For the elliptical regions detected by Harris-affine covariant detectors, two regions are matched if the overlap error ϵ defined in [22, 40] is less than 0.4. If the match agrees with the ground truth, the match is classified as correct match; otherwise, it is false match. The transformation between each image pair can be described by a homography which can be used as the ground truth. The results are plotted in forms of recall versus 1-precision curves. The values of ϵ and t are varied to obtain the curves. Recall is defined as the number of correct matches over the number of possible correct matches in the image pair:

$$recall = \frac{\#correctmatches}{\#correspondences}$$

1-precision is defined as the number of false matches over the sum of the number of matches:

$$1 - precision = \frac{\#falsematches}{\#correctmatches + \#falsematches}$$

Experimental Results

Four common transformations in images are evaluated in this experiment. They are scale change and rotation, illumination change, viewpoint change and image blur. For each transformation, a recall versus 1-precision graph is plotted.

1. **Scale change and rotation.** We used the image pair shown in Figure 2.8(a),(b) to evaluate the performance for scale change and rotation. Result is shown in figure 2.9. As observed from the figure, description using Harris-affine covariant regions performs better than using non-affine covariant regions, and SIFT descriptor performs the best.

2. **Illumination change.** We used the image pair shown in Figure 2.8(c),(d) to evaluate the performance for illumination change. The result is shown in figure 2.10. These images are obtained by changing the camera setting likes exposure. As observed from the figure, all the descriptors under test are robust to illumination changes. The reason is that all of them use the same illumination normalization technique.

Nevertheless, we observed that SIFT descriptor remains the best among the three descriptors and Harris-Laplacian detector performs better than Harris-affine covariant detector. PCA-SIFT descriptor performs very well at high precision but the recall rate does not increase much when precision is less.

3. **Viewpoint change.** We used two image pairs shown in Figure 2.8(e),(f) and Figure 2.8(g),(h) to evaluate the performance for viewpoint changes. The result is shown in Figures 2.11 and 2.12. Viewpoint change in the wall image pair are less than those in the graf image pair. As observed from the figures, for the wall image pair, SIFT descriptor based on Harris-Laplacian detector performs the best, while for the graf image pair, SIFT descriptor based on Harris-affine covariant detector performs the best. This illustrates Lowe's SIFT descriptor itself is invariant to small amount of viewpoint changes and retains the highest distinctiveness. For high amount of viewpoint changes, performance of Lowe's SIFT descriptor drops significantly. However, when used with Harris-affine covariant detector, its performance is improved a lot. We observed that Harris-affine covariant detector really helps improve the robustness of descriptor. Yet this improvement may be limited only to cases with large viewpoint changes.

4. **Image blur.** We used the image pair shown in Figure 2.8(i),(j) to evaluate the performance for image blur. Blur effect is introduced to the image by adjusting the camera focus. The result of the experiment is shown in Figure 2.13. Both SIFT and PCA-SIFT descriptors perform well in this image pair. PCA-SIFT, again, performs very well at high precision but SIFT is better at lower precision.

Conclusion

From these experiments, we arrived at this conclusion: *SIFT > GLOH > PCA-SIFT*. SIFT descriptor is the best among the three descriptor in most of the cases. SIFT always performs slightly better than GLOH, so GLOH descriptor is only the second best. PCA-SIFT descriptor always performs very well at high precision requirement but not at lower precision so it is ranked the third. This fact does not depend on the types of interest regions.

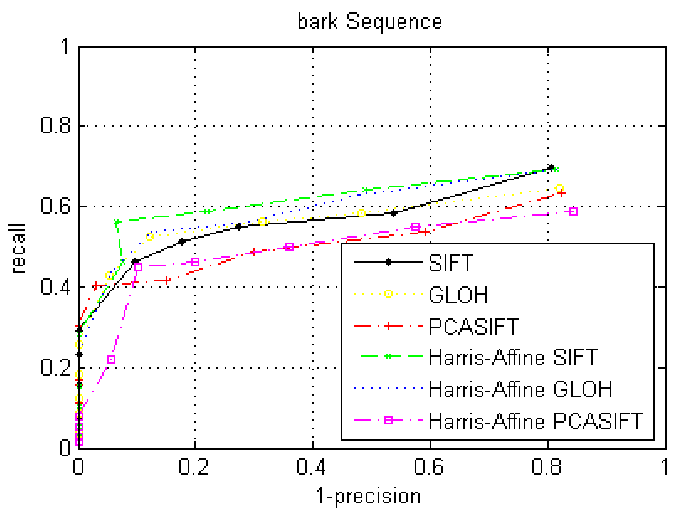


Figure 2.9: Experimental Result for scale change and rotation on bark image sequence.

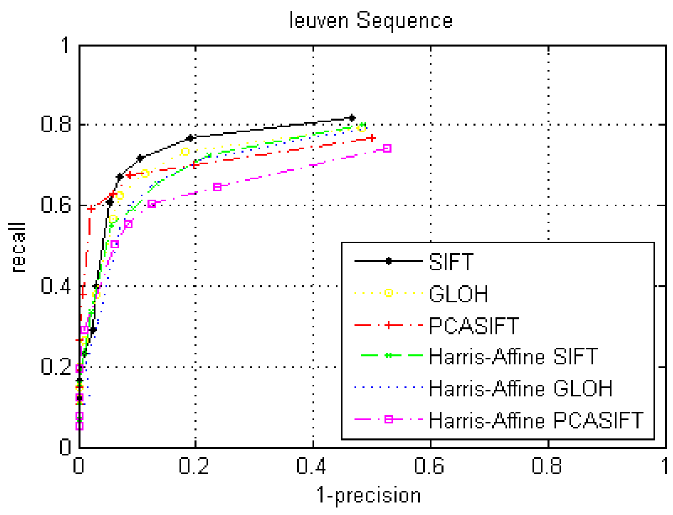


Figure 2.10: Experimental Result for illumination change on leuven image sequence.

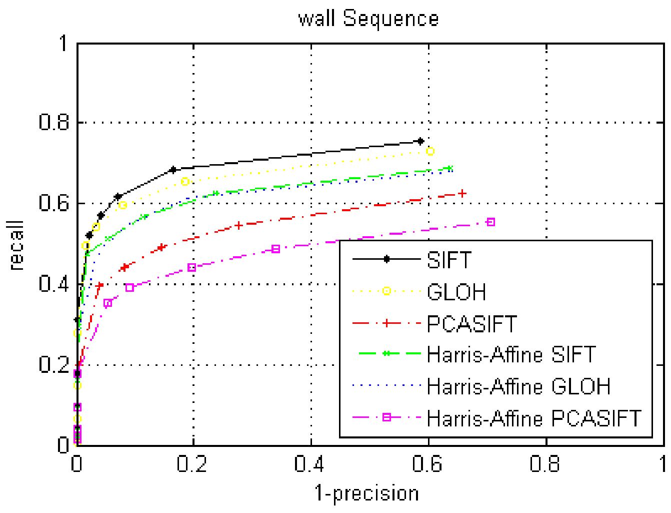


Figure 2.11: Experimental Result for viewpoint change on wall image sequence.

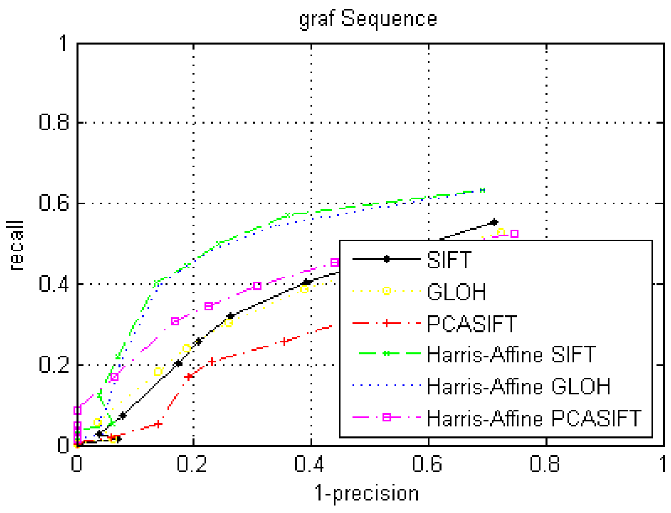


Figure 2.12: Experimental Result for viewpoint change on graf image sequence.

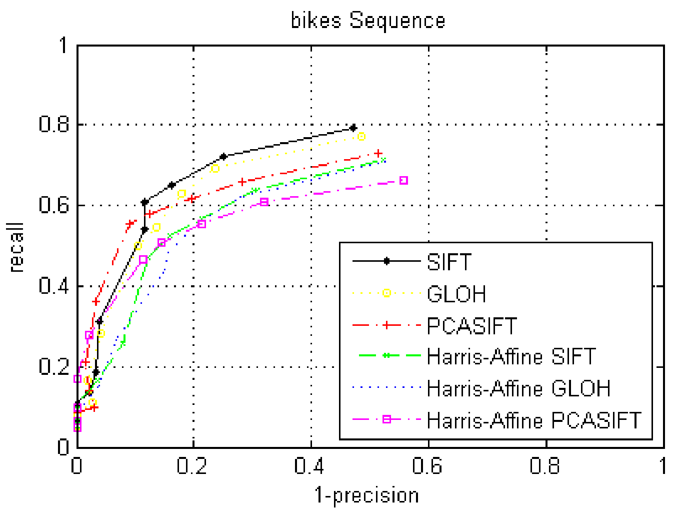


Figure 2.13: Experimental Result for image blur on bikes image sequence.

2.5 Feature Matching

A distribution-based descriptor represents local context in the form of a histograms. Thus, in comparing two descriptors, we can consider the distance measures commonly adopted in comparing two histograms. There are two main types of distance measures: bin-by-bin dissimilarity measures and cross-bin measures. Bin-by-bin dissimilarity measures only compare the contents of corresponding bins of two histogram while cross-bin measures also compare the non-corresponding bins. Recently, a cross-bin measure is proposed by Haibin Ling [30] and it is claimed that it significantly improves the original SIFT feature matching approach. In this section, we will introduce some of these distance measures, including those adopted in comparing the local descriptors. Then we will introduce some common feature matching techniques. We assumed there are two histograms: $X = (x_1, x_2, ..., x_n)$ and $Y = (y_1, y_2, ..., y_n)$. Histogram Y is one of the histogram stored in a database that histogram X will match with.

2.5.1 Matching Criteria

There are three common criteria in determining whether a feature *matches* with another feature:

1. Similarity Threshold. Two features are matched if the distance between the two features are below an absolute threshold. Each feature may have more than one match under this matching criterion.

2. Nearest Neighbor with threshold. Feature A *matches* with feature B in a database if B is the nearest neighbor of A among other features in the database and the distance between them is lower than a threshold.

3. Nearest Neighbor Distance Ratio. Feature A *matches* with feature B in a database if B is the nearest neighbor of A among other features in the database and the distance between them is lower than the distance between A and the second nearest neighbor in the database by a multiply constant. This criterion is shown to give higher precision to the above two methods in [22].

2.5.2 Distance Measures

Dissimilarity of two features is evaluated by measuring the distance between them.

Minkowski Distance

The Minkowski distance of order p (p-norm distance) is defined as:

$$d_p(X,Y) = (\sum_i |x_i - y_i|^p)^{\frac{1}{p}}$$

2-norm distance is the Euclidean distance. This is the most common distance measures used in comparing local descriptors. SIFT, GLOH, PCA-SIFT and GIH adopt this distance measure.

Histogram Intersection

Histogram intersection is defined as:

$$d(X,Y) = 1 - \frac{\sum_i min(x_i, y_i)}{\sum_i y_i}$$

For 2-D histogram, the distance is related to the area of intersection of two input histograms. The distance is normalized by the area of histogram Y. This distance measure is adopted by the color histogram proposed by Swain et al. [52].

χ^2 Statistic

χ^2 Statistic is defined by:

$$d(X,Y) = \sum_i \frac{(x_i - m_i)^2}{m_i}, m_i = \frac{x_i + y_i}{2}$$

This distance measure is adopted by Shape context descriptor.

Quadratic-form Distance

Quadratic-form distance is a cross-bin distance. Assume X and Y are histograms expressed in the form of column vectors. It is defined as follows:

$$d(X, Y) = (X - Y)^T A(X - Y)$$

where A is a similarity matrix $A = [a_{ij}]$ and a_{ij} is the similarity between bins i and j, which can be defined as:

$$a_{ij} = 1 - \frac{dist(i, j)}{dist_{max}}$$

This distance is commonly used in matching color histograms.

2.5.3 Searching Techniques

Exhaustive Search

Each feature in an image is matched with all features in another image or in the database. This is the most simplest method but it involves a brute-force computation of all distances and its complexity is very high.

k-D Tree

k-D tree is a binary space partition which recursively partitions the feature space at the mean in the dimension with the highest variance. k-D tree is a commonly used data structure for nearest neighbor query and range query. However, the performance of this structure is poor if the dimension of the data entries is high. A modified version called Best-Bin First tree is used by Lowe to match SIFT features.

Locality-Sensitive Hashing

Locality-Sensitive Hashing (LSH) is first proposed by Indyk & Motwani [19] and further improved in [6]. Locality-sensitive hashing is designed to provide a solution for high-dimensional near neighbor problem. It can answer queries in sublinear time with each near neighbor being reported with a fixed probability. This hashing scheme differs from other kinds of hashing in that under this scheme, the probability that two points share the same hash value decreases with the distance between them. This makes it suitable for feature matching purpose, in which features similar to the query feature should be returned.

According to [6], LSH family is defined as a family $H = \{h : S \to U\}$. If for any q, the function $p(t) = Pr[h(q) = h(v) : ||q - v||_2 = t]$ is strictly decreasing in t. That is, the probability of a collision of points q and v is decreasing with the distance between them. Then, for any points, q, v, u, with v within the ball of radius R centered at q while u is not, we would have $p(||q - v||_2) > p(||q - u||_2)$. In other words, we could hash the points from the dataset into some domain U, and at the query time compute the hash of q and consider only the points with which q collides.

To shorten the search time, the gap between the collision probabilities for the range $[0, R]$ and the range (R, ∞) is usually amplified. The commonly adopted method is to concatenate several functions $h \in H$. By concatenating k such functions, we obtain a function family $G = \{g : S \to U^k\}$ such that $g(v) = (h_1(v), ..., h_k(v))$, where $h_i \in H$. On the other hand, to increase the accuracy of near neighbor search, usually more than one such function family are used to hash the data points. Thus, usually LSH algorithm chooses L functions $g_1, ..., g_L$ from G, independently and uniformly at random. During the creation of the LSH hash tables, the algorithm stores each data point in the dataset into buckets $g_j(v)$, for all $j = 1, ..., L$. Then, during the processing of a query q, the algorithm searches all buckets $g_1(q), ..., g_L(q)$. For each point v found in a bucket, the algorithm computes the distance from q to v, and reports the points if and only if $||q - v||_2 \leq R$ where v is said to be the R-near neighbor.

E^2LSH (Exact Euclidean LSH) [1] is a package that implements the above LSH algorithm and can be used to solve the nearest neighbor problem. It can ensure point p satisfying $||q - p||_2 \leq R$ has to be reported with certain probability.

Under the implementation of E^2LSH, each hash function $h_{a,b}(v) : \Re^d \to Z$ maps a d dimensional vector v onto the set of integers. The hash function $h_{a,b}$ is given by $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor$ where a is a d dimensional vector with entries chosen independently from a Gaussian distribution and b is a real number chosen uniformly from the range $[0, w]$. In words, the hash function first projects each vector to the real line through the dot product $a \cdot v$. It then divides the real line into equi-width segments of appropriate size w and assign hash values to vectors based on which segment they project onto. It is shown that for any two vectors, say v_1 and v_2, the distance between their projections $(a \cdot v_1 - a \cdot v_2)$ is distributed as $||v_1 - v_2||_2 X$ where X is a random variable with Gaussian distribution. It follows that two vectors closer in Euclidean space would more likely share the same hash value.

□ **End of chapter.**

Chapter 3

A Distributed Scheme for Large-Scale CBIR

A long-standing challenge in content-based image retrieval (CBIR) is to develop a fast solution indexing high-dimensional image contents efficiently, which is crucial to building real-world large-scale CBIR systems. Traditional indexing schemes, which typically work well for low dimensional data, often suffer from the curse of the dimensionality problem when dealing with high-dimensional representations of image data. To attack this challenge, in this paper, we propose a parallel and distributed indexing scheme based on the locality sensitive hashing (LSH) technique, which is an effective high-dimensional indexing technique proposed recently. Our scheme not only exploits the powerful performance of LSH for high-dimensional data indexing, but also overcomes the scalability disadvantage of typical LSH solutions, ensuring that the overall indexing solution is scalable to real-world large-scale applications. We have conducted an extensive set of evaluations on a large CBIR testbed of over 1 million real-world images. The empirical results show that our proposed solution is effective and scalable to hundreds of thousands of images, which is promising for the development of large-scale Web CBIR systems.

3.1 Overview

The volume of multimedia data, particularly images and videos, has been increasing dramatically on the world wide web (WWW) due to the popularity of digital devices and personal computers. Effective retrieval of content from the huge volumes of media data in large-scale multimedia databases raises a number of critical challenges. In recent years, multimedia information retrieval has attracted more and more attention in the research community.

One of the most popular and fundamental research topics is content-based image retrieval (CBIR) [27], [50]. Although CBIR has been extensively studied in both academia and industry for many years, a number of challenging issues, still lack effective solutions. These issues are often related to long-standing challenges among several interdisciplinary research areas, including database searching, computer vision, information retrieval, and machine learning.

In general, the development of a CBIR system consists of at least four stages: data acquisition and processing, feature representation, data indexing, query processing, and feedback processing. The first stage is to acquire the images by either collating photos from users or trawling existing images from the WWW. The feature representation stage studies techniques of low-level feature extraction from image pixels and the application of effective similarity measures. The data indexing stage studies techniques for indexing low-level features to facilitate query processing. The last stage is to respond to users' queries and process the relevance feedback interaction based on the indexing system constructed.

For all these stages, an efficient data indexing system is critical to making the CBIR system scalable to large-scale real-world applications. Although various data indexing techniques have been well studied in the database community, traditional indexing solutions usually work well only for low dimensional data. They often suffer from the curse of the dimensionality problem when handling high dimensional data [19]. Since images are usually represented in high dimensional feature spaces, applying traditional indexing techniques to CBIR may not effectively solve the indexing problem, particularly for large-scale applications. Therefore, it is imperative to developing an efficient indexing solution that can deal with high dimensional data efficiently and scale well to large-scale data.

In this paper, we propose a parallel and distributed indexing scheme for building scalable CBIR systems. The proposed indexing solution applies an emerging indexing technique, locality-sensitive hashing (LHS) [19, 6], which enjoys some significant advantages for indexing high dimensional data. Our parallel and distributed indexing scheme enables our CBIR system to adapt to large-scale applications efficiently. In summary, our contributions in this paper include: (1) a study of LSH techniques as applied to CBIR for the purpose of overcoming the high dimensional indexing challenge; (2) a novel parallel and distributed indexing scheme with LSH for large-scale high dimensional indexing problems; (3) a comprehensive evaluation of large-scale pure CBIR systems with over 1 million images.

The rest of this paper is organized as follows. Section 3.2 reviews some related work. Section 3.3 presents our proposed scalable CBIR solution with

LSH. Section 3.4 discusses our feature representation methods. Section 5.4 gives our evaluations on a large-scale image testbed. Detailed experimental results are presented to show that our distributed solution is scalable to large-scale data. Section 3.6 presents a fast, online CBIR web application prototype which is built based on the proposed distributed solution. Section 3.7 sets out our summary.

3.2 Related Work

Content-based image retrieval has been a very active research topic in the multimedia community since the early 1990s [27, 50, 47]. Many CBIR prototype systems have been built and studied in both industry and academia. Some famous systems include IBM's QBIC [41], Virage [2], RetrievalWare [7], MIT's Photobook [42], VisualSEEK [51], and MARS [35], VIMA [24], etc. Some of the earlier work focused on the feature extraction methods by applying image processing techniques to extract useful features, such as color [20], texture [56], shape [36], etc. Since there is a well-known semantic gap between an image's low-level features and the high-level semantic concepts represented in the image, relevance feedback in which a human user evaluates the relevance of images selected by searching algorithm is one of the most popular research topics recently in CBIR [53]. Many previous papers have applied various machine learning techniques and often demonstrated good results on small scale testbeds. In addition, there has been an amount of research effort focusing on distance metric learning [14, 15] and log-based image retrieval [16] in recent years. However, most previous work in CBIR has employed relatively small image testbeds, usually in a scale of several thousand images, which may not reflect their real-world performance. One of the reasons for this limitation is the lack of efficient indexing solutions for fast retrieval over large-scale image databases.

In fact, the study of data indexing techniques is not new to CBIR researchers [47, 54]. A variety of traditional multidimensional indexing techniques, such as k-d tree, quad-tree, and R-tree [12] and its variants R^+-tree and R^*-tree [3], have been applied to CBIR. A comprehensive review and comparison of these traditional indexing techniques in CBIR can be found in [54]. However, traditional indexing techniques often suffer from the curse of the dimensionality problem in dealing with high dimensional data. For example, as shown in a previous empirical study [54], a well-known k-d tree indexing method is usually not better than a simple Euclidean measure when the number of dimensions is greater than 20. Since images in CBIR are often represented in a high dimensional feature space, retrieval would be a difficult task if traditional indexing techniques are applied in [25, 28]. Until now, there have been still few sophisticated solution is proposed for indexing large-scale images in CBIR. Indeed, some large-scale CBIR applications have to engage keywords-based indexing to facilitate search tasks [44].

Recently, a novel emerging indexing technique, locality sensitive hashing, has emerged and has been proposed as a means of attack on the high dimensional data indexing problems. LSH enjoys several of advantages for high-dimensional data indexing. M. Datar et al. gave some empirical results to show that LSH searches for near neighbors 30 times faster than the k-d tree does when the dimensionality of the dataset goes beyond 200 [6]. Recently LSH has attracted more and more attention in various applications, such as video retrieval [18] and image copy detection [43]. However, there has been, as yet, little comprehensive study of the application of LSH to solve CBIR applications [55]. A straightforward application of LSH to CBIR may generate several problems, which hinder its scalability to large-scale applications. Our work is focused on applying LSH to CBIR and solving the scalability problem through a novel parallel and distributed scheme.

3.3 Scalable Content-Based Image Retrieval Scheme

3.3.1 Overview of Our Solution

In order to take advantages of the powerful performance of LSH for high dimensional indexing, we propose a scalable content-based image retrieval scheme based on a parallel and distributed indexing solution using LSH techniques. Our proposed indexing scheme overcomes the scalability issue when applying LSH directly to large-scale applications, making the developed CBIR systems capable of adequate real-time performance in large-scale applications. In the following discussion, we first introduce LSH in terms of its advantages of indexing high dimensional data, and then discuss some difficulties when applying it directly to large-scale applications. To solve these problems, we then present the proposed solution.

3.3.2 Locality-Sensitive Hashing

Locality-sensitive hashing, an emerging new indexing algorithm, has recently been proposed to solve high-dimensional near neighbor searching problems in Euclidean space l_2, and is therefore very suitable for indexing high dimensional data in CBIR applications. It was first proposed by Indyk &

Motwani [19] and further improved in [6]. Like any other traditional hashing algorithm, LSH employs a hash function to map a query point to a hash bucket at which the target data point can be found. However, in contrast to the traditional algorithms, LSH's hash functions are locality sensitive, which means the probability that two points share the same hash value decreases when the distance between them increases. This property makes LSH useful in solving the near neighbors problem. By using LSH, given a query point, we can obtain its near neighbors simply by computing its hash value, locating the hash bucket, and then examining the points inside the hash bucket. Typically, LSH can answer queries in sublinear time, with each near neighbor being reported with a fixed probability. This enables us to build large-scale scalable CBIR systems.

For building a scalable system, we employ the E^2LSH (Exact Euclidean LSH) package [1], an implementation of the LSH algorithm that can solve the nearest neighbor problem. Under the implementation of E^2LSH, each hash function $h_{a,b}(v) : \Re^d \rightarrow Z$ maps a d dimensional vector v onto a set of integers. The hash function $h_{a,b}$ is given by $h_{a,b}(v) = \lfloor \frac{a \cdot v + b}{w} \rfloor$ where a is a d dimensional vector with entries chosen independently from a Gaussian distribution and b is a real number chosen uniformly from the range $[0, w]$. In other words, the real line is first divided into equal segments of appropriate size w. The hash function then projects the query feature vector to the real line through the dot product $a \cdot v$ and assigns a hash value to the vector based on which segment it is projected onto. It is shown that for any two vectors, say v_1 and v_2, the distance between their projections $(a \cdot v_1 - a \cdot v_2)$ is distributed as $||v_1 - v_2||_2 X$ where X is a random variable with Gaussian distribution. It follows that two vectors closer in Euclidean space would be more likely to share the same hash value.

The probability of finding a certain near neighbor can be controlled by two parameters, L and k. Parameter L controls the number of hash tables to be built. Each hash table contains an independent set of hash buckets located by a different set of hash functions. A larger value L increases the probability of finding all near neighbors. Parameter k controls the number of hash functions to be used together to locate a hash bucket. A larger k reduces the chance of hitting data points in a hash bucket that are not the near neighbors of the query point.

3.3.3 Scalable Indexing Solutions

The application of LSH for solving large-scale CBIR is not straightforward. One major problem is that E^2LSH is a main memory based implementation.

[1]http://web.mit.edu/andoni/www/LSH/index.html

Although it can answer a given query very efficiently, it requires all the data points, the R-near neighbor (R-NN) data structures, and the LSH hash tables to be stored in main memory for fast memory access. If the total demand for main memory is larger than the size of the main memory, swap space of the disk will be used. In that case E^2LSH may not answer a query efficiently because disk swapping may happen during the query. Therefore, the maximum database size E^2LSH can handle at the same time is limited by the the amount of free main memory available in the machine. This bottleneck limits its direct application for large-scale databases.

To make our CBIR system scalable to large-scale data, we propose two multi-partition indexing approaches to tackle the above issue: *disk-based multi-partition indexing* and *parallel multi-partition indexing*. Both of these approaches divide the whole database into multiple partitions. Each partition is associated with a *partition structure*, which consists of a corresponding subset of data points in the database, R-NN data structures, and LSH hash tables for that subset of data points. In order to fit the partition structure into the main memory, the size of each partition structure is set carefully. Since E^2LSH requires some time to compute the R-NN data structures and the LSH hash tables, very partition structure newly computed by our system is saved to disk. By doing so, when our system has to process queries in a particular partition, it can obtain that partition quickly by loading its associated structure from the disk into the main memory.

The approach described above employs one machine to provide the indexing functionality. That machine loads one partition structure at a time from disk into the main memory and processes the query on it. These two steps are done sequentially until the query has been processed on all partitions. However, an approach is available, employing multiple machines. In this approach, each machine serves requests over only one partition of the database. Thus, the number of the machines required is equal to the number of partitions of the indexing database. Both of these two indexing approaches are further discussed below.

3.3.4 Disk-Based Multi-Partition Indexing

The detailed procedure for using this approach to process a query is shown in Algorithm 1.

This solution is able to handle very large-scale databases without any size limit. It is useful for building off-line CBIR applications, particularly when there are only a few machines available. However, there are still some critical issues that need to be addressed in order to apply the above solution to huge web-scale applications. One main limitation of the above approach

Algorithm 1 A Disk-Based Multi-Partition Indexing Scheme

```
1: Divide the database into $n$ partitions, where the number of partitions
   $n = \lceil \frac{database\ size}{max.\ partition's\ size} \rceil$.
2: Run $E^2LSH$ indexing on each of the partitions to create the R-NN data
   structures and hash tables.
3: Save all partition structures on the disk.
4: for a query $q$ or a set of queries $qS$ do
5:     for each partition structure $s_i$ do
6:         Load $s_i$ (including the data points, the R-NN data structures, and
           the LSH hash tables) into the main memory
7:         Run $E^2LSH$ query on $s_i$ to retrieve the top $k$ ranking images with
           respect to $q$ or each query in the set $qS$
8:     end for
9:     Collect the results for all partitions and return the top $k$ ranking results
       with respect to the query $q$ or each query in the set $qS$.
10: end for
```

is the disk-access overhead for loading the partition structures into the main memory. When the database size is very large, the disk-access time will be a critical problem. As will be shown in section 5.4, the total time required to load all the five partition structures of a 0.5 million image database into the main memory is up to 750 seconds. This clearly cannot satisfy real-time applications. To solve this problem, we propose the parallel multi-partition indexing approach, which overcomes the disk-access overhead and expedites the overall process at the same time.

3.3.5 Parallel Multi-Partition Indexing

In this approach, the query is processed by a parallel and distributed system. The system consists of three main components: *Frontend*, *Master*, and *Slave*. The frontend is the interface between the query processing module and the client machine. The master is the coordinator which acts as a communication bridge between the frontend and the slave. The slave is the core of the system, which performs the LSH search upon request.

In contrast to the previous approach, this approach distributes the data indexing tasks over a cluster of *Slave* machines. Each Slave in the cluster is responsible for answering user queries over a designated portion of the database, which is assigned by a Master machine during the initialization stage. The Slaves store their corresponding partition structures in their main memory spaces permanently. This ensures that no re-loading of the partition structures is required after the initialization stage. In other words, this approach avoids the disk-access overhead problem that exists in the previous approach.

The Slaves are managed by the Master machine. The Master listens to the query requests from the Frontend machines and then forward the requests to the Slaves. Query answers from the Slaves are merged together in the Master and the top k ranking images among the merged results are returned to the Frontend.

The system setup and query processing procedures of this approach are given in Algorithm 2.

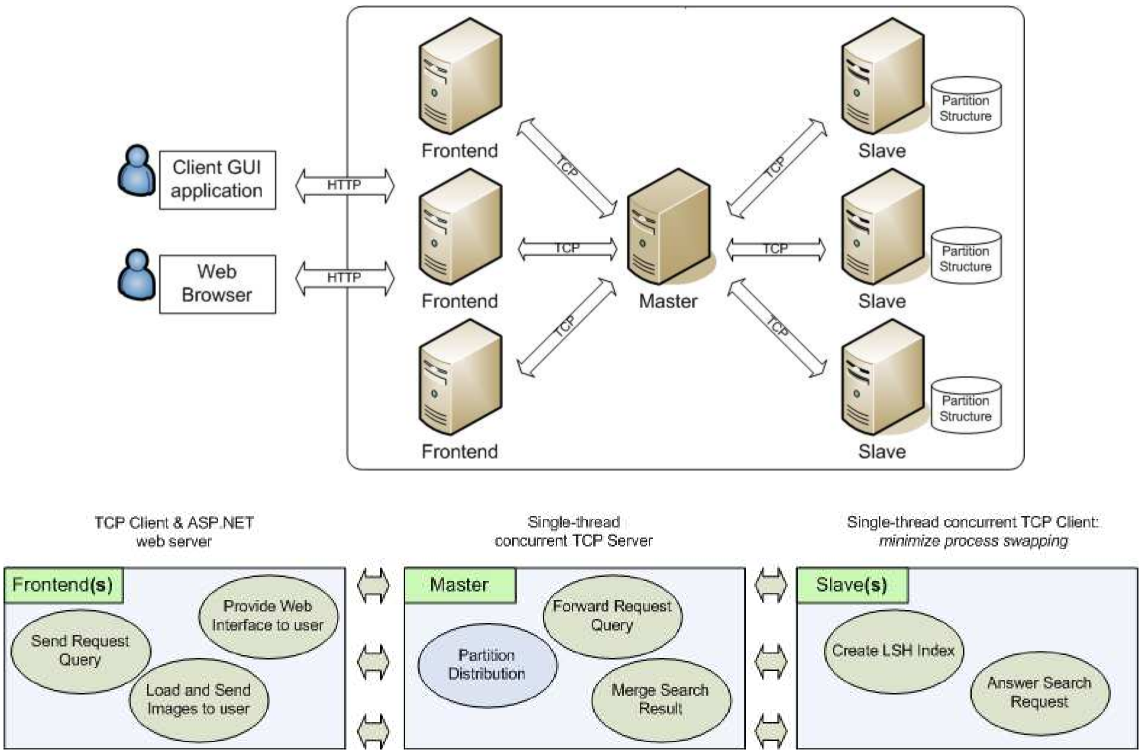


Figure 3.1: The system architecture of our proposed parallel and distributed indexing and retrieval scheme

System Architecture

Figure 3.1 shows the system architecture of the parallel multi-partition indexing approach. The Master is a single-thread concurrent TCP server that listens to two ports: a TCP port that receives Frontends' queries, and a

TCP port that receives Slaves' answers. The task of the Master is to forward requests from the Frontend to the Slaves, merge the replies from the Slaves, and then send the result back to the requesting Frontend. Since the top k ranking images from the Slaves have already been sorted in similarity values by the Slaves before replying, the replies from the Slaves can be quickly merged in the Master by an $O(n)$ merging algorithm. Thus, the workload of the Master is light. However, it is better to separate the task of the Master from the Frontend and the Slaves, because the workload of the Frontend and the Slaves is usually high. Moreover, when we engage more Frontends and Slaves, the workload of the Master will increase. If necessary, we can consider using multiple Masters or a hierarchy of Masters so as to improve the responsiveness of the system.

The Slave is a single-thread TCP client that operates the indexing and retrieval tasks. Since the indexing service is a CPU-intensive task, we implement the Slave as a single-thread process and make the Slave serve only one query at any time so that time is saved from thread swapping or process swapping during query processing.

The system can have many Frontends. The Frontend is a TCP client during empirical evaluation and is also an ASP.NET web server during system deployment. When the system is deployed, the Frontend will have to serve hundreds of users at a time. The Frontend is responsible for accepting HTTP requests, serving HTTP responses, loading images from disks, and displaying images to users. Thus, the workload of the Frontend is heavy and multiple Frontends are required to improve the responsiveness of our system.

Procedures and Time Performance Analysis

As shown in Algorithm 2, there are two phases in building the CBIR system, i.e., *system setup* and *query processing*. Since the system setup is not critical in terms of time performance, we will analyze only the time performance of the query processing procedure below.

Our system is distributed over a network of machines, thus there are inevitably some network transmission delays in the communications between the Frontends, the Master, and the Slaves. To minimize the adverse effect of the network transmission delays, we design several ways to maximize the efficiency of the communications such that the round-trip-time (RTT) between the Frontend and the Master, and that between the Master and the Slave, are negligibly small. Firstly, in the query and reply messages transmitted between the machines, we represent each image that also exists in the image database by a 4-byte integer image ID number, instead of a long feature vector or a large image format file. The image ID number can be

Algorithm 2 A Parallel and Distributed Multi-Partition Indexing and Retrieval Scheme

```
 1: Start **PHASE 1:** *System Setup*
 2: Set the Master up. Path to the database file and the maximum partition's size
    are specified.
 3: Set the Slaves up. The Slaves register with the Master and wait for job assign-
    ment.
 4: The Master calculates the number of Slaves required and the range of each
    partition. Then it assigns the partitions to the available Slaves.
 5: for each slave machine $Slave_i$ do
 6:    if $Slave_i$ found a partition structure in disk then
 7:       $Slave_i$ loads the partition structure into the main memory.
 8:    else
 9:       $Slave_i$ runs $E^2LSH$ indexing on its assigned partition to create partition
          structure in the main memory and then saves the structure on disk.
10:    end if
11: end for
12: Set the Frontends up. The Frontends register at Master and wait for user
    commands.
13:
14: Start **PHASE 2:** *Query Processing*
15: The Frontend presents a small set of random images obtained from the image
    database to the user.
16: The user can either upload her own image or select one of the provided im-
    ages and then request the Frontend to perform top $k$ similarity search on the
    specified image.
17: if the user selects one of the images provided by the Frontend then
18:    The Frontend attaches the pre-computed feature vector of the query image
       to a query message.
19: else
20:    The Frontend computes the feature vector of the query image provided by
       the user and attaches it to a query message.
21: end if
22: The Frontend sends the query message to the Master.
23: The Master duplicates and distributes the query message to the Slaves.
24: The Slaves run $E^2LSH$ query on their partition structures to retrieve the top
    $k$ ranking images with respect to the query image and then send the results
    back to the Master.
25: The Master collects and merges the results from the Slaves. It then attaches
    the top $k$ ranking results with respect to the query image to a reply message
    and sends the reply message to the Frontend.
26: Upon receiving the reply message, the Frontend loads the images from disk
    according to the image references given in the reported ranking results and
    displays them to the user.
```

set as the index of the corresponding image in the image database. Secondly, we represent each of the query and reply message by a *Structure* in C/C++. Since most of the data represented in the query and reply message are either floating-point numbers or large integers, representing these data as binary data in the form of a *Structure* is both accurate and space-saving. In this form, the query message passed from the Frontend to the Master and from the Master to the Slave can esaily fit into one 1500-byte TCP packet, even if it represents a 238-dimensional query feature vector and up to 120 positive/negative relevance feedback image ID numbers. The reply message which is passed from the Slave to the Master and the Master to the Frontend can also fit into one TCP packet, with the capability of containing up to the top 180 ranking image ID numbers. Since both the query and reply message can be transmitted in only one TCP packet each, communications between the Frontend, the Master, and the Slave are fast and efficient. In our system, we employ TCP in all communication channels to ensure the packet delivery is reliable. Since all the connections are persistent throughout the system lifetime, during query processing, no time is needed for the time-consuming TCP connection initialization step. As a result, the communication channels are both reliable and efficient.

In general, the Slave can finish a query request and reply to the Master within a second. However, in some situations, the Slaves may not be able to reply requests immediately. For example, the Slave may be delayed by network problems or other exceptions. To offer some quality of service to prevent users from long delays, the Master will timeout any Slave that cannot reply to the request within a period of time, e.g. 1 second. Therefore, the total system's query processing time (T_{Total}) can be expressed as the following formula:

$$\begin{aligned} T_{Total} = T_{Frontend} + RTT + T_{Master} + RTT + \\ \min(\max(T_{Slave_1}, ..., T_{Slave_N}), T_{Timeout}) \end{aligned} \tag{3.1}$$

where T_M represents the processing time of machine M, $Slave_i$ is the ith Slave machine, $T_{Timeout}$ is the maximal time that the Master will wait for the reply from a Slave, and RTT, the round-trip-time, is defined as

$$RTT \approx RTT_{Frontend \leftrightarrow Master} \approx RTT_{Master \leftrightarrow Slave}$$

From our experimental experience, the RTTs between Frontend and Master $RTT_{Frontend \leftrightarrow Master}$, and that between Master and Slave $RTT_{Master \leftrightarrow Slave}$ are as small as $2ms$ in a $100Mbps$ local area network. These two RTTs, together with the processing time of the Master, accounts for only 8% of the total query processing time when each Slave is indexing 0.25 million data points. Most of the query processing time is spent on running the LSH query on the Slaves.

Disk-Loading Issues for Relevance Feedback

To perform relevance feedback, each Slave needs a way to obtain the feature vectors of the images specified in the positive relevance feedback image ID list of the query message. The way we adopt is to fetch the feature vectors of the specified images directly from the disks upon request. Since the number of image feature vectors required by each LSH query is small (at most 120 feature vectors), the disk-access overhead is relatively low. This approach has some immediate advantages:

(a) It relieves the necessity of having feature vectors consuming precious main memory space so that more space can be used to contain as much partition structure as possible. This minimizes the number of Slave machines needed.

(b) It relieves the necessity of loading all the data points of the dataset into the main memory, which is not scalable due to the limit on the size of the main memory of the Slave.

(c) It relieves the necessity of transmitting the feature vectors of the positive relevance feedback images between the machines, which would increase the network load.

Furthermore, the disk loading of feature vectors can be accelerated by means of two tricks:

(a) By saving the database of feature vectors in binary format. Since each feature vector occupies the same number of bytes i.e. $(238 \times 4 =)$ 952 bytes, a feature vector with a known corresponding image ID can be fetched quickly by offsetting $952 \times ImageID$ bytes.

(b) By sorting the positive relevance feedback image IDs in ascending/descending order. This means that the hard disk head can read all of the specified feature vectors by moving in a single direction one time only. This reduction in disk head motion can effectively shorten the disk seek time.

Major Advantages of the Parallel Indexing Approach

The followings are some advantages of the parallel multi-partition indexing approach over the disk-loading based approach:

(a) **No Disk-Access Overhead.** Since the whole database is distributed over multiple Slaves, in each of which partition of the database fits in main memory, Slaves can store the partition permanently in memory and thus no disk-access overhead is required during query processing. This is critical to ensuring the real-time query performance of our system.

(b) **Significant Speedup by Parallelization.** We employ the parallelization technique to reduce the query time significantly. Of course, the acceleration depends on the available hardware resources.

(c) **Guaranteed Time Performance.** Since each Slave handles a portion of data points below a fixed maximum size, the query response time of each Slave is usually smaller than a fixed value. This may enable our proposed CBIR system to deliver adepquate quality of service (QoS) in real-world applications.

Remark. Although the above scheme is promising, there are still some open issues to be studied in future work. For example, for the parallelization of the indexing scheme, how to equalize the retrieval time in each partition and minimize the overall query time is an open problem. Moreover, we currently simply divide the data randomly into multiple partitions. In future work, we may study some data-sensitive partition techniques to facilitate the indexing procedure.

3.4 Feature Representation

Feature representation is a key step for building CBIR systems. In our solution, three types of visual features are engaged: color, shape and texture [16].

For color, we use the grid color moment. Each image is partitioned into 3×3 grids and three types of color moments are extracted for representing color content of each grid. Thus, an 81-dimensional color moment is adopted for the color feature.

For shape, we employ the edge direction histogram. A Canny edge detector is used to acquire the edge images and then the edge direction histogram is computed from the edges. Each histogram is quantized into 36 bins of 10 degrees each. An additional bin is used to count the number of pixels without edge information. Hence, a 37-dimensional edge direction histogram is used for the shape feature.

For texture, we adopt the Gabor feature [34, 56]. Each image is scaled to 64×64. Gabor wavelet transformation is applied on the scaled image with 5 scale levels and 8 orientations, which results in 40 subimages. For each subimage, three moments are computed: mean, variance, and skewness. Thus, a 120-dimensional feature vector is adopted for the texture feature.

In total, a 238-dimensional feature vector is employed to represent each image in our image database.

3.5 Empirical Evaluation

3.5.1 Experimental Testbed

To examine the scalability of our proposed scheme, we have collected a testbed containing $1,000,000$ images trawled from WWW. Further, to enable an automatic objective evaluation, an additional set of $5,000$ images from COREL image CDs are engaged as the query set for performance evaluation. This image set contains 50 categories. Each category in the dataset consists of exactly 100 images that are randomly selected from relevant examples in the COREL image CDs. Every category represents a different semantic topic, such as *antique*, *antelope*, *aviation*, *balloon*, *botany*, *butterfly*, *car*, *cat*, *dog*, *firework*, *horse* and *lizard*.

The motivation for selecting the COREL images in semantic categories for evaluation is twofold. First, it allows us to evaluate whether the proposed approach is able to retrieve images that are not only visually relevant but also semantically similar. Second, it allows us to evaluate the retrieval performance automatically, which significantly reduces the subjective errors arising from manual evaluations. This evaluation methodology has been widely adopted in previous CBIR research [16].

3.5.2 Performance Evaluation Metrics

We evaluate the retrieval performance of our CBIR system based on two standard performance metrics: *precision* and *recall*. The *precision* is calculated as the proportion of relevant images in the set of returned images, while the *recall* is calculated as the ratio of the number of relevant images in the returned images to the total number of relevant images in the database. The returned image is judged as being relevant with respect to a query image if it is one of the 100 COREL images in the database which share the same category as the query image. Since each selected COREL category contains 100 images only, we assume that the total number of relevant images in the database for each query is equal to 100. To evaluate the efficiency of our CBIR system, we measure the average CPU time elapsed for a given query.

3.5.3 Experimental Setup

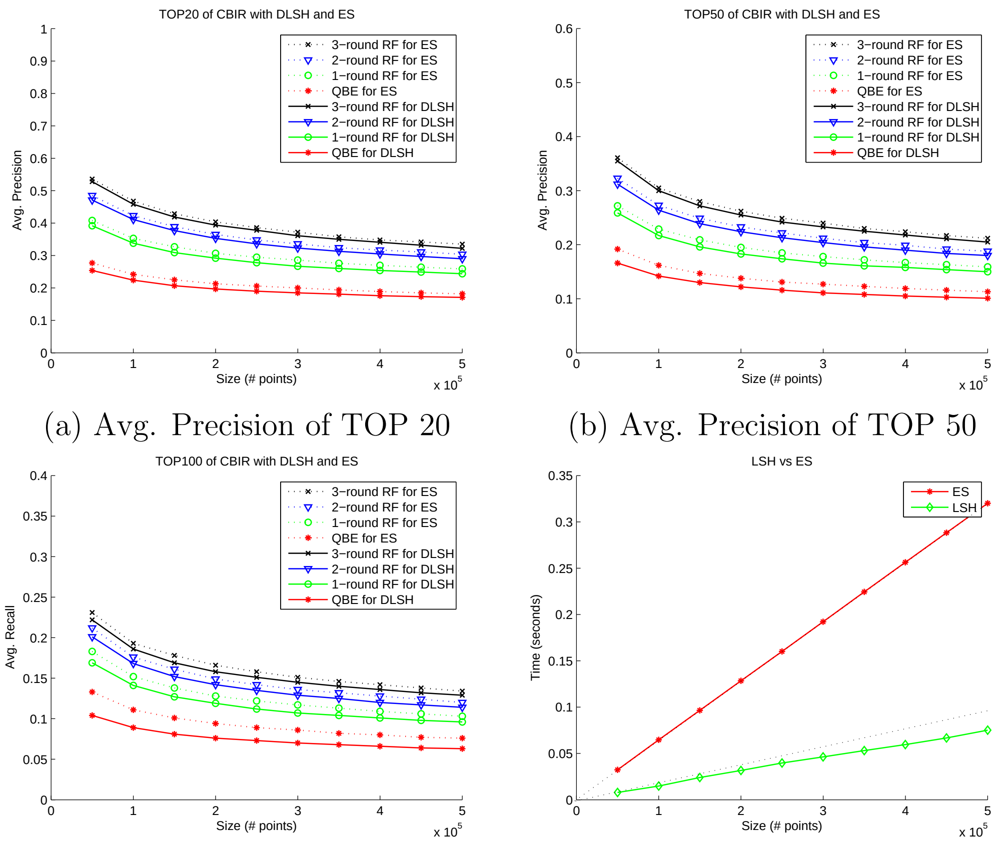
To choose a query set for performance evaluation, we randomly select 20 images from each of the 50 COREL image categories, to form a query set of 1000 image examples. To evaluate how our CBIR system performs with respect to different database scales, we prepare a number of image databases of different sizes up to $1,000,000$ images. For example, a database of size D contains $5,000$ COREL images and $D-5000$ other images selected from our testbed.

We employ the techniques described in section 3.4 to extract the image features (color, shape and texture) of the query and database images, in which each image is represented by a 238-dimensional feature vector.

We have implemented two systems for each of the two indexing approaches . Both of the systems employ LSH with parameters $L = 550$ and $k = 34$. The average number of nearest neighbors returned in every LSH query is controlled at around 6000 nearest neighbors by tuning the parameter R. For the system that implements the disk-based multi-partition indexing approach, a single 3.2GHz Intel Pentium 4 PC with 2GB memory running Linux kernel 2.6 is used for all of our experiments. All implementations are programmed in C++ and the maximum partition size is set to $100,000$. As for the system that implements the parallel-based multi-partition indexing approach, each Slave program is run on an individual 3.2GHz Intel Pentium 4 PC with 2GB memory running Linux kernel 2.6, and the Master program is hosted on a Sun Blade 2500 (2 x 1.6GHz US-IIIi) machine with 2GB memory running Solaris 8. During performance evaluation, a console-based Frontend program is run on a Nix dual Intel Xeon 2.2GHz with 1GB memory running Linux kernel 2.6. All implementations are programmed in C++ and the maximum partition size varies between experiments. During functionality demonstration, another Frontend program is used. This is an ASP.Net server program with C# code behind, running on a PC with Windows OS. All of these machines are connected together in a 100Mbps network speed local area network.

3.5.4 Experiment I: Disk-Based Multi-Partition Indexing Approach

In this experiment, we have prepared 10 databases with size ranging from $50,000$ to $500,000$ images. For each of these databases, we perform two sets of experiments to compare our LSH solution with the traditional methods. One set of the experiments employs our proposed LSH solution, while the other set uses an exhaustive linear search (ES) method based on Euclidean distance similarity measure. In each experiment, we query the database with



(a) Avg. Precision of TOP 20 (b) Avg. Precision of TOP 50

(c) Avg. Recall of TOP 100 (d) Avg. Query Time

Figure 3.2: Graphs showing the average recall, the average precision, and the query time of the disk-based approach over the 0.5 million image dataset.

a set of query images. For each query image, we retrieve the top 20, top 50 and top 100 ranking images. The returned ranking images are checked against the ground truth to determine the relevant images. Based on the number of relevant images, we measure the precision and recall rates. The recall and precision rates for all queries are averaged for final performance comparison. In each experiment, we simulate three rounds of relevance feedback. Relevance feedback is based on a k-Near Neighbors approach [17] by re-querying the database with the relevant examples with respect to the given query.

The CBIR system employing the disk-based multi-partition indexing approach is called the DLSH system, while that employing the exhaustive linear search approach is called the ES system. Fig. 5.4(a) and (b) shows the experimental results of the average precision obtained using LSH and ES respectively in our CBIR system. In these figures, the top 20 and the top

Table 3.1: Speed of LSH over ES on different databases

Size	ES (ms)	LSH (ms)	CPU Speedup
50000	32.3	7.6	4.246
100000	64.7	14.3	4.507
150000	96.9	22.8	4.244
200000	129.7	30.3	4.271
250000	160.7	37.3	4.306
300000	192.9	43.6	4.417
350000	224.8	49.8	4.508
400000	256.6	55.3	4.637
450000	288.4	62.4	4.617
500000	320.0	68.9	4.640

50 ranking images are returned for evaluation. The “Query-By-Example” (QBE) curve represents the average precision without any relevance feedback. The “1-round RF” curve represents the average precision when one round of relevance feedback is applied to refine the QBE result. The “2-round RF” curve shows the average precision when another round of relevance feedback is applied to refine the “1-round RF” result. From the figure, we can observe that the results from LSH are very close to the ES results. The maximum difference is no more than 5% for any database size. Fig.5.4(c) shows the average recall of our CBIR system using LSH and ES respectively, when the top 100 ranking images are returned for evaluation. From Fig.5.4(a), (b), and (c), we can see that the average recall and average precision of our CBIR system decrease with the database size, yet the rate of decrease diminishes as the database size increases. Fig.5.4(d) shows the average query time of our CBIR system with LSH versus that with ES. From the results in Fig.5.4(d), we can see that the LSH approach outperforms the ES approach. To examine the details, we also show the time performance comparison in Table 3.1. We can see that the LSH approach is much faster than the ES solution, with an average speedup greater than 4 times. Also, the gap of time performance between them increases as the database size increases. In the next section, we present the experimental results for our parallel solution, which further improves the efficiency of this approach.

3.5.5 Experiment II: Parallel-Based Multi-Partition Indexing Approach

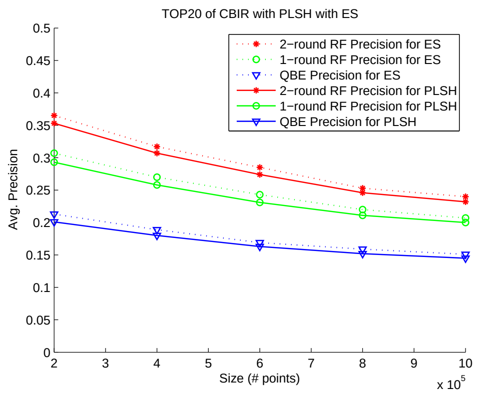
For the implementation of the parallel-based multi-partition indexing approach, the number of Slaves is a critical factor in system performance. If we evenly distribute the partitions of a database to all of the available Slaves, having more Slaves avaiable means each one handles a smaller partition. As seen in Fig. 5.4, LSH query time is sublinear relative to the database size. This implies that when the number of Slaves increases, each of the Slaves will perform a faster LSH query, and at last, the eventual system responsiveness will be improved. Of course, increasing the number of Slave means that more hardware resources are required.

In the following experiments, we evaluate the performance of the parallel systems using the maximum and minimum number of available machines. The system processing time of the parallel-based approach using the minimum possible number of machines is also compared with that of the disk-based approach.

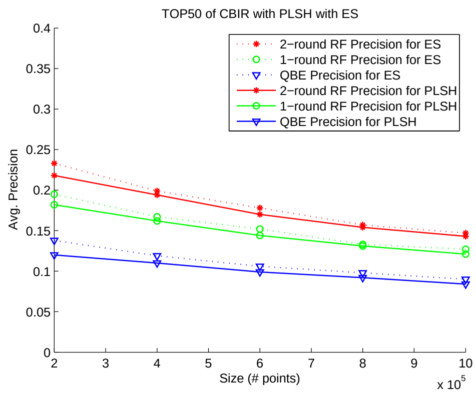
System Performance with Maximum Number of Machines

In this experiment, we employ four machines to serve as the Slaves of our parallel system, which we called the *PLSH* system. Irrepective of the database size, we employ all the possible Slave machines and evenly distribute the partitions to all of them. This experimental setting ensures the system performance is maximized by using the maximum possible hardware resources. In total, five tests are performed. Each test is done on a different-sized database. The sizes range from 0.2 million to 1.0 million images. Each of these databases is evenly divided into four partitions that are to be assigned to the four Slaves. In each test, the recall, the precision, and the processing time are recorded. As in the previous experiment, we compare the performance of our system with that of the *ES* system, which uses the exhaustive linear search method based on a Euclidean distance similarity measure.

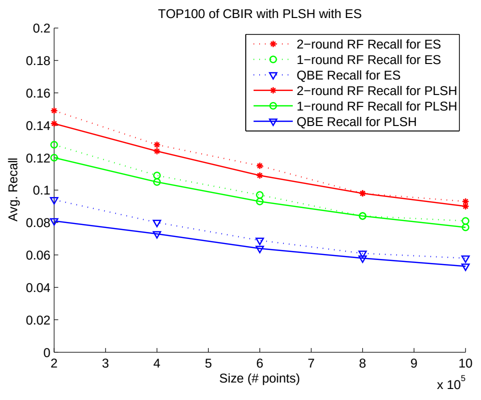
The experimental results are summarized in Fig. 3.3. As in the previous experiment, we conduct two rounds of relevance feedback in each test. The results from the parallel system are represented by the "PLSH" curve, while those from the system employing exhaustive linear search are represented by the "ES" curve. As seen from Fig. 3.3(a), (b), and (c), the decreasing rate of both the average recall and the average precision of the PLSH system diminishes as expected when the database size increases. This result agrees with the result of the previous experiments conducted on databases with size below 0.5 million. This indicates that the accuracy of our system is scalable



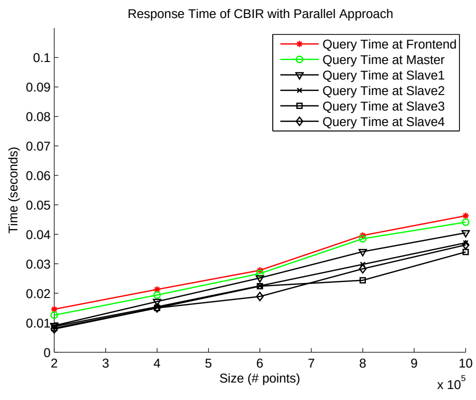
(a) Avg. Precision of TOP 20



(b) Avg. Precision of TOP 50



(c) Avg. Recall of TOP 100



(d) Avg. Query Time

Figure 3.3: Graphs showing the average recall, the average precision, and the query time of the parallel-based approach using the maximum number of machines on a 1 million image dataset.

for databases as large as 1.0 million images.

Fig. 3.3(d) shows the amount of time elapsed between the time a request is received and the time a reply is sent out in each machine. This empirical data is recorded in all machines, i.e. the Frontend, the Master, and the four Slaves.

As seen from Fig.3.3(d), there is a *constant* time difference between the query time recorded at the Frontend and the Master. As mentioned before, during experimental evaluation, the Frontend is just a simple TCP client simulating user requests and receiving answers for evaluation. It does not need to spend much time on loading and sending images. Therefore, the processing time of the Frontend is negligible. Thus, this time difference is mainly composed of the RTT between the Frontend and the Master. Similarly, there is a *constant* time difference between the query time recorded at the Master and the maximum times recorded at the Slaves. However, it can be easily observed that this time difference is slightly larger than that between the Frontend and the Master. This is because the processing time of the Master is relatively longer. Nevertheless, both of these time differences are relatively small compared with the LSH query times captured by the query times at the Slaves, as shown in Fig.3.3(d). This indicates our system is efficient, since the total time overhead of all processes in our system other than the LSH query process is small.

The variation of the total system processing time against the size of the database is captured by the "Query Time at Frontend" curve in Fig.3.3(d). In our experimental design, when the database grows larger, the size of the partition distributed to each Slave increases accordingly. This in turn increases the LSH query time of each Slave. As expected, as seen from the figure, the total time increases as the database grows larger. The query time is more or less linear relative to the size of the database.

System Performance with Minimum Number of Machines

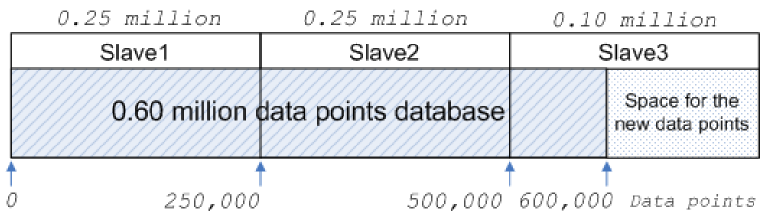


Figure 3.4: An example of the partition assignment on a 0.60 million data points database.

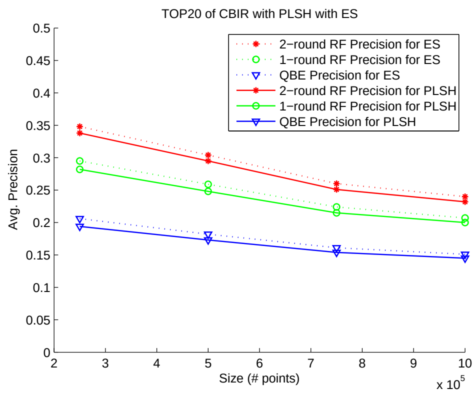
In contrast to the previous experiment, we use here the minimum possible number of Slave machines to provide the similarity search service. This is to minimize the demand for hardware resources and maximize the usage of the hardware resources that are employed. As in the previous experiment, we evaluate the system by measuring the recall, the precision, and the query time of the system.

All Slaves used in our experiments possess exactly the same hardware and software configuration. Thus, they have the same maximum partition size. Under our experimental settings, the maximum partition size is 0.25 million data points.

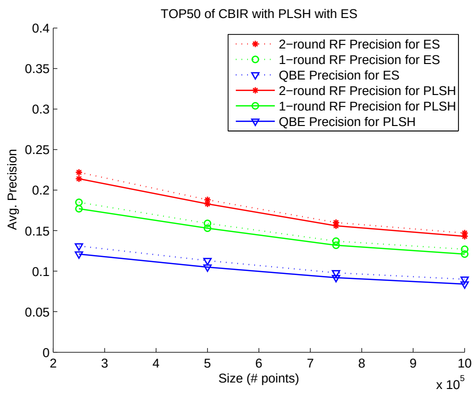
To minimize the number of Slaves while preventing the use of the virtual memory (disk swapping) in any of the Slaves, the database is divided into as few partitions as possible under the constraint that each partition contains at most 0.25 million data points. Fig. 3.4 shows an example of a way to divide the database. In this example, the first 0.25 million data points in the database is assigned to partition 1 which is to be indexed by Slave1. The second 0.25 million is indexed by Slave2, and so on. If there is any partially filled partition, e.g., the partition indexed by Slave3 in Fig.3.4, future data points will be assigned to this partition. Otherwise, new data points will be assigned to a new partition indexed by a new Slave machine. This simple partition distribution algorithm enables the whole system to be highly scalable and easily maintained. For instance, when adding new data points to the system, the system only need to modify one Slave without changing other Slave machines. In future work, we may study better partition strategies to improve other aspects of performance, such as workload balance.

Fig. 3.5(a), (b), (c), and (d) show the average precision, the average recall of the top 50 ranking images, the average recall of the top 100 ranking images, and the average query time of our system respectively. As shown in the figures, the average recall and precision of the PLSH system using the minimum possible number of machines are nearly the same as that using the maximum number of machines. This indicates that although we have used fewer machines, our system can achieve the same level of accuracy. As for the average query time, we can see that the query time tends to keep at a constant value. Since some Slaves may not be used when the database is not large, there are no query time record.

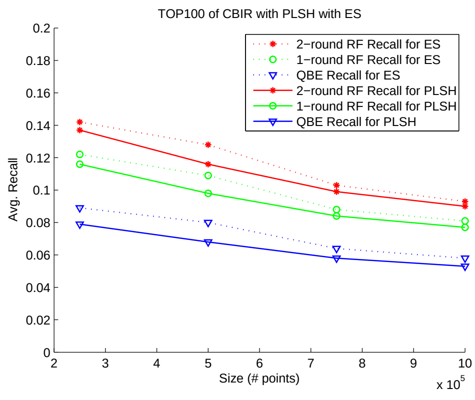
We discuss the processing time of this PLSH system in more details in the next section.



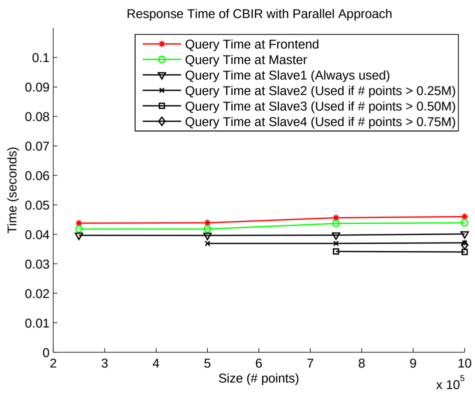
(a) Avg. Precision of TOP 20



(b) Avg. Precision of TOP 50



(c) Avg. Recall of TOP 100



(d) Avg. Query Time

Figure 3.5: Graphs showing the average recall, the average precision, and the query time of the parallel-based approach using the minimum number of machines over 1 million image dataset.

Table 3.2: The Comparison of Time Performance and Speedup.

SIZE ($\times 10^6$)	$\mathbf{T^{ES}}$ (ms)	$\mathbf{T^{ES}_{DAO}}$ ($\times 10^3$ ms)	$\mathbf{T^{PES}}$ (ms)	$\mathbf{T^{DLSH}}$ (ms)	$\mathbf{T^{DLSH}_{DAO}}$ ($\times 10^3$ ms)	$\mathbf{T^{PLSH}}$ (ms)	$\frac{\mathbf{T^{ES}}}{\mathbf{T^{DLSH}}}$	$\frac{\mathbf{T^{ES}_{TOTAL}}}{\mathbf{T^{DLSH}_{TOTAL}}}$	$\frac{\mathbf{T^{ES}_{TOTAL}}}{\mathbf{T^{PLSH}}}$	$\frac{\mathbf{T^{PES}}}{\mathbf{T^{PLSH}}}$
0.1	64.7	0	68.7	14.3	0	19.8	4.52	4.52	3.27	3.47
0.2	129.7	79.4	133.7	30.3	144.0	35.9	4.28	0.55	2216.43	3.72
0.3	192.9	158.9	164.7	43.6	290.0	43.8	4.42	0.55	3632.48	3.76
0.4	256.6	236.3	164.7	55.3	438.0	43.8	4.64	0.54	5400.43	3.76
0.5	320.0	320.0	164.7	68.9	597.0	43.9	4.64	0.54	7296.41	3.75
0.6	383.8	392.4	164.7	89.1	749.0	44.6	4.31	0.52	8806.12	3.69
0.7	448.7	473.1	164.7	99.4	895.3	45.4	4.51	0.53	10430.77	3.63
0.8	525.8	551.8	164.7	128.2	1045.9	45.7	4.10	0.53	12085.54	3.60
0.9	581.7	631.3	164.7	143.8	1191.6	45.8	4.05	0.53	13795.65	3.60
1.0	658.9	709.2	164.7	161.2	1343.9	46.0	4.09	0.53	15432.47	3.58

The Comparison of Time Performance and Speedup

By comparing Fig.3.5(a), (b), and (c) and Fig.5.4(a), (b), and (c), we can observe that the PLSH system has nearly the same accuracy as the DLSH system which employs the disk-based multi-partition indexing approach. As for the processing speed, they are summarized in Table-3.2. The meanings of the table headers are as follows:
SIZE: the number of data points in the database.
$\mathbf{T^{ES}}$: the query time (milliseconds per query) of the ES system, which uses an exhaustive linear search by Euclidean distance measure.
$\mathbf{T^{ES}_{DAO}}$: the disk access overhead (seconds per query) spent on loading feature vectors from disk into the main memory such that Exhaustive Linear Search can be performed. For example, if the system can keep at most $100,000$ data points in the main memory, when the database contains more than $100,000$ data points, the system has to erase the previously loaded data points and load another $100,000$ data points in order to perform a similarity search.
$\mathbf{T^{PES}}$: the query time (milliseconds per query) of the Parallel ES (PES) systems which is a parallel version of the ES system. The way we parallelize the ES system is exactly the same as the PLSH system that uses the minimum number of machines. Thus, this PES system also contains the three major components: Frontend, Master, and Slave. This implies some network transmission delays; the total delay can be obtained by subtracting the query

time in this column by that in the $\mathbf{T^{ES}}$ column. To enable fair comparison between the performance of the PES system and that of the PLSH system, the hardware resourcse used by both systems are controlled to be exactly the same. That is, the PES system and the PLSH system are set to use the same number of machines.
$\mathbf{T^{DLSH}}$: the query time (milliseconds per query) of the DLSH system required to perform a similarity search using LSH indexing.
$\mathbf{T^{DLSH}_{DAO}}$: the disk access overhead (seconds per query) spent on loading the partition structure from disk into the main memory such that LSH query can be performed. Since the system can keep the partition structure of at most (for example) $100,000$ data points at a time in the main memory, when the database contains more than $100,000$ data points, the system has to erase the previously loaded partition structure and load another partition structure of $100,000$ data points in order to perform a similarity search.
$\mathbf{T^{PLSH}}$: the query time (milliseconds per query) of the PLSH system required to perform a similarity search using LSH indexing.
$\mathbf{\frac{T^{ES}}{T^{DLSH}}}$: the speedup of the DLSH system over the ES system *without* taking the disk access overhead into account.
$\mathbf{\frac{T^{ES}_{TOTAL}}{T^{DLSH}_{TOTAL}}}$: the speedup of the DLSH system over the ES system taking the disk access overhead into account. $T^{ES}_{TOTAL} = T^{ES} + T^{ES}_{DAO}$ and $T^{DLSH}_{TOTAL} = T^{DLSH} + T^{DLSH}_{DAO}$.
$\mathbf{\frac{T^{ES}_{TOTAL}}{T^{PLSH}}}$: the speedup of the PLSH system over the ES system taking the disk access overhead into account.
$\mathbf{\frac{T^{PES}}{T^{PLSH}}}$: the speedup of the PLSH system over the PES system.

From the sum of the $\mathbf{T^{DLSH}}$ column and the $\mathbf{T^{DLSH}_{DAO}}$ column, it can be seen that the system processing time is too long for a real-time online application. Moreover, if we take the disk access overhead into account, from the $\mathbf{\frac{T^{ES}_{TOTAL}}{T^{DLSH}_{TOTAL}}}$ column, it can be seen that the DLSH system is no faster than the ES system. The reason is that, for the DLSH system, there are more data to be loaded into the main memory from disk before similarity search can be performed. To eliminate the disk access overhead, we introduce the parallel multi-partition indexing approach. As seen from the $\mathbf{\frac{T^{ES}_{TOTAL}}{T^{PLSH}}}$ column, the parallel approach is a considerably faster than the ES system, and the speedup increases with the size of the database. For instance, when the database grows as large as 1.0 million data points, the parallel approach is 15432 times faster than the ES system.

It may be argued that this is an unfair comparison, since the ES system would use only one machine to perform similarity search while the PLSH system would use multiple machines. To address this, we setup a parallelized

ES system for fair comparison, as described above. By parallelize the ES system, disk access overhead can be eliminated since each Slave maintains only a limited size of database and all the data points of a partition can be completely loaded into the main memory without putting the swap space to use. The query time of this PES system is shown in the $\mathbf{T^{PES}}$ column. From the $\frac{\mathbf{T^{PES}}}{\mathbf{T^{PLSH}}}$ column, it can be seen that the PLSH system still processes queries much faster than the PES system. This supports our choice of LSH indexing technique in the CBIR system for performing similarity searches.

3.6 Application to WWW Image Retrieval

To demonstrate the real-time performance of the PLSH system, we have built a web-based application that serves as the Frontend of our system. Since the Frontend is built using ASP.NET and hosted on a computer, users can access the Frontend through the Internet to use our similarity search service through a browser.

Figs.3.6 demonstrates the functionality of our web-based Frontend application, *PD-CBIR*. In each of the demonstrations, a real query example is presented. Fig.3.6 shows an example, querying topic *plate* over a 1 million image database. To start a search, the user can request to search for a specific semantic topic by selecting one of the images randomly shown in the *Search Page* (Fig.3.6(a)). After the request is submitted, the top k images most similar to the query image are then shown (Fig.3.6(b)). Then, the user can perform the following operations:

1. Select relevant images and irrelevant images with respect to the query topic by checking the tick and cross radiobox respectively.

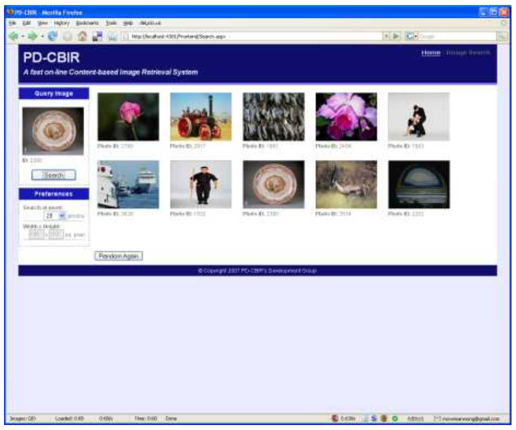
2. Refine the search result by submitting the relevance feedback information (Fig.3.6(c), (d)).

3. Change the query image by clicking the *search this* button under the target image.

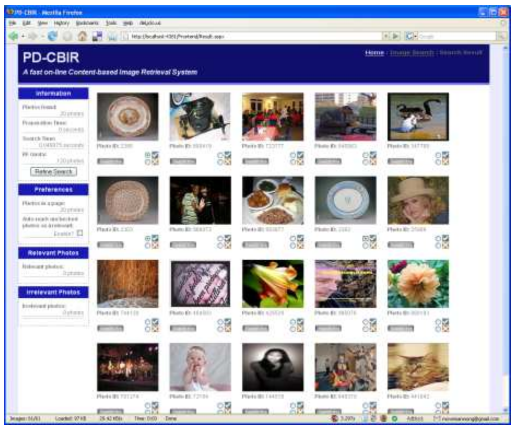
Similarly, Fig.3.7 shows an example for querying topic *surfing* over a 1 million image database and Fig.3.8 shows an example for querying topic *kung fu* over a 1 million image database.

3.7 Summary

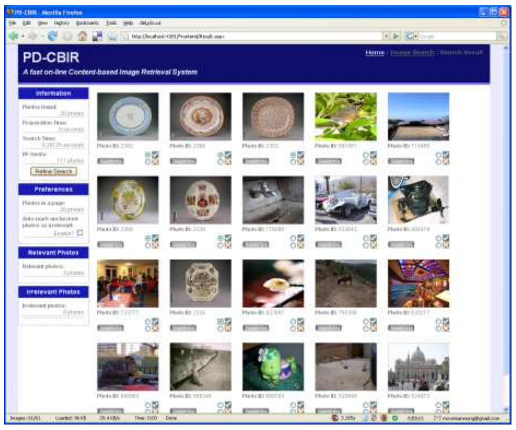
In this paper we proposed a novel parallel and distributed indexing scheme for building scalable content-based image retrieval systems. The suggested



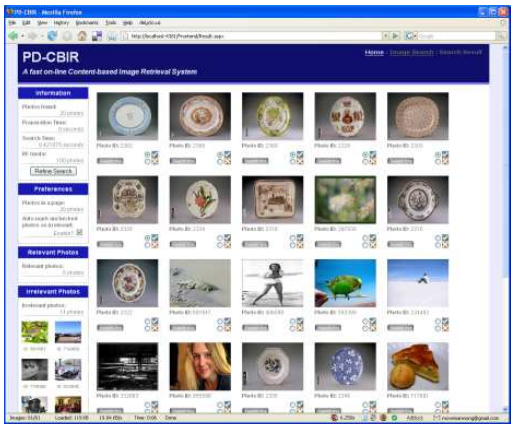
(a) Search page.



(b) Search Result.

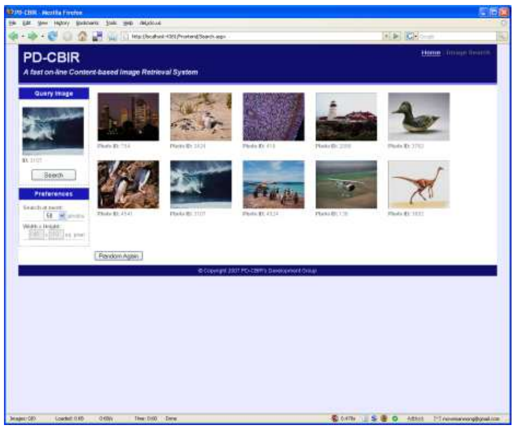


(c) 1-Round Relevance Feedback.

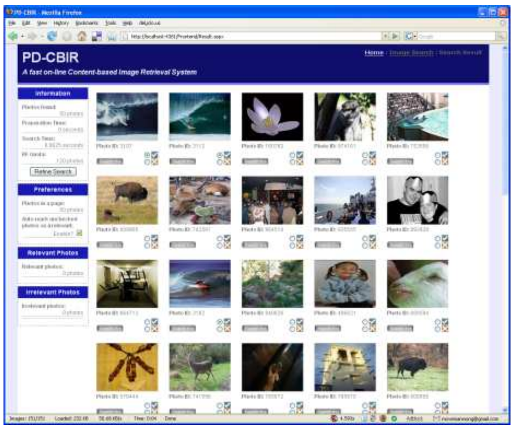


(d) 2-Round Relevance Feedback.

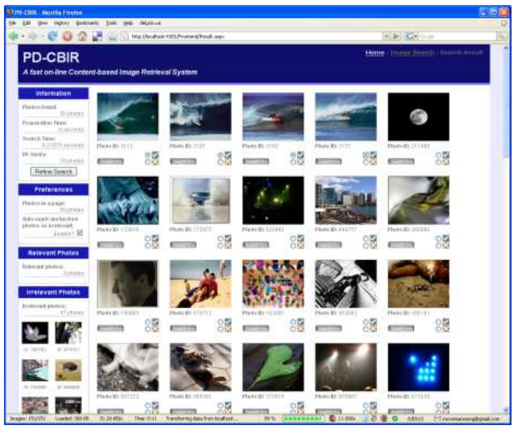
Figure 3.6: Some screenshots of *plate* search using our web-based Frontend application, *PD-CBIR*, over a 1 million image database.



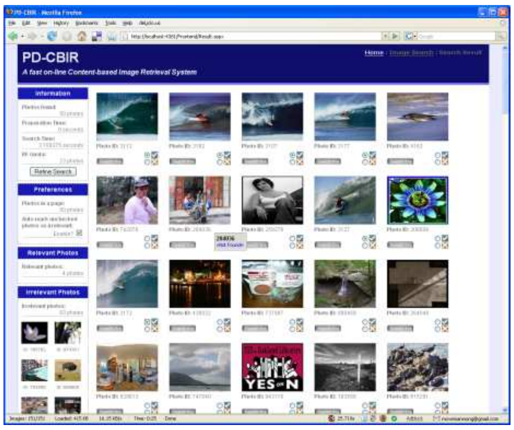
(a) Search page.



(b) Search Result.

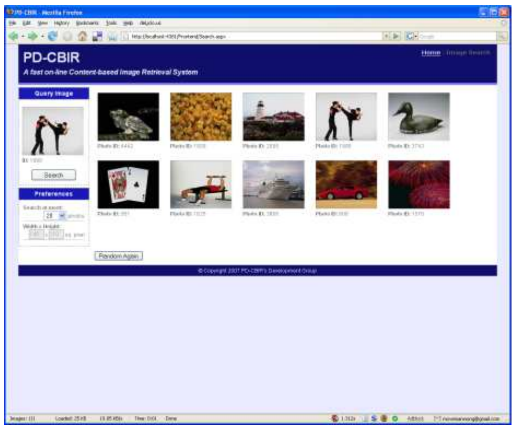


(c) 1-Round Relevance Feedback.

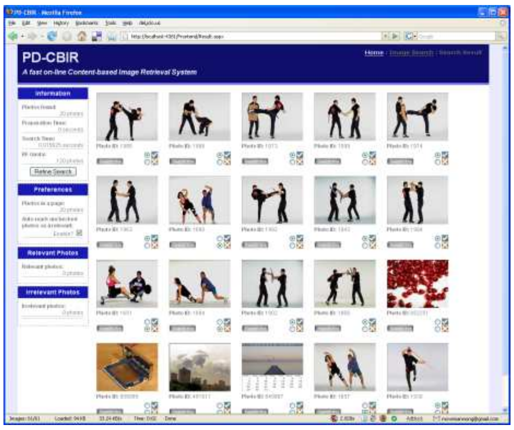


(d) 2-Round Relevance Feedback.

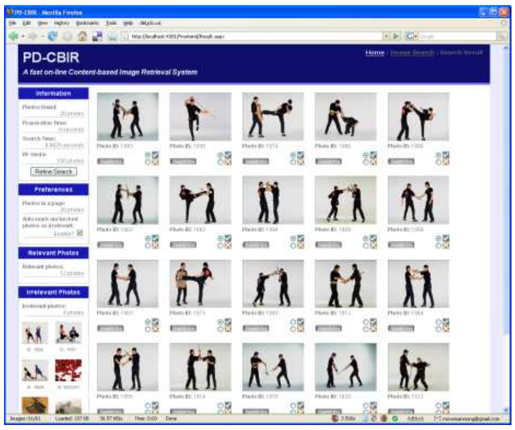
Figure 3.7: Some screenshots of *surfing* search using our web-based Frontend application, *PD-CBIR*, over a 1 million image database.



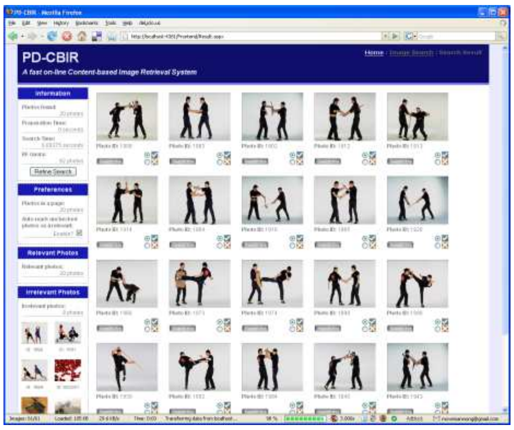
(a) Search page.



(b) Search Result.



(c) 1-Round Relevance Feedback.



(d) 2-Round Relevance Feedback.

Figure 3.8: Some screenshots of *kung fu* search using our web-based Frontend application, *PD-CBIR*, over a 1 million image database.

indexing scheme employs an efficient indexing technique, locality-sensitive hashing (LSH), which is efficient for high dimensional data indexing. We have implemented our proposed parallel and distributed indexing scheme and applied it to large-scale content-based image retrieval of WWW images. We conducted an extensive set of evaluations, including the evaluation of time performance and retrieval performance of query and relevance feedback, over a large-scale testbed of 1 million images. The promising results showed that our proposed parallel and distributed solution delivers highly efficient performance, which is significantly more efficient than traditional exhaustive linear searching methods. We believe that this empirically-proven scheme could be valuable for future development of real-world large-scale CBIR applications.

□ **End of chapter.**

Chapter 4

Image Retrieval System for IND Detection

4.1 Overview

4.1.1 Motivation

Users of content-based image retrieval systems are usually interested in looking into a limited number of search results, say the top 50 or the top 100 of the search results, in each search query. Thus, it would be highly undesirable to them if there exist some duplicated images in the results, because it means the information gained by them will further be decreased. Therefore, removing duplicated images prior to the return of results is critical in improving the quality of CBIR search. Duplication removal can be performed during web image crawling and after crawling.

During web image crawling, it is not difficult to encounter some web pages that contain multiple sizes of the same photo. In these web pages, the thumbnail photos are usually displayed to the users first and links are left for the users to browse or download the same photo with higher resolution. This photo sharing scheme is more and more popular and there are increasing number of web sites adopting this scheme to fasten the loading speed in the client sides. The multiple sizes of the same photo are certainly different pieces of photos. However, with respect to their contents, they are the same. Because of this, these photos are very likely to be returned together to the users as different search results in CBIR. As discussed before, this is not desirable to the users and should be avoided. However, normal web image crawlers can not differentiate multiple sizes of the same photo but just regard these photos as different individuals. This is because they do not analyze the contents of the images they have crawled. In this chapter, we will introduce an image near-duplicate (IND) detection system that can identify near-duplicate images in a database based on the analysis of the contents of the images. Equipped with this system, web image crawler can decide to retain only a subset of the near-duplicate images in a web site which in turn decreases the memory storage and improving the quality of the CBIR search. Since the number of images in a web site are usually small, with a fast searching technique, our IND detection system can find near-duplicate images in fast speed.

Sometimes near-duplicate images can not only be found in the same web site, but also in different and unrelated web sites. Thus, IND detection can also be performed over all crawled images after the web image crawling process is completed. It can be imagined that the crawled image database would be huge and the IND detection would take a long time to search for all near-duplicate images. Nevertheless, since web image crawling is an off-line application, the speed of the IND detection is not critical.

Image near-duplicate detection has been an important application in recent years. Because of the widespread of the Internet, more and more people publish images on the Web for many purposes. People may publish their own images or publish the images bought from photo agencies. However, some people illegally copy the images from the others without acknowledging the owners of the images. This undeniably affects the businesses of photo publishing agencies. Since people always manipulate the pirate images before publishing, the pirate image and the near-duplicate images cannot simply be detected by digital watermarking method. On the contrary, image matching using invariant local feature approaches can easily tackle this problem.

Beside private image detection and CBIR search refinement, another possible use of IND detection system is in finding similar web pages on the Internet. Since similar web pages usually have similar images, one can use our IND detection system to detect similar images from two web pages and use it as a cue to evaluate the similarity between the two web pages.

4.1.2 Related Work

Different definitions in near-duplicate image exist in the academia. Yan Ke [23] and Berrani et al. [5] define near-duplicates as images altered with common manual image manipulation process such as changing contrast, saturation, resizing, cropping, framing, jpeg-compression, etc. Images of a scene taken under different environments or camera conditions are not regarded as near-duplicate image at all. On the other hand, Zhang [57] defines near-duplicates as images that are close to exact duplicate but with variations due to content changes, camera parameter changes, and digitization conditions. Both definitions are reasonable but they are suitable for different applications. For CBIR search refinement, the former definition is more suitable. When we prepare a dataset for performance evaluation, this definition is taken into consideration. Most of the previous works adopt traditional CBIR approaches in building the IND detection system. Since these approaches generally adopt global features of image as the key to search for near-duplicates, their performances can suffer when significant cropping, resizing or framing is applied on the near-duplicates. Moreover, systems using global features usually lack a self-verification process, likes the geometric verification in [23]. Thus, they tend to have many false positives. Instead of using global features, some IND detection systems build parts-based representation of images using invariant local features were proposed. Berrani [5] proposed a IND detection system employing local differential descriptors and approximate similarity search. Yan Ke [23] proposed to use PCA-SIFT invariant local feature descriptors and Locality-Sensitive Hashing (LSH). The matched features are filtered using geometric verification such that the precision rate is significantly improved. According to the recent performance evaluation done by Mikolajczyk et al. [22], SIFT-based descriptors [32] outperforms the differential descriptor in image matching problem. There also exists some other SIFT-based descriptors, such as SIFT and GLOH, that are better than PCA-SIFT in term of both the recall and precision rate. Therefore, their IND detection systems can be further improved by using more powerful feature descriptors. Locality-sensitive hashing has been proved [23] to be effective in finding near neighbors both in accuracy and speed. However, the LSH algorithm employed by Yan Ke assumes L1 (Manhattan) distance in the analysis of near neighbors which is not as effective as L2 distance, as shown in [23].

4.1.3 Objective

We aim our IND detection system at copyright protection and CBIR search refinement purpose. Therefore, the former definition of near-duplicates is more suitable. In other words, images with variations due to camera parameter changes and content changes are not counted as near-duplicates. Our system is a variant of content-based image retrieval system in which the desired images to be retrieved are originated from a source copy which is the same as that of the query image. Although the desired images and the query image share the same source, they differ with some common image transformations due to the manual image manipulation process.

To make our system more practical, we chase for high recall, high precision, and high speed. To achieve this, we employ SIFT feature descriptor which perform the best in our performance evaluation presented in Section

2.4. We would also try the SURF feature descriptor since it is shown to have comparable performance with SIFT but has smaller number of dimensions and thus it is fast to compute and search. To improve the speed of searching, we employ locality-sensitive hashing (LSH) [6] which is a special kind of hashing algorithm first proposed by Indyk & Motwani. It is designed for solving the approximate/exact near neighbor search in high dimensional Euclidean space. The implementation of this algorithm is released to public in [1].

4.1.4 Contribution

The first contribution of this chapter to IND detection is the integration of the state-of-the-art SIFT feature descriptor with fast LSH retrieval method that makes our IND detection system accurate and practical. The second contribution of this chapter is the introduction of a new verification process, called orientation verification, on the candidate matches such that the recall and precision of the system can be further improved. The third contribution of this chapter is the introduction of a new distance metric, called k-NNRatio, that integrates k Nearest Neighbor algorithm with distance ratio to further improve the average recall and precision rate. Although our system is targeted at IND detection, it can easily be modified to suit for other CBIR applications.

4.2 Database Construction

Our system consists of two main phases: the database construction phase and database query phase. In the database construction phase, we process every image in the image collection and extract a set of invariant local features from it. We then build an index for all the extracted features using LSH algorithm. In the database query phase, user can issue a query to find near-duplicates of the submitted query image. The system extracts a set of invariant local features from the query image and issues search queries to the pre-built LSH hash table based on the extracted features. Search results of all query features are then verified and the top k most probably near-duplicates are returned to the user. Detail description of the database construction phase will be given in this section and the database query phase in the next section, Section 4.3.

4.2.1 Image Representations

Scale-invariant feature transform (SIFT) feature descriptors are very suitable for tackling IND detection problem. The image manipulations commonly applied on near-duplicate images include *changing illumination, contrast, coloring, saturation, resizing, cropping, framing, affine warping, and jpeg-compression.* SIFT descriptor is invariant to all of these transformations because of the following reasons. Firstly, the descriptors are normalized such that it is invariant to illumination change and contrast. Secondly, the descriptors are built using the grayvalues of color images and thus it is invariant to coloring and saturation. Thirdly, the descriptors are computed on many different scales of each image such that there are always some features common to two images with different scales only. Thus the descriptors are invariant to resizing. Fourthly, the descriptors are local in nature which means some local changes to the image, like cropping and framing, have little adverse effect to the IND system, as long as the number of features sampled in each image is large enough. Fifthly, the descriptors are orientation histograms over 4×4 sample feature regions. They are less sensitive to significant shift in gradient positions and thus they are invariant to affine warping. Finally, Gaussian filters of different widths are applied on the images before SIFT descriptors operate on them. Thus the effect of the changes in the content of compressed images due to jpeg-compression are reduced. Because of the above reasons, we adopt the powerful SIFT descriptor as the image representation of our system.

Despite of these advantages, there is one strong reason why invariant local descriptor is not desirable to practical system. The reason is the size of local feature database is huge. Since each image can generate hundreds of SIFT features and each feature is a 128 bytes long vector, the size of a feature database of just thousands of images is already huge. Searching for desired features in such database using simple exhaustive linear search algorithm is not practical since it takes too long to run. However, a recently proposed feature indexing method, LSH, solve this problem. By building an index of the feature database, the searching process can be performed extremely fast with acceptable accuracy. The detail of index building is to be discussed next.

4.2.2 Index Construction

Among many previously proposed indexing algorithms, hashing is the fastest algorithm to lookup a database. The time complexity of database lookup of a hashing algorithm is $O(1)$ on average, which means a database lookup usually takes constant amount of time independent of the size of the database. Because of this attractive feature, hashing algorithm is commonly employed in indexing large-scale database. However, simple hashing algorithms do not satisfy our requirements. Different from the normal use of hashing, our query key is a high-dimensional vector of real numbers that is usually not an identical copy of any entry in the hash table. Moreover, the desired keys are not limited to an identical copy of itself but all the entries that are close to the query vector in Euclidean space l_2. The hashing algorithms that satisfy our requirements are a recently proposed technique called locality-sensitive hashing (LSH). It is first proposed by Indyk & Motwani [19] and further improved in [6]. Locality-sensitive hashing scheme can answer queries in sublinear time with each near neighbor being reported with a fixed probability. What is special with this type of hashing scheme is that it is locality-sensitive. By locality-sensitive, it means the probability that two points share the same hash value decrease with the distance between them. As for the hashing scheme we employed, the distance is Euclidean distance. Since points close to each other in Euclidean space share the same hash value, we can find the near neighbor of the query point by checking the collided hash buckets.

Advantages

Locality-sensitive hashing compromises speed with accuracy. It only ensures reporting every near neighbor of the query point with a certain probability. However, it is sufficient to our system. It is because an image usually contains at least hundreds of local features. Even if a large proportion of the near neighbors are missing, there are always enough near neighbors reported and contributed to voting the correct image hypothesis.

We have employed the E^2LSH(Exact Euclidean LSH) package [1] to build the LSH index of our feature database. This makes our LSH index conceptually different from that employed by Yan Ke [23]. The LSH algorithm employed by Yan Ke reports c-approximate R-near neighbors problem only while that employed by us reports exact R-near neighbors problem. The definitions of these two types of near neighbor problems are described below:

The c-approximate R-near neighbor problem formulates that if there exist a point p in the set of points P in the database that is at distance at most R from the query point q (i.e., satisfying $||p - q|| \leq R$), any point within the distance of at most cR from the query point (i.e., satisfying $||p - q|| \leq cR$) has to be reported.

The exact R-near neighbor problem formulates that each point p satisfying $||p - q|| \leq R$ has to be reported with a certain probability.

By using the exact R-near neighbor solution, we can ensure that the k nearest neighbors of any query point are found at a certain probability. Of course, if we demand a higher probability, the speed of the LSH algorithm will be slower. As for our system, we can obtain high recall and precision even when a low probability is set.

Scalability Problem

The major problem of E^2LSH is scalability. Although E^2LSH can answer the query in fast speed, it stores all the data points and the R-near neighbor data structures (which is built by the E^2LSH for indexing the database) in the main memory. Thus, the size of the database it can index at a time is limited by the amount of free main memory space in the computer. If the IND detection is to be applied on a database once only in a batch mode fashion, we can employ the following approach to solve the scalability problem:

1. Divide the dataset into several parts.

2. For each of the parts, run E^2LSH to build the index and run all queries on that part.

3. Collect the results from all the parts together to create the total results for the queries.

The hash index of each part will be created once altogether and thus there is no significant difference in speed between creating the index for one huge database and creating the divided databases one by one. However, to achieve this, one would have to obtain **all** the queries beforehand. This is not suitable for online applications in which users can submit query at any time. Nevertheless, this method is suitable for performance evaluation. To solve the scalability problem, we can adopt the parallel and distributed solution proposed in chapter 3.

Solving k-NN

The k-nearest neighbor problem can be solved using E^2LSH. However, each R-near neighbor data structure is suitable indexing with a specfic value of radius R. Thus, to find the k-nearest neighbor, an effective method is to create several R-near neighbor data structures with $R = \{R_1, R_2, ..., R_t\}$, where R_t is the threshold distance from the query point to its near neighbor. The query can be started with the near neighbor data structure with the smallest radius, say R_1, and continue with the data structure with larger radius until k-nearest neighbors are found. However, this method costs too much memory space to store the data structures and is thus not desirable. Instead, we create one near neighbor data structure with moderate radius only. k-nearest neighbors of the query are identified based on the distance between the query point and the query result points obtained from data structure.

4.2.3 Keypoint and Image Lookup Tables

For the sake of reducing the main memory usage, we do not store the details of the keypoints (SIFT feature vectors) and their corresponding images in the hash table. We store the details of keypoints and images in separate files on disk. In the keypoint lookup table, the details of each keypoint are stored in one line in plain text according to the following format:

Image ID	x-coordinate	y-coordinate	Scale	Orientation	45 chars

The keypoints are stored such that the indexes of keypoints in the LSH hash table are equal to the line numbers of the corresponding keypoints. Thus, as we retrieve a keypoint represented by a keypoint index from the LSH hash table, we can obtain the detail of that keypoint by looking up the line in the keypoint description file with line number equals the keypoint index. Since each line is of equal width of 45 characters, the memory address of the starting byte of a specific line can easily be calculated by this formula: $45 * index$.

In the keypoint lookup table, the image associated with each keypoint is represented by “Image ID”. The ID is actually an index to the image lookup table. The details of each image are stored in one line in plain text according to the following format:

Image ID	# keypoints	Length of File Name	File Name	86 chars

4.3 Database Query

After the index is built, query can be issued to search for near-duplicate images. During query search, the system extracts SIFT features from the query image and submit the features to E^2LSH one by one. E^2LSH then returns keypoints that are sufficiently close to the query keypoint in Euclidean space. However, not all returned keypoints are regarded as the matches. They are determined by the following matching strategies.

4.3.1 Matching Strategies

Since a query image may have several near-duplicate images in the database, a query keypoint of the query image can have several possible correct matches. Therefore, we should take a group of keypoints instead of taking a single keypoint as the candidate matches of a query keypoint. There are two commonly used matching strategies that are suitable for our requirements. They are threshold based matching and k-NN matching. We have also proposed a new matching strategies, called k-NNRatio matching, which is a integration of the k-NN matching the and distance ratio matching. Since E^2LSH supports retrieving keypoint within a threshold radius to the query keypoint, the keypoints returned from E^2LSH are already the candidate matches under the threshold matching strategy. To identify the k nearest neighbor of each query keypoint, we can employ the method discussed in the previous section, Section 4.2.2. The candidate matches under the k-NNRatio matching strategy are actually the same as those under the k-NN matching strategy, but they are additionally assigned a weight specifying its importance in deciding the matching images of the query image. The higher the weight, the more important is the candidate match. A weight closes to zero mean that the keypoint is hardly regarded as a candidate match and we may just remove it from the list of candidate matches.

Threshold Based Matching

Under the threshold based matching strategy, a query keypoint matches with a keypoint in the database if the distance between them is below a threshold R. We have adopted Euclidean distance (L_2) metric for SIFT feature descriptors in our experiment.

It is actually very hard to design a fixed threshold for this strategy. It is because the image transformations applied on different images are variant. However, under different transformations, the average distances between the query points and their matches are usually different. Therefore, a threshold suitable for certain transformation may not be suitable for the others. If we set the threshold too tight, too few matches will be obtained. We may not be able to determine the matching images based on a small amount of candidate matches and thus the recall rate will be low. On the contrary, if we set the threshold too loose, too many matches will be obtained. The matching images will be seem like being chosen by random and thus the precision rate will be low. To overcome these problems, we can set the threshold to a large value and rely on the orientation verification and RANSAC affine transformation verification processes to filter out, hopefully, all of the false matches. However, certainly, this will significantly reduce the speed of the system and not all false matches can be removed by these two processes if the original amount of false matches are numerous.

k-NN Matching

Under the k-NN matching strategy, a query keypoint Q matches with a keypoint K_A in the database if K_A is among one of the k nearest neighbors of Q and if the distance between them is below a threshold. With this approach a query keypoint has up to k matches. The threshold should be set large enough such that keypoints under serious image transformations are still within the threshold radii from the query keypoints such that it is highly probably that no correct matches are missed before choosing the nearest k.

The value k is, again, a fixed value. Since the content of database is variant, we actually cannot tell how many matches exist for each query keypoint. Certainly, we can tell how many matches exist in performance evaluation, but we cannot do so when our system is deployed to public use. As k is set much lower than the actual number of correct matches, not all matches of a query points may be included and this makes the recall rate low. On the contrary, as k is set far higher than the actual number of correct matches, many false matches are included and this makes the precision rate low.

k-NNRatio Matching

Under the above strategies, all keypoints within a threshold radius of a query points or among the k nearest neighbors are counted as the matches regardless of the inter-distances between those keypoints. To the extreme, for instance, there may exist a case in which ten keypoints are located within the threshold radius of a query keypoint. Among the ten keypoints, one of them is very close to the query keypoint while all the others are very far away from the query keypoint. Then, even if all the ten keypoints have some chances to be the correct matches, the nearest one has higher chance to be a correct match than the others.

To solve the above problem and to soften the adverse effect of a fixed value of threshold and k in the above matching strategies, we introduce a new matching strategy called k-NNRatio matching. Under the k-NNRatio matching strategy, a query keypoint Q matches with a keypoint K_A in the database if the distance ratio between K_A and the next nearest neighbor K_B is high enough, and if K_A is among the nearest k and the distance between K_A and Q is below a threshold. Under this definition, we can say the k-NNRatio matching strategy is an extension of the above strategy. We further integrate it with the nearest neighbor distance ratio employed in SIFT [32]. The requirement that the distance ratio between K_A and K_B

has to be high enough seems ambiguous. We say so because there is no hard threshold on the distance ratio that the keypoint has to reach to be a candidate match. Instead, a weight is assigned to each keypoint. A keypoint with high weighting means that the keypoint is more important in the image voting process because we have larger confidence that this keypoint is a correct match. The weight assignment follow the following two principles:

1. If a keypoint is nearer to the query point, it is more likely a correct match and its weight should be higher. Otherwise, its weight should be lower.

2. If a keypoint has large distance ratio to its next nearest neighbor, no other match seems like the correct match and its weight should be higher. Otherwise, the weight should be lower.

Our motivation to incorporate distance ratio into the weight for determining candidate matches is that, distance ratio matching strategy has been shown [22] to perform better than threshold based and nearest neighbor matching strategy in term of recall and precision. Now we employ it to determine multiple matches by incorporating it in the weight assignment process.

To satisfy the above requirements, the weight of a keypoint is formulated as follow:

$$Weight(K_A) = (\frac{a}{k(K_A)})^b \times (\frac{dist(K_B, Q)}{dist(K_A, Q)})^c \tag{4.1}$$

where a, b, and c are the real numbers to be empirically determined. dist(K,Q) is the Euclidean distance between keypoint K and keypoint Q. $k(K_A)$ is the rank number of K_A among the k nearest neighbors. That is, if keypoint K_A is the nearest neighbor of Q, then $k(K_A)$ is 1. If keypoint K_B is the second nearest neighbor, then $k(K_B)$ is 2, and so on. The weight of a keypoint depends on the rank number of the keypoint among the k nearest neighbors and also the distance ratio. The term $(\frac{a}{k(K_A)})^b$ is designed to satisfy the first requirement while the term $(\frac{dist(K_B,Q)}{dist(K_A,Q)})^c$ is designed to satisfy the second requirement. To balance the influence of these two terms, we introduce the parameters a, b, and c. We will discuss the choice of these values in the performance evaluation section.

To implement this matching strategy, we first query E^2LSH and obtain the k nearest neighbors of the query point. For each of the nearest neighbors, we calculate the weight by taking the first nearest neighbor and the second one into calculation. During the calculation of the weight of the second one, we take the second nearest neighbor and the third one into calculation.

There are several immediate advantages using this matching strategy. Firstly, the nearest neighbor does not always gain high weight. It will not have high weight if it is far from the query point. Secondly, a keypoint with higher rank number can still gain high weight if it is far away from all the other neighbors with higher rank number. Thirdly, the k nearest neighbors do not get the same weight and thus they have different voting power during image voting. If there are only two possible matches for a query keypoint, ideally only two keypoints will get high weights and all the others will get lower weights. This softens the adverse effect of the fixed value of K and the threshold.

4.3.2 Verification Processes

After the candidate matches for each query keypoint were selected, they are first sorted by their image IDs. Recall that each keypoint owns a keypoint index. Using this keypoint index, we lookup the keypoint lookup table for the line containing the detail of the keypoint. From that line, we can obtain an image ID that uniquely identifying the image from which the keypoint is extracted. By sorting the candidate matches by their image IDs, the keypoints extracted from the same images are brought together. We then group the keypoints together according to their image IDs. The keypoints in each group are then filtered based on their geometric relationship so as to reduce the number of probable false matches. We filter each group through two verification processes: orientation verification and affine geometric verification using RANSAC.

These two processes can only filter certain percentage of false matches. Therefore, we should not flood the inputs of these two processes with a large number of candidate matches of each image and totally rely on these two processes to filter out the large number of false matches. In other words, we should not set the threshold too large when using threshold based matching or set the K too large when using other matching strategies. This is to say, the matching strategy just discussed is an important part of the system and can not be skipped.

These two processes are applicable for IND detection but not applicable for applications like CBIR. This is because in CBIR, the scenes or objects inside the query image and the desired image are not necessary the same. They may just under to the same semantic topic. The geometrical transformation from the desired image to the query image may neither be affine transformation nor perspective transformation. For those applications in which the geometrical transformation among the matching keypoint pairs can be modeled by affine transformation, the two proposed verification processes will work perfectly.

The affine geometric verification process was adopted by Yan Ke [23] in building his IND detection system. However, we observed that the orientation of keypoints are not verified in his system. Thus, we propose a verification process that can work together with the affine geometric verification process to further remove probable false matches.

Orientation Verification

The orientation of a keypoint refers to the canonical orientation of the keypoint. It is determined by the image gradients of pixels in both x and y directions within the feature region. For detail, please see Section 2.4.1. Under most of the image manipulation processes, the orientation of a keypoint will change in similar amplitude as that of any other keypoints in the same image. In other words, all keypoints of an image rotates in similar amplitude as the image is manipulated using the previously discussed manipulation process. Therefore, the difference of in orientation between any of the query keypoint and its match should be more or less the same.

The orientation verification process of our system makes use of the consistency of this difference to remove the probable false matches and retains only the largest set of candidate matches that have consistent differences for each group.

Here are the steps of the orientation verification process for each group of candidate keypoint matches:

1. *Input* a group of keypoint matches.

2. For each candidate keypoint match, obtain the orientations of the query keypoint and the database keypoint through the keypoint lookup table using their indexes in the hash table as the indexes to the lookup table.

3. For each candidate keypoint match, subtract the orientation of the query keypoint from that of the database keypoint to obtain the difference in orientation. The range of the difference is $[-360°, 360°]$.

4. Fit the difference of each match into the range of $[0°, 360°]$ by adding $360°$ to it if it is negative.

5. Divide the range of difference into 36 bins, each with $10°$ width. Map the difference of each keypoint to one of the 36 bins and add the keypoint match pair into that bin.

6. Slide a moving window of the width 3 bins over the 36 bins. Slide for 1 bin each time for 36 times and wrap the window around at the end.

7. Find the maximum window which is the moving window having the maximum number of match pairs inside its 3 bins.

8. *Replace* the list of candidate keypoint matches of the current group with the list of matches existing in the 3 bins of the maximum window.

With this verification process, the number of false matches are significantly reduced. This can be reflected by the recall and precision rate of the system.

Affine Geometric Verification using RANSAC

The affine transformation between two images can be modeled by the following equation:

$$\mathbf{Ax} = \mathbf{b}$$

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} u_0 \\ v_0 \\ 1 \end{bmatrix}$$

where $\mathbf{x}$ are the homogenous coordinates of a keypoint in the query image, $\mathbf{b}$ are the homogenous coordinates of the matched keypoint in the database image, and $\mathbf{A}$ is the transformation matrix with six unknowns. To compute the transformation matrix, we need 3 keypoint match pairs. With the 3 keypoint match pairs $(\mathbf{x}_0, \mathbf{b}_0)$, $(\mathbf{x}_1, \mathbf{b}_1)$ and $(\mathbf{x}_2, \mathbf{b}_2)$, we can compute the matrix $\mathbf{A}$ by solving the following linear equation:

$$\begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix} \begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

Since the large matrix on the left is a square matrix, we can find the least square solution of the above linear equation by multiplying the inverse of that matrix with the vector on the other side:

$$\begin{bmatrix} a_{00} \\ a_{01} \\ a_{02} \\ a_{10} \\ a_{11} \\ a_{12} \end{bmatrix} = \begin{bmatrix} x_0 & y_0 & 1 & 0 & 0 & 0 \\ x_1 & y_1 & 1 & 0 & 0 & 0 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_0 & y_0 & 1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \end{bmatrix}^{-1} \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ v_0 \\ v_1 \\ v_2 \end{bmatrix}$$

In our implementation, we have employed the LU decomposition function and backward substitution function in the Numerical Recipes in C++ package [46] to compute inverse of matrix.

With the matrix **A**, we can affine warp every query keypoint with homogenous coordinates **x** from the query image to the database image by multiplying the the 3×3 matrix **A** with vector **x**.

We adopt RANdom SAmple Consensus (RANSAC) [10] to eliminate probable false matches in the group of candidate matches. Here are the steps of affine geometric verification for each group of candidate keypoint matches:

1. Check if there are at least 3 pairs of keypoint matches, remove the whole group from the list of candidate matches and finish the process if it is false.

2. Randomly pick three keypoint match pairs.

3. Calculate the affine transformation matrix based on these three match pairs only.

4. For all the other keypoint matches, map the query keypoint onto the database image and calculate the Euclidean distance between the mapped coordinates and the coordinates of the database keypoint. Compute the support of the current transformation by counting the number of matches with the distance smaller than a preset threshold, which is set to 10 in our performance evaluation.

5. Loop the above steps for a number of times, which is set to ten times in our performance evaluation. Find the transformation that receives the greatest support.

6. Retain only the candidate keypoint matches that support the greatest support transformation.

This verification process further improves the recall and precision rate of our system.

4.3.3 Image Voting

After the verification processes, a large percentage of false matches should have been removed. However, there usually still remain a number of groups of candidate matches. Each group represents a different database image that may be near-duplicate of the query image. To determine which is more likely to be the the correct match, we compare the support of that group which is defined as the number of verified candidate matches inside that group. The larger the support, the greater the probability that the corresponding image is a near-duplicate of the query image. If the system employs threshold based or k-NN matching strategy, we sort the groups by their supports in descending order and remove those that have supports fewer than the minimum support which is 5. The top N(=10) groups are returned to the user and counted as a match during performance evaluation. Under the k-NNRatio matching strategy, not only the "quantity" of a group but also the "quality" is used to rank the groups. We first calculate the weight of a group which is defined as the summation of all the weights of the keypoint matches in that group. We then sort the groups by their weights in descending order instead of simply by their supports. This makes the more probable keypoint matches contribute more to the image voting process than those less probable matches. Similarly, those groups that have weights smaller than the 5 are discarded and the top 10 groups are returned to the user.

4.4 Performance Evaluation

We have done a number of experiments to show that our proposed approaches do improve the performance of the whole system and that our system is effective. We followed [23] to use 150 images as the query images and the transformed versions of the query images as the database images in the image database. The images we used are downloaded from [8]. They are photography in many different themes. For each image, 8 different transformations are applied to produce 8 different database images. The transformations include the followings:

1. Three cropping transformations done by cropping the query image by 50%, 70%, and 90% respectively. All cropped images are resized back to original size. These image capture both the cropping and the resizing transformations.

2. Three shearing transformations done by applying an affine warp on the query image along the x axis by 5°, 10°, and 15° respectively.

3. Two contrast changing transformations done by increasing the contrast of the query image by 3× and decreasing it by 3× respectively.

Since each query image produces 8 transformed versions, there are altogether 1200 database images. Before building an index, we extract keypoints from each image. Each image contains hundreds of keypoints. Thus, the keypoint database contains 1 million of keypoints.

The parameters k, m and L in E^2LSH are set to be 26, 28, and 50 respectively. All of our experiments use a Intel P4 3.2GHz machine with 2GB of memory running on Fedora Core 3 (Linux Kernel 2.6).

4.4.1 Evaluation Metrics

The performance of our system is evaluated using Receiver Operating Characteristic (ROC). We define a correct match as a match between a query image and one of its transformed versions in the database. Any other matches are false matches. The recall and precision rate are defined as follows:

$$recall = \frac{number\ of\ correct\ matches}{total\ number\ of\ correct\ matches}$$
$$precision = \frac{number\ of\ correct\ matches}{total\ number\ of\ matches}$$

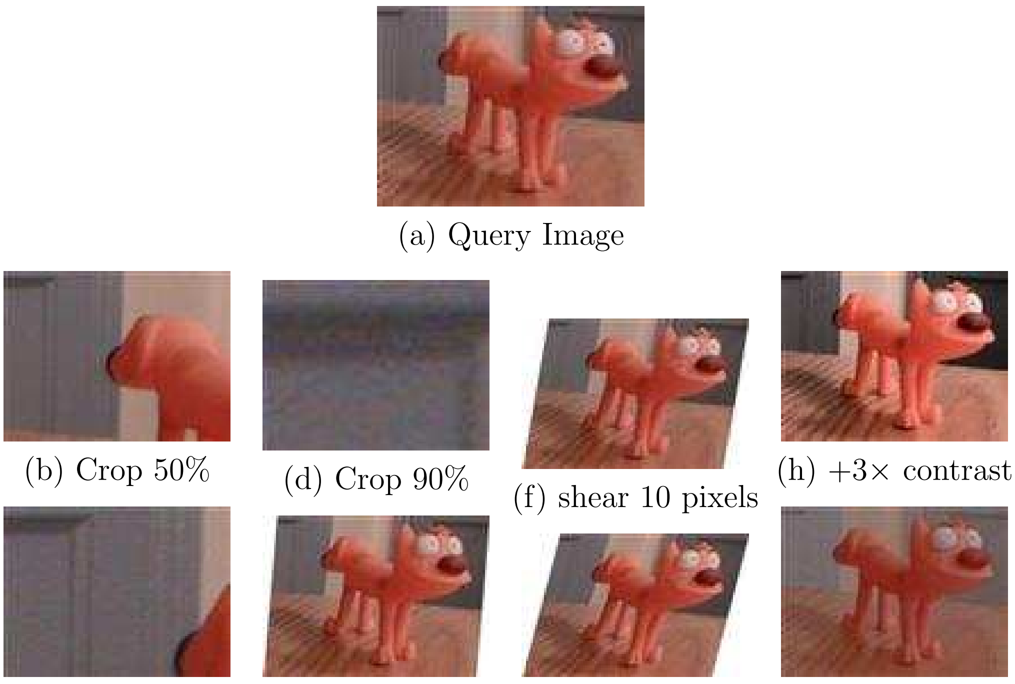
Intuitively, we want both recall and precision rate to be high.

4.4.2 Results

Preliminary Comparison on the Three Matching Strategies

In this experiment, we compare the performance of our system with different matching strategies by making a query using the query image shown in 4.1. The eight transformed versions are also shown in 4.1.

The setting of each matching strategy is summarized in the table 4.1. The performance comparison is shown in the table 4.2. From table 4.2, we can see that the system using threshold based matching performs the worst. It cannot give any correct match. It is because there are numerous candidate keypoint matches lie within the threshold R. The image voting seems like a randomized result and thus no correct match result. As for the proposed k-NNRatio matching strategy, it gives one more correct match than k-NN matching strategy. That correctly matched image is the 50%-cropped version of the query image which is difficult to match correctly. There should have a few keypoints voting this image. However, under the proposed k-NNRatio matching strategy, the influence of a few keypoints can be large in image voting. This contributes to the higher recall rate of the k-NNRatio matching.



(a) Query Image

(b) Crop 50% (d) Crop 90% (f) shear 10 pixels (h) $+3\times$ contrast

(c) Crop 70% (e) shear 5 pixels (g) shear 15 pixels (i) $-3\times$ contrast

Figure 4.1: The Query Image and its eight transformed versions

Matching strategies	Settings
Threshold based	R = 350
k-NN	R = 350, K = 10
k-NNRatio	R = 350, K = 10, a = b = c = 1

Table 4.1: Table summarizes the experiment's settings.

Matching strategies	b	c	d	e	f	g	h	i
Threshold based	0	0	0	0	0	0	0	0
k-NN	0	0	0	67	57	75	17	68
k-NNRatio	10	0	0	41	7	164	10	65

Table 4.2: Table summarizes the results of query under each matching strategy without using any verification process. If any of the images (b) - (i) are among the top 10 during the image voting step, this table will show the support / weight of that image in the column that represents that image and in the row that represents the matching strategy in use.

Result of Orientation Verification

In the this experiment, we evaluate the performance of the orientation verification process. We query the database using the 150 query images (a) with orientation verification only, (b) with affine geometric verification only, and (c) with both verifications. The total number of possible matches is $150 \times 8 = 1200$. The experiment setting is presented in table 4.3.

The results are summarized in the table 4.4. As seen from the table, the orientation verification contributes to further improve the recall and precision rate.

Parameter Name	Value
Matching strategy	*k*-NN matching strategy
R	350
k	10

Table 4.3: Experiment setting.

Determine parameters a, b, and c of *k*-NNRatio

To determine the parameters a, b, and c of *k*-NNRatio matching strategy, we compare the performance of our system under different choice of a, b, c,

Verification	# correct matches	# false matches	recall	precision
(a)	975	471	81%	67%
(b)	1001	430	83%	70%
(c)	1011	301	84%	77%

Table 4.4: Table summarizing the # correct matches, # false matches, recall, and precision rate.

and other system parameters listed in the table 4.5.

Parameter Name	Value
Matching strategy	k-NNRatio matching strategy
Verification	Both
R	350
K	10 - 40
N	10 - 40
a	2 - 10
b	0.11 - 1.00
c	1.00 - 8.30

Table 4.5: Experiment setting. Note that the value N limits the maximum number of images being voted and returned to the user.

Part of the experimental result is shown in table 4.6. This is the part that shows the best setting of the system. The best *a, b, and c* parameter as determined by our experiment are **4**, **0.2**, and **4** respectively. The best *recall* and *precision* rate are **87**% and **85**% respectively. Comparing these results with that performed using k-NN matching strategy, we can see that the recall and precision rate is increased by **3**% and **8**% respectively. As seen from table 4.6, by using k-NNRatio matching, our system can perform 99% precision rate with just a bit lower recall rate, 84%, which is still higher than that of k-NN matching.

Running Time

The speed of E^2LSH is fast. To query 100 keypoints in a 1 millions keypoint database, the query takes only 10 seconds to finish. The only problem is that its memory requirement is large and thus it causes the scalability problem discussed before.

N	a	b	c	#correct	#false	recall	precision
10	4	1	8.3	933	570	0.78	0.62
10	4	1	4	1041	459	0.87	0.69
10	4	1	2.6	1037	456	0.86	0.69
10	4	1	2	1031	454	0.86	0.69
10	4	1	1.6	1029	449	0.86	0.7
10	4	1	1.3	1029	445	0.86	0.7
10	4	1	1.1	1029	440	0.86	0.7
10	4	1	1	1024	443	0.85	0.7
10	4	0.33	8.3	944	539	0.79	0.64
10	4	0.33	4	1041	205	0.87	0.84
10	4	0.33	2.6	1031	71	0.86	0.94
10	4	0.33	2	1024	33	0.85	0.97
10	4	0.33	1.6	1021	26	0.85	0.98
10	4	0.33	1.3	1017	15	0.85	0.99
10	4	0.33	1.1	1017	14	0.85	0.99
10	4	0.33	1	1015	10	0.85	0.99
10	4	0.2	8.3	950	528	0.79	0.64
10	4	0.2	4	1040	177	0.87	0.85
10	4	0.2	2.6	1027	66	0.86	0.94
10	4	0.2	2	1022	31	0.85	0.97
10	4	0.2	1.6	1017	28	0.85	0.97
10	4	0.2	1.3	1016	19	0.85	0.98
10	4	0.2	1.1	1018	16	0.85	0.98
10	4	0.2	1	1015	18	0.85	0.98
10	4	0.14	8.3	949	527	0.79	0.64
10	4	0.14	4	1039	176	0.87	0.86
10	4	0.14	2.6	1027	67	0.86	0.94
10	4	0.14	2	1022	40	0.85	0.96
10	4	0.14	1.6	1018	27	0.85	0.97
10	4	0.14	1.3	1016	21	0.85	0.98
10	4	0.14	1.1	1013	21	0.84	0.98
10	4	0.14	1	1015	13	0.85	0.99
10	4	0.11	8.3	949	528	0.79	0.64
10	4	0.11	4	1037	192	0.86	0.84
10	4	0.11	2.6	1027	76	0.86	0.93

Table 4.6: A portion of the performance evaluation result using different setting of value.

Comparing with Exhaustive Linear Search technique, LSH costs some setup time building LSH hash tables, R-NN data structure, etc. However, once the setup is ready, the built LSH hash tables and R-NN data structure can be used by all the query images. Moreover, during the crawling process of web image crawler, when an image is downloaded, what the system need to do is to incrementally add the feature vectors of the downloaded image into the LSH hash tables. Since the number of feature vectors in one image is not high, the speed of the incremental adding process is fast. By sacrificing a small amount of time for adding the feature vectors into the LSH hash tables, the system can perform fast IND detection.

4.4.3 Summary

We have demonstrated our IND detection system is effective in detecting near-duplicate images in a large database with high recall and precision rate. The proposed k-NNRatio matching strategy has been shown to be better than k-NN matching strategy in terms of system's recall rate and precision rate. The proposed orientation verification scheme is also shown to be effective in removing probable false matches and this is also reflected in the system's recall rate and precision rate.

□ **End of chapter.**

Chapter 5

Shape-SIFT Feature Descriptor

5.1 Overview

Approaches based on invariant local descriptors have been widely employed in many computer vision applications, including automatic panorama stitching, image retrieval, and object class recognition [37]. Although these approaches are generally more robust to clutter background than those using global descriptors, their recognition performance is still significantly reduced when the background color changes. This is because a change in background color will distort the local descriptors sampled near the boundary between an object and the background. The distortion appears frequently on the descriptors sampled in large feature regions. It degrades the recognition rate significantly when the target objects are textureless, since the success in recognizing descriptors near the contours of objects the becomes critical. On the other hand, invariance of descriptors to changes in object color is not regarded as important in the recently proposed local descriptors. However, objects in the images of the same semantic topic can always share the same shape with different colors. A black car and a white car of the same model, for example, can both belong to car category or category of car with certain model number. This raises a need to design a local descriptor that is more robust to such a change for image retrieval purpose.

In this chapter we propose a new descriptor resembling Scale Invariant Feature Transform (SIFT) [32] that is also invariant to background and object color changes. By object color changes, we mean the colors of different parts of an object may change independently. We observe that if the shape of object remains the same, significant changes in background or object color can cause some SIFT descriptors to change significantly because of the flipping of the image gradient orientations. The flipping of gradient orientation affects not only the description process but also the orientation assignment process of

SIFT. We propose some methods to handle these problems, and design a new descriptor, Shape-SIFT, for a comprehensive solution. We will also present how experiments are carried out and the evaluation results.

5.2 Related Work

Since Schmid and Mohr [49] introduce invariant local descriptor for solving image matching problems, many research tasks have been performed to further improve the detection, description and matching process of local descriptors in three interacting aspects: the distinctiveness, the extent of invariance, and the speed of the process.

SIFT descriptor [32] is one of the state-of-the-art descriptors that is shown to be very robust to many image transformations. It is a 4×4 array of histograms, each has 8 orientation bins. Each sample point in the detected feature region is added to the corresponding bin according to its gradient orientation and is weighted by its gradient magnitude. Gradient magnitudes $m(x, y)$ and gradient orientations $\theta(x, y)$ of each image sample point, $L(x, y)$, are computed using pixel differences:

$$\begin{aligned} m^2(x,y) =& (L(x+1,y) - L(x-1,y))^2 + \\ & (L(x,y+1) - L(x,y-1))^2 \\ \theta(x,y) =& \tan^{-1}\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)} \end{aligned} \tag{5.1}$$

To achieve rotation invariance, the coordinates of the descriptor and the gradient orientations within the feature region are rotated according to the *canonical orientation* which is the most frequent orientation within that region. After that, the 16×16 sample region around each keypoint is divided into 16 subregions with 4×4 samples in each. A 36-bin orientation histogram covering $360°$ is then created in each subregion and SIFT descriptor is formed by combining the 16 histograms.

Recently, several SIFT-based descriptors were proposed and were shown in [22] to be superior to many other types of local feature descriptors on the feature matching task. They include PCA-SIFT [45] and GLOH [22]. PCA-SIFT descriptor is a vector of image gradients in x and y directions computed in the feature region detected by the SIFT's detection technique and is reduced to 36 dimensions with PCA. GLOH descriptor is another extension to SIFT. It computes the SIFT descriptor with a different arrangement of subregions and employs PCA to reduce its dimension to 128. Other descriptors that show good performance include Shape Context [4], which is a 3D histogram of edge point locations and orientations under the implementation of Mikolajczyk [22].

5.3 SHAPE-SIFT Descriptors

According to our preliminary test, SIFT is robust to a small degree of change in background and object color. This indicates that both the gradient orientations and gradient magnitudes are robust to a small change due to the thresholding and normalization process of SIFT. However, as the change becomes larger, the gradient orientations of the sample points along the contour may flip. This causes significant changes in the features' orientation histograms, leading to a great drop in the matching performance. We attack this problem as follows.

5.3.1 Orientation assignment

Our algorithm, called SSIFT, takes the output of SIFT as the input. From the keypoint detection and orientation assignment result of SIFT, we recalculate the gradient orientation $\theta(x, y)$ of each image sample point with the following equation:

$$\theta(x, y) = \tan^{-1} \left| \frac{L(x, y+1) - L(x, y-1)}{L(x+1, y) - L(x-1, y)} \right| \tag{5.2}$$

Since we take the absolute value on the fraction of pixel intensity difference, $\theta(x, y)$ covers only 180° range of orientations. As flipping of gradient orientations does not affect the values of $\theta(x, y)$, it is insensitive to the change in background and object color. On the other hand, gradient magnitude $m(x, y)$ is calculated in the same way as SIFT in Eq.(5.1).

5.3.2 Canonical orientation determination

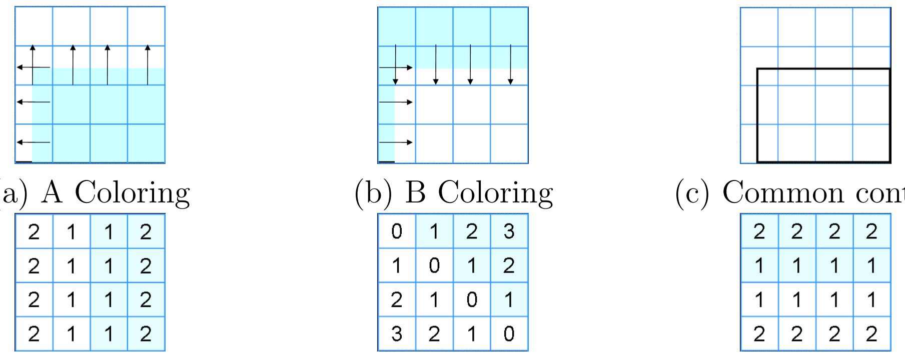
After each image sample point is assigned to a new gradient orientation, a 18-bin orientation histogram covering 180° is created at each keypoint. Sample points around a keypoint are added to an orientation histogram according to their $\theta(x, y)$. Similar to SIFT, we determine the canonical orientation of the keypoint based on the peak orientation bin in the histogram. However, since gradient orientation covers only 180° range of orientations, we cannot simply take the peak orientation as the canonical orientation of the keypoint. We thus propose the following approaches to solve this problem.

Approach 1 - Duplication
The simplest method to tackle this problem is to create another keypoint for the same sample point but with different canonical orientations. The canonical orientations of the two keypoints will be θ_p and $180° + \theta_p$, respectively. Since every keypoint is duplicated, the number of features stored in the database is doubled. This approach solves the problem without lowering the matching performance but it significantly increases the storage requirement, which is not desirable.

Approach 2 - By the distribution of peak orientation
Canonical orientations can be determined using an orientation dependent statistical value, as long as the statistical value responds to the two candidate canonical orientations (θ_p and $180° + \theta_p$) differently and is invariant to common image transformations. We propose to use the distribution of peak sample points around the keypoint, which have their orientations falling into the peak orientation bin as the statistical value. This value is calculated based on the concept behind the first image moment. First, the coordinates of the descriptor and the gradient orientations of the sample points are rotated by θ_p. Then the feature region is divided into two halves, region A and B. The statistical value corresponding to region A, $Mass_A$ and that to region B, $Mass_B$, are then computed by the following equations:

$$Mass_S = \sum_{x,y \in S} [w(x,y) \times m_p(x,y)] \; where \; S \in \{A, B\}$$

where $m_p(x,y)$ is the gradient magnitudes of a peak sample point and $w(x,y)$ is a weighting function depending on the relative position of the point to the keypoint center. The canonical orientation is then determined by the following rules: (1) If $Mass_A > k \times Mass_B$, then the canonical orientation is θ_p, else (2) if $Mass_B > k \times Mass_A$, then it is $180° + \theta_p$. Otherwise (3), we duplicate the keypoint such that each keypoint takes one of the two candidate canonical orientations. If $Mass_A$ and $Mass_B$ are too close to each other, a significant transformation in feature may result in a totally different canonical orientation, leading to different feature representations. Thus, the difference between $Mass_A$ and $Mass_B$ has to be large enough. This is ensured by an empirical threshold `k`. There are many different ways to divide and weight the region, four of which show good performance in experiment: (1) *Right-Left Halving Scheme* divides the feature region *vertically* around the center. Each 16×16 feature region is divided into 4 x 4 subregions. The weight of each subregion is represented by the weighting function $w(x,y)$. The weighting function of this scheme and the following two schemes are shown in Fig.5.1. As pixels near the dividing line may easily fall into the other side and add up to its mass, these pixels get lower weights.



(a) A Coloring (b) B Coloring (c) Common contour

2	1	1	2
2	1	1	2
2	1	1	2
2	1	1	2

(d) R-L Halving

0	1	2	3
1	0	1	2
2	1	0	1
3	2	1	0

(e) Diagonal Halving

2	2	2	2
1	1	1	1
1	1	1	1
2	2	2	2

(f) U-D Halving

Figure 5.1: (a-c) A simple illustration of different coloring to image gradient orientations. (d-e) Different halving and weighting schemes. The shaded halve is region A while the unshaded halve is region B.

(2) *Diagonal Halving Scheme* divides the region *diagonally*.
(3) *Up-Down Halving Scheme* divides the region *horizontally*.
(4) We further suggest *Hybrid Halving Scheme* which combines the above three schemes. The $Mass_A$ and $Mass_B$ of this scheme is defined as the multiple of the mass of the above three schemes, $Mass_A$ and $Mass_B$ respectively. The hybrid scheme can further boost the filtering performance.

To evaluate the performance of these halving schemes and to find the best threshold `k` for each of them, we conduct assessment by two ratios, filtering ratio and matching ratio. Filtering ratio equals (1 - # duplicated features / # features) while matching ratio equals (# correctly matched features / Max. # correctly matched features). "Max. # correctly matched features" is the maximum overall halving schemes in many different choices of the threshold `k`. We prefer both the filtering ratio and the matching ratio to be close to 1. That is, fewer features are to be duplicated and most determined canonical orientations are stable. Fig.5.2 shows the result used to examine the effect of varying the threshold `k` to filtering and matching ratios. The assessment is performed on data sets from VGG [1] that capture common image transformations. As expected, the hybrid scheme achieves excellent filtering ratio but it degrades the matching ratio quite significantly. On the other hand, the diagonal halving scheme has a high filtering ratio at any threshold `k` and achieves nearly the best matching ratio at $k = 1.3$. Thus, we adopt the diagonal halving and weighting scheme in creating SSIFT descriptor and fix k to be 1.3.

[1]http://www.robots.ox.ac.uk/ vgg/research/affine/index.html

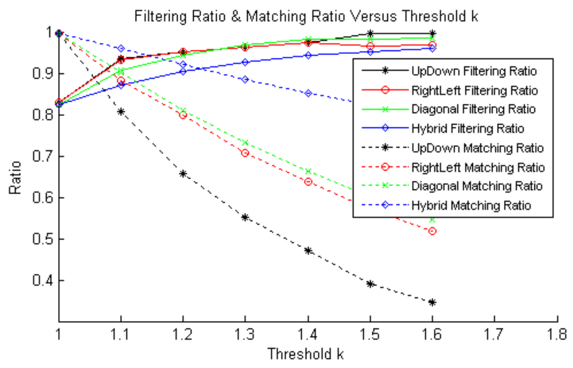
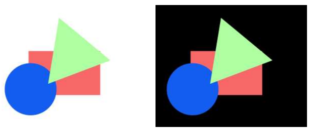


Figure 5.2: Find the best halving scheme and threshold k.

5.3.3 Keypoint descriptor

After the sample region around each keypoint is rotated to its canonical orientation, it is divided geometrically into 4×4 subregions arranged in a grid pattern like those in Fig.5.1. Each subregion contains 4×4 sample points. Inside each subregion, we build a 4-bin orientation histogram covering 180° by adding all sample points within that subregion into the bins corresponding to their gradient orientations. Each point added is weighted by its gradient magnitudes and a Gaussian-weighted circular window fit inside the region. At last the 64-element descriptor is thresholded and normalized to unit length so that it is less sensitive to change in illumination. We call the descriptor built from these histograms SSIFT-64. Since our descriptor is more invariant to change in background and object color than SIFT, it is inevitably a bit less distinctive than SIFT in some cases. To compensate the drop in distinctiveness, another 4-bin orientation histogram is built in each subregion and appended to the descriptor. A matching mechanism is designed to retain the performance of our descriptor on changing background and object color while boosting the performance on other cases. These 4-bin orientation histograms contain north, east, south and west orientation bins, covering 360° range of orientations. The gradient orientation $\theta(x, y)$of each sample point covers the 360° range of orientations and is calculated by Eq.(5.1). We name this descriptor SSIFT-128. Matching SSIFT-64 can be performed by exhaustive searching in the database for the descriptor with the smallest Euclidean distance ratio [32]. The matching mechanism for SSIFT-128 descriptor is as follow: we find the best match $match_{64}$ using the first 64 elements of SSIFT-128 and calculate the distance ratio dr_{64}. Then we refine the match using also the last 64 elements and calculate the distance ratio dr_{128}. We further compare the value of dr_{64} and dr_{128}. If dr_{64} is smaller, then we take the $match_{64}$ as the best match of the descriptor. Otherwise, we

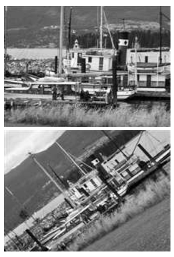

(a) Polygons

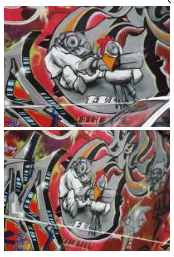
(b) PSP

(c) dragon

(e) basmati

(f) blurring


(g) rotation


(h) view change

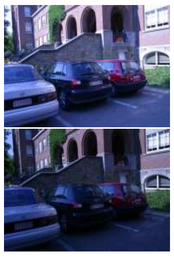

(i) dimming

Figure 5.3: (a) shows two synthetic images that are created by changing the background color of three colored polygons. They capture background color change. (b) shows a sample pair of real images with color difference. (c) shows two real sample images that capture background color change. (d) shows a basmati box [32] and a scene containing an inverted color basmati box. (f-i) shows sample images with different image transformations from VGG used in [22].

take $match_{128}$. This matching mechanism makes the overall performance of SSIFT-128 better than SSIFT-64, as shown in the next section.

5.4 Performance Evaluation

We evaluate the performance of our descriptor on synthetic images and real images with background and object color changes as well as with different geometric and photometric transformations. Sample images of our data sets are shown in Fig.5.3. We evaluate the performance of our local descriptor with other descriptors on a keypoint matching problem using the same evaluation criterion in [22, 45]. Our evaluation criterion differs from [22] in that: (1) A correct match is the match with the largest distance ratio but not the match with distance between the query keypoint and matched keypoint

Table 5.1: Experimental Results

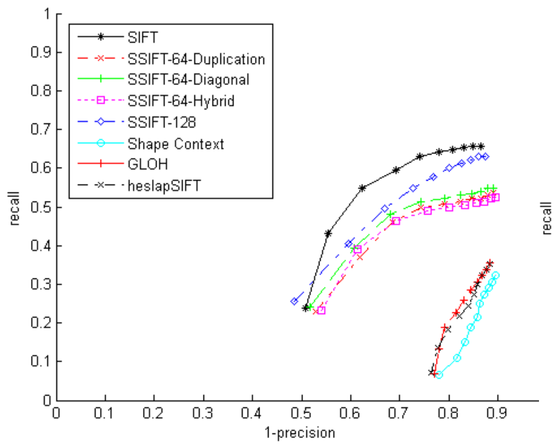
Data Set	Primitives		PSP		Dragon	
Descriptor	recall	1-prec.	recall	1-prec.	recall	1-prec.
SSIFT-128	0.58	0.55	0.29	0.95	0.89	0.35
SIFT [32]	0.16	0.87	0.00	1.00	0.83	0.38
GLOH [22]	0.11	0.92	0.05	0.99	0.53	0.73
Shape Context [22]	0.10	0.93	0.05	0.99	0.52	0.74

Data Set	Basmati		Bikes	
Descriptor	recall	1-prec.	recall	1-prec.
SSIFT-128	0.34	0.88	0.88	0.52
SIFT [32]	0.00	1.00	0.90	0.47
GLOH [22]	0.00	1.00	0.70	0.52
Shape Context [22]	0.00	1.00	0.66	0.55

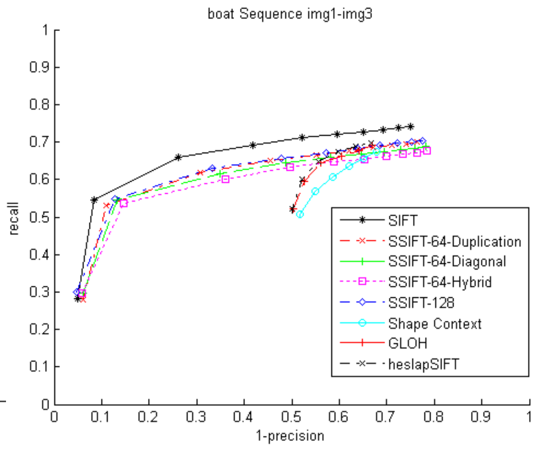
within a threshold distance, since distance ratio criterion is shown to yield better overall results. (2) We do not allow the same descriptor to be repeatedly matched. (3) We also test the original SIFT executable provided by D. Lowe [2]. (4) Over ten thousands different features are added as distracters. Due to the above differences, our experimental results are different from those in [22]. Table 5.1 and Fig.5.4 show the performance of several descriptors on our data sets. We compare the performance of different variants of our descriptor, SIFT implemented by D. Lowe, and SIFT, GLOH and shape context implemented by Mikolajczyk et al. [22]. We adopt the feature detectors that were proposed to use with the corresponding descriptors. That is, the last three descriptors use Harris-Laplacian detector while other descriptors use DOG extrema detector.

Fig.5.4(a-c) shows that SSIFT is much more invariant to changes in background color than other descriptors. Fig.5.4(a-d) shows that SSIFT has similar performance with the state-of-the-art descriptor, SIFT, on images with scale, rotation, viewpoint and illumination changes. Although the precision rate of SSIFT is a bit poorer than SIFT in Fig.5.4(a-d), its recall rate is as high as SIFT.

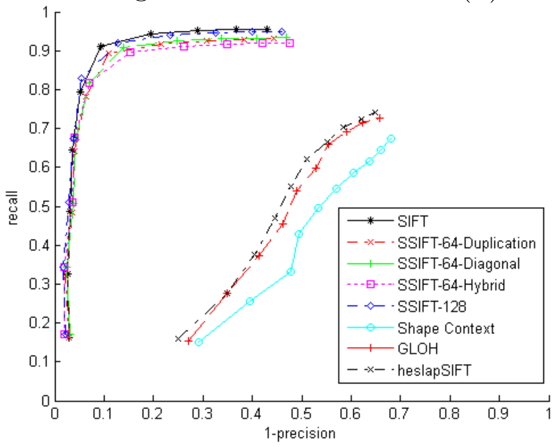
[2]http://www.cs.ubc.ca/ lowe/keypoints/



(a) viewpoint change



(b) rotation



(c) dimming

Figure 5.4: (a-c) Target images have undergone the labeled transformations. They correspond to the images in Fig.5.3(g-i).

5.5 Summary

This chapter introduces an alternative to SIFT to build orientation histograms for feature descriptor. Our descriptor, SSIFT, is shown to be more invariant to background and object color changes than any other descriptors we tested in the experiments. We propose a new canonical orientation determination process to ensure a consistent representation of each descriptor. The process is shown to be effective in finding a canonical orientation while keeping the adverse effect to feature matching small. Currently, we are investigating alternative ways to extend SSIFT-64 to SSIFT-128. Instead of extending SSIFT-64 with more orientation histograms, we can employ color histograms which may be a better complement to SSIFT-64.

□ **End of chapter.**

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis proposed a scalable content-based image retrieval system for searching images of a specific semantic topic over a very large database. The system is designed to work with a new high dimensional indexing technique: locality-sensitive hashing (LSH). We have addressed some limitations and challenges in applying LSH in our system and suggested a parallel and distributed solution to overcome the problems. We have conducted extensive empirical evaluations on a large testbed of a million images and shown that our solution is fast, accurate, and scalable to very large image database.

To refine the quality of the CBIR search, we proposed and implemented an IND detection system to remove the near-duplicate images during and after the Web image crawling process. The system is accurate and efficient due to the integration of powerful feature detector, descriptor, matching scheme, the new matching strategy, and the new verification process.

Furthermore, the thesis discussed several recent research work on invariant local grayvalue features and evaluated the performance of several popular feature descriptors. We found that SIFT feature descriptor remains the best comparing with other descriptors in the experiment. We then introduced our newly proposed feature descriptor, SSIFT, which extends SIFT feature descriptor to be invariant to the changes in background and object color. We evaluated the performance of our descriptor with SIFT and showed that our descriptor does better in the cases that changes in background and object color occur.

6.2 Future Work

The query processing power of the proposed parallel and distributed CBIR system can further be improved by minimizing the usage of Slaves. Currently, for each search query, the system gets all the Slaves involved and makes all of them busy searching for similar images, no matter whether a similar image can be found in each Slave. If we know there is no data point within a threshold radius from the query point in certain Slave machine, performing similarity search in that Slave would not be necessary. Therefore, if the system can decide *not* to perform similarity search in that Slave, the system can eventually be able to process more search queries at the same time. To achieve this, we need a dispatcher in Master that can *precisely* dispatch a query to a subset of Slave machines. We suggest to apply PCA to reduce the dimension of the query point and then apply k-d tree to determine at which Slaves similar data points of the query point would be located. This dispatcher will also be employed during the partition distribution process to assign the data points of the database to the suitable Slaves.

□ **End of chapter.**

Appendix A

Publication

Published Paper

[1] Yuk-Man Wong, Chu-Hong Hoi and Michael R. Lyu, "An Empirical Study on Large-Scale Content-Based Image Retrieval," in *Proceedings of The 2007 IEEE International Conference on Multimedia and Expo (ICME'2007)*, Beijing, 2007

□ **End of chapter.**

Bibliography

[1] A. Andoni. http://web.mit.edu/andoni/www/LSH/index.html.

[2] J. R. Bach, C. Fuller, and et al. The virage image search engine: An open framework for image management. In *Proc. SPIE Storage and Retrieval for Image and Video Databases*, volume 2670, pages 76–87, 1996.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 322–331. ACM Press, 1990.

[4] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *PAMI*, page 509, 2002.

[5] S.-A. Berrani, L. Amsaleg, and P. Gros. Robust content-based image searches for copyright protection. In *MMDB '03: Proceedings of the 1st ACM international workshop on Multimedia databases*, pages 70–77, New York, NY, USA, 2003. ACM Press.

[6] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proc. 20th Annual Symposium on Computational Geometry*, pages 253–262, New York, NY, USA, 2004.

[7] J. Dowe. Content-based retrieval in multimedia imaging. In *Proc. SPIE Storage and Retrieval for Image and Video Databases*, 1993.

[8] DPChallenge. http://www.dpchallenge.com/.

[9] R. Egas, D. P. Huijsmans, M. S. Lew, and N. Sebe. Adapting k-d trees to visual retrieval. In *Visual Information and Information Systems*, pages 533–540, 1999.

[10] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[11] M. Flickner, H. Sawhney, W. Niblack, J. Ashley, Q. Huang, B. Dom, M. Gorkani, J. Hafner, D. Lee, D. Petkovic, D. Steele, and P. Yanker. Query by image and video content: The qbic system. *IEEE Computer*, 28(9):23–32, 1995.

[12] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[13] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey88*, pages 147–152, 1988.

[14] T. Hertz, N. Shental, A. Bar-Hillel, and D. Weinshall. Enhancing image and video retrieval: Learning via equivalence constraints. In *Proc. of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[15] S. C. Hoi, W. Liu, M. R. Lyu, and W.-Y. Ma. Learning distance metrics with contextual constraints for image retrieval. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR2006)*, New York, US, June 17–22 2006.

[16] S. C. H. Hoi and M. R. Lyu. A novel log-based relevance feedback technique in content-based image retrieval. In *Proc. ACM Multimedia Conference*, New York, US, Oct. 10–16 2004.

[17] S. C. H. Hoi, M. R. Lyu, and R. Jin. A unified log-based relevance feedback scheme for image retrieval. *IEEE Trans. KDE*, 18(4):509–524, 2006.

[18] S. Hu. Efficient video retrieval by locality sensitive hashing. In *Proc. IEEE ICASSP*, volume 2, pages 449–452, 2005.

[19] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proc. 13th ACM Symposium on Theory of computing*, pages 604–613, New York, NY, USA, 1998.

[20] A. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 29(8):1233–1244.

[21] A. Jain and A. Vailaya. Shape-based retrieval: A case study with trademark image databases, 1998.

[22] C. S. K. Mikolajczyk. A performance evaluation of local descriptors. *PAMI*, 27:1615–1630, 2005.

[23] Y. Ke, R. Sukthankar, and L. Huston. An efficient parts-based near-duplicate and sub-image retrieval system. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 869–876, New York, NY, USA, 2004. ACM Press.

[24] W.-C. Lai, C. Chang, E. Chang, K.-T. Cheng, and M. Crandell. Pbir-mm: multimodal image retrieval and annotation. In *MULTIMEDIA '02: Proceedings of the tenth ACM international conference on Multimedia*, pages 421–422, Juan-les-Pins, France, 2002.

[25] W.-C. Lai, E. Chang, and K.-T. Cheng. An anatomy of a large-scale image search engine. In *poster proceedings of World Wide Web Conference.*

[26] M. S. Lew. Next-generation web searches for visual content. *Computer*, 33(11):46–53, 2000.

[27] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2(1):1–19, 2006.

[28] X. Li, L. Chen, L. Zhang, F. Lin, and W.-Y. Ma. Image annotation by large-scale content-based image retrieval. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 607–610, Santa Barbara, CA, USA, 2006.

[29] H. Ling and D. Jacobs. Deformation invariant image matching. In *ICCV*, pages II: 1466–1473, 2005.

[30] H. Ling and K. Okada. Diffusion distance for histogram comparison. In *CVPR06*, 2006.

[31] D. G. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

[32] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60(2):91–110, 2004.

[33] B. Manjunath, P. Wu, S. Newsam, and H. Shin. A texture descriptor for browsing and similarity retrieval.

[34] B. S. Manjunath and W. Ma. Texture features for browsing and retrieval of image data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI - Special issue on Digital Libraries)*, 18(8):837–42, Aug 1996.

[35] S. Mehrotra, Y. Rui, O.-B. Michael, and T. S. Huang. Supporting content-based queries over images in mars. In *Proc. of IEEE Int. Conf. on Multimedia Computing and Systems*, 1997.

[36] B. M. Mehtre, M. S. Kankanhalli, and W. F. Lee. Shape measures for content based image retrieval: a comparison. *Inf. Process. Manage.*, 33(3):319–337, 1997.

[37] K. Mikolajczyk, B. Leibe, and B. Schiele. Multiple object class detection with a generative model. In *CVPR*, pages I:26–36, 2006.

[38] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27.

[39] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *ICCV*, pages 525–531, 2001.

[40] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool. A comparison of affine region detectors. *IJCV*, 65(1-2), 2005.

[41] W. Niblack, R. Barber, and et al. The QBIC project: Querying images by content using color, texture and shape. In *SPIE Storage and Retrieval for Image and Video Databases*, 1994.

[42] A. Pentland, R. Picard, and S. Sclaroff. Photobook: Content-based manipulation of image databases. *IJCV*, 18(3):233–254, 1996.

[43] A. Qamra, Y. Meng, and E. Y. Chang. Enhanced perceptual distance functions and indexing for image replica recognition. *IEEE Trans. on PAMI*, 27(3):379–391, 2005.

[44] T. Quack, U. Mönich, L. Thiele, and B. S. Manjunath. Cortina: a system for large-scale, content-based web image retrieval. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 508–511, New York, NY, USA, 2004. ACM Press.

[45] Y. K. Rahul. Pca-sift: A more distinctive representation for local image descriptors. In *CVPR*, pages 511–517, 2004.

[46] N. Recipes. http://www.numerical-recipes.com/.

[47] Y. Rui, T. Huang, and S. Chang. Image retrieval: current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10(4):39–62, Apr. 1999.

[48] Y. Rui, T. S. Huang, and S. Mehrotra. Relevance feedback techniques in interactive content-based image retrieval. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 25–36, 1998.

[49] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *PAMI*, 19(5):530–535, May 1997.

[50] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Trans. on PAMI*, 22(12):1349–1380, 2000.

[51] J. R. Smith and S.-F. Chang. Visualseek: A fully automated content-based image query system. In *ACM Multimedia*, pages 87–98, 1996.

[52] M. Swain and D. Ballard. Indexing via color histograms. In *DARPA90*, pages 623–630, 1990.

[53] S. Tong and E. Chang. Support vector machine active learning for image retrieval. In *MULTIMEDIA '01: Proceedings of the ninth ACM international conference on Multimedia*, pages 107–118, Ottawa, Canada, 2001.

[54] D. A. White and R. Jain. Similarity indexing: Algorithms and performance. In *Storage and Retrieval for Image and Video Databases (SPIE)*, pages 62–73, 1996.

[55] Y. M. Wong, S. C. Hoi, and M. R. Lyu. An empirical study on large-scale content-based image retrieval. In *IEEE International Conference on Multimedia & Expo (ICME2007)*, Beijing, China, July 2007.

[56] P. Wu, B. Manjunath, S. Newsam, and H. Shin. A texture descriptor for browsing and similarity retrieval. *Journal of Signal Processing: Image Communication*, 16(1-2):33–43, Sep 2000.

[57] D.-Q. Zhang and S.-F. Chang. Detecting image near-duplicate by stochastic attributed relational graph matching with learning. In *MULTIMEDIA '04: Proceedings of the 12th annual ACM international conference on Multimedia*, pages 877–884, New York, NY, USA, 2004. ACM Press.