REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

The Eff^2 *Image Retrieval System Prototype*

Sigurður H. Einarsson, Ragnheiður Ýr Grétarsdóttir,
Björn Þór Jónsson, Laurent Amsaleg

RUTR-CS05003 — November 2005

Reykjavík University - Department of Computer Science

Technical Report

ISSN 1670-5777

REYKJAVÍK UNIVERSITY
HÁSKÓLINN Í REYKJAVÍK

The *Eff*^{2} Image Retrieval System Prototype

Sigurður H. Einarsson*, Ragnheiður Ýr Grétarsdóttir*,
Björn Þór Jónsson*, Laurent Amsaleg†

Technical Report RUTR-CS05003, November 2005

Abstract: Content-based image retrieval has become increasingly important in recent years. In the *Eff*^{2} project we have been working with one of the most advanced image description schemes available, namely a fine-grained image recognition scheme based on local descriptors. While this description scheme has been used successfully in the demanding application of content-based image copyright protection, query processing in this approach is very slow. This report describes the *Eff*^{2} image retrieval system prototype, which is designed to facilitate user interaction with the local descriptor search process. We demonstrate, through experimental results, that the time to obtain a good answer may be reduced by over 85% by allowing the user to examine intermediate results.

Keywords: Image retrieval; Local descriptors; User interface; Approximate searches; Result quality.

(Útdráttur: næsta síða)

This technical report is an extended version of a paper that appeared in the IASTED DBA 2005 conference [EGJA05].

This project is part of the *Eff*^{2} project on *Efficient and Effective Image Retrieval* (see http://datalab.ru.is/eff2). The *Eff*^{2} project is a cooperation between researchers at the IRISA laboratory in Rennes, France and Reykjavík University, Iceland, and is partially supported by Rannís Technical Research Grant 030290004 and EGIDE Jules Verne Travel Grant 4-2003.

* Reykjavík University, Ofanleiti 2, IS-103 Reykjavík, Iceland. sigurdure/ragnheidur/bjorn@ru.is
† IRISA–CNRS, Campus de Beaulieu, 35042 Rennes, France. laurent.amsaleg@irisa.fr

HÁSKÓLINN Í REYKJAVÍK
REYKJAVÍK UNIVERSITY

Frumgerð *Eff*^{2} Myndleitarkerfisins

Sigurður H. Einarsson, Ragnheiður Ýr Grétarsdóttir,
Björn Þór Jónsson, Laurent Amsaleg

Tækniskýrsla RUTR-CS05003, Nóvember 2005

Útdráttur: Mikilvægi innihaldsháðrar myndleitar hefur aukist mjög undafarin ár. Í *Eff*^{2} verkefninu höfum við unnið með eina af bestu myndlýsingaraðferðum sem þekktar eru, fíngerða myndlýsingaraðferð byggða á staðværum lýsingum, sem hefur verið notuð með góðum árangri við höfundarréttarvörn á myndum. Úrvinnsla fyrirspurna með þessari aðferð er hins vegar mjög hægvirk. Þessi skýrsla lýsir viðmóti *Eff*^{2} myndleitarkerfisins, sem er hannað til að gefa notendum innsýn í leitarferlið. Sýnt er með tilraunaniðurstöðum að stytta má tímann sem líður þar til góðar niðurstöður fást um meira en 85% með því að leyfa notendum að skoða milliniðurstöður.

Lykilorð: Myndaleit; Staðværir lýsingar; Notendaviðmót; Nálgunarleit; Gæði svara.

(Abstract: previous page)

Contents

1 Introduction

In recent years, content-based image retrieval has become more and more important in many fields, such as medicine, geography, weather forecasting and security. Content-based image retrieval is typically implemented by mapping the images to multi-dimensional *descriptors*, which are then used in similarity searches to determine which images in the collection are most similar to a query image. Many different descriptors have been proposed, ranging from traditional "global" descriptors offering a rough description of the overall composition of the image (e.g., see [FBF$^+$94, SS94, HKM$^+$97, MPE]), to advanced "local" descriptors, where each descriptor describes a small portion of the image and the overall picture is described by many such descriptors (e.g., see [FRKV94, Low99, AG01]).

1.1 The *Eff*^{2} Project

In the *Eff*^{2} project we have been working with one of the most advanced image description schemes available, namely a fine-grained image recognition scheme based on the local descriptors proposed in [FRKV94] for gray-scale images and extended to color images in [AG01]. With this description scheme, each image of the collection yields many descriptors (several hundreds for high-quality images), each descriptor describing a small area of the image. To retrieve the images of the collection that are similar to a query image, a k-nearest neighbor query is run for each local descriptor computed on the query image. Each nearest neighbor yields one vote for some image in the collection, and the most similar images are found by aggregating the votes and ranking the images based on the number of similar descriptors.

This description scheme has been used successfully in the demanding application of content-based image copyright protection [BAG03a, BAG03b]. Experiments with the StirMark benchmark [PSR$^+$01] for simulating image pirate activities have shown these descriptors to be very effective, finding well over 90% of the pirated copies and losing only those that have been modified so much as to change the visual content significantly. Query processing in this approach is very slow, however, for two primary reasons.

First, the query processing is very time consuming. In fact, in [AG01] it was shown that a sequential scan of the descriptor collection is as fast as using the most advanced indexing methods of the time. The key observation to be made is that since each query image yields tens or hundreds of descriptors, tens or hundreds of nearest neighbor queries are performed over the descriptor index. For an indexing approach to be profitable, it must therefore require reading and processing only a small portion of a percent of the collection. We are currently investigating several approaches to solving this problem, but they are outside the scope of the current report.

Second, the approach of running the query to completion and then presenting the results does not utilize the capacity of the user. Human perception of images is very accurate, and thus presenting intermediate results to the user of the system may be very beneficial. The

user may, for example, determine that a matching image has indeed been found after only a small portion of the processing time, and at that time choose to stop the search. Alternatively, the user may decide that the intermediate results indicate that further processing is unlikely to yield a good match, and choose to abort the query at that point. In both of these cases, the system would act as a filter, only presenting the most likely candidates to the user, instead of the whole collection. This report describes the *Eff*^{2} retrieval system prototype, which is defined precisely to allow the user such intervention into the search.

1.2 Contributions of this Report

This report, which is an extended version of [EGJA05], presents our approach to facilitating user interaction with the local descriptor search process. It makes the following three primary contributions:

1. The local descriptor search process was encapsulated into a prototype server architecture. At each time, the intermediate results of the search are accessible to the user through a client API.

2. A prototype user interface was implemented that connects periodically (or upon request) to the server, obtains the current status of the search and displays to the user. The user can decide when to stop the search, based on the intermediate results presented.

3. Finally, we conducted experiments that show that intermediate results can indeed have significant value to the users. We also measured the overhead of the search due to client interactions and found that compared to the gains of presenting intermediate results, the overhead is small.

1.3 Overview of the Report

The remainder of the report is organized as follows. Section 2 reviews the local descriptors used in our work and the algorithm used to search the collection. Section 3 presents our approach to facilitating user interaction, including our server implementation, the client API used to control the search, and the client prototype. Section 4 then describes our experiments to validate the assumption that presenting intermediate results is indeed beneficial to the user. Finally, Section 5 presents related work and Section 6 gives our conclusions.

2 Local Descriptor Search

The local descriptors and search process used in our project were presented in [AG01] and are reviewed here. The section first presents briefly the local descriptors, their properties and how they are used to estimate image similarity. Then the query processing algorithm is described in detail.

2.1 Local Descriptors

The creation of the local descriptors of [AG01] proceeds in three steps. First, specific points in the image, called interest points, are selected based on the shape of the image signal at these points. Second, the signal around each interest point is characterized by its convolution with a Gaussian function and its derivates up to the third order. Finally, these derivatives are mixed to enforce invariance properties and to make the descriptors robust to several changes to the images. For details, please see [AG01].

This scheme has been shown to be robust to many different types of image modifications [AG01, BAG03b]. It is largely insensitive to resizing, color variation, cropping, rotation, jpeg-compression, mirroring, etc. For example, since the local descriptors are distributed throughout the image, cropping is handled implicitly. Of course, it is possible to crop the image to the extent that only very few descriptors prevail in the cropped image, but in this case the visual content of the image has severely been altered, often resulting in an uninteresting image.

The current version of the descriptor creation results in 24-dimensional descriptors. The number of descriptors per image can vary significantly, depending on the size, resolution, quality and contents of the images. For typical images, several hundreds of descriptors may be created; for large, high quality images, even more than a thousand descriptors. Computing descriptors over all the images is done off-line. To know which image a descriptor has been computed from, image identifiers are stored together with the descriptors.

The similarity retrieval proceeds as follows. Interest points are first identified in the query image and the corresponding local *query descriptors* computed. The query descriptors are then used to query the descriptor database. For each descriptor of the query image, the system returns the 30 most similar descriptors found in the database (using Euclidean distance for the measure of similarity; for further details of the query processing, see below). The image identifiers of these descriptors indicate the associated image from the collection, and it is straight-forward to count the number of occurrences of each image identifier during the whole retrieval process. Once all the query descriptors have been used to probe the database, the occurrence counters allow the system to rank the candidate images by decreasing similarity, with the image with the most votes considered most similar, the image with the second most votes the second most similar, and so on.

In general, the number of images receiving at least one vote is very large. If the database does not contain any image that is similar to the query image, then the votes of all the images returned are roughly similar and of small values. In contrast, if one or more images are indeed similar, then they have many more votes.

2.2 Similarity Search

2.2.1 Descriptor Database

The system uses two binary data files, the image catalog and the descriptor database. Figure 1 shows the structure of these files.

Image Catalog	
int	*imageid*
char[100]	*location*

Descriptor Database	
int	*imageid*
float[24]	*descriptor*

Figure 1: Structure of database entries

Query Descriptor	
float[24]	*descriptor*
int[30]	*identifiers*
float[30]	*distances*
int	*neighbors*
int	*farthest*

Figure 2: Query descriptor data structures

The image catalog contains, for each image, the identifier of the image and its location. This information is only used when submitting (intermediate or complete) results to the user. The image location is in URL format, so that images can be located on different servers. This feature offloads image delivery from the image retrieval server.

The descriptor database has one entry for each descriptor. Each descriptor entry contains the identifier of the image, followed by 24 real numbers which constitute the descriptor itself. Many descriptors, of course, share the same image identifier.

2.2.2 Search Data Structures

In order to implement a 30-nearest neighbor search for each query descriptor, the data structure shown in Figure 2 is used. Aside from the descriptor itself, there are two arrays with 30 entries, one entry for each nearest neighbor seen so far. The first array contains the image *identifiers* of the 30 nearest image descriptors seen so far in the search, while the second array contains their *distances* from that query descriptor.

The *neighbors* variable indicates how many neighbors are stored at each time. It is initialized to 0, and then increased each time a new neighbor is inserted, until it reaches 30 and both arrays are full. From that time on, an index into these arrays is maintained, that points to the *farthest* (or 30th) neighbor. When processing a new image descriptor from the collection, its distance to that query descriptor is compared to the distance of this farthest neighbor. If it is further away, it is simply discarded. If it is nearer, the identifier of the descriptor and its distance are inserted into the arrays instead of the previous farthest neighbor. The distance array is then scanned to find a new farthest neighbor; the pseudo-code for finding a new farthest neighbor is shown in Figure 3.

Procedure FindFarthest(q)

Input:
Query descriptor (see Figure 2), q

$dist = 0$
for $i = 1 \ldots 30$ **do**
  if $q.distances[i] > dist$ **then**
    $dist = q.distances[i]$
    $q.farthest = i$

Figure 3: Finding Farthest Neighbor.

2.2.3 Search Algorithm

The most efficient search process described in [AG01] is based on a sequential scan of the collection, and proceeds as follows. After reading (part of) the descriptor database from disk, the first image descriptor is compared to all query descriptors. As it is the current nearest neighbor to all of them, it is inserted into the nearest neighbor arrays (*identifiers* and *distances*) for all the query descriptors. The same happens for the next 29 descriptors. After that, each database descriptor is compared to each query descriptor. If it is found to be nearer than the *farthest* descriptor, it replaces that descriptor. Then the database descriptor is compared to the next query descriptor. Figure 4 shows pseudo-code for the search (simplified by assuming 30 neighbors are already in place).

Once all database descriptors have been compared to all query descriptors, processing of the database is complete, and the identifier arrays of all the query descriptors contain all the votes given to individual database images. These votes are then simply summarized and the resulting image list ranked based on the number of votes. This list is then combined with information from the image catalog to return a list of images to the user.

The inverted approach of comparing the database descriptors to the query descriptors is very similar to a nested loops join, where the smaller relation (query descriptors) is the inner relation to optimize the disk processing of the search. In order to optimize disk processing even further, large chunks of disk pages are read into memory at each time (the default setting is 80 pages of 4 KB each).

The CPU processing, however, is the bottleneck of the search, as each database descriptor must be compared against many query descriptors. For each distance calculation, the value of each dimension of the query descriptor must be subtracted from the value of that dimension of the database descriptor, and the result squared. Two simple tricks are used to optimize CPU processing. First, the square root is omitted from the Euclidean distance function, as omitting it does not change the relative ranking of the descriptors. Second, during the calculation of the distance, if at any point the distance becomes greater than the distance to the *farthest* neighbor, the distance calculation is aborted, and a very large num-

Procedure Search(d, q)

Input:
A set of database descriptors, $d[i]$
A set of query descriptors, $q[j]$

for each $d[i]$ **do**
  for each $q[j]$ **do**
    $dist = \text{Distance}(d[i], q[j])$
    if $dist < q[j].distances[q[j].farthest]$ **then**
      $q[j].identifiers[q[j].farthest] = d[i].imageid$
      $q[j].distances[q[j].farthest] = dist$
      FindFarthest$(q[j])$

Figure 4: Searching for Neighbors.

ber is returned instead of the actual distance. Figure 5 shows pseudo-code for the distance calculation using these two tricks.

Note that in the discussion above no assumptions were made about the order of the descriptors within the file. If all the descriptors belonging to the same image are stored together, then all the votes of that image will be found at about the same time. If the descriptors are randomly ordered, however, the votes of individual images will slowly accumulate, leading to better intermediate results. Therefore, we have chosen to order our descriptor database randomly.

3 Facilitating User Interaction

From the description of the search process above, we can observe that at any point in time some of the votes of any given query descriptor may be votes of eventual nearest neighbors, while other votes will at some point be replaced. By recalling that very similar images have many votes, while random images have very few votes, it is likely that the intermediate results will soon give a good idea of the final results. The goal of our work is to allow the user to tap into these intermediate results. Of course, the "user" may in many cases be an automated system, rather than a person. We are primarily interested in supporting the following actions by the user in our prototype:

- Once the user finds that one of the top images is indeed a matching image, the search is ended and that image is chosen for further processing.
- Once the user determines that no image is likely to separate itself from the random images, the search can also be ended, with no similar image found.

Function Distance(d, q)

Input:
A database descriptor, d
A query descriptor, q

Output:
Distance between d and q

$dist = 0$
for $i = 1 \ldots 24$ **do**
  $dist += (d.descriptor[i] - q.descriptor[i])^2$
  if $dist > q.distances[q.farthest]$ **then**
    return *MAXDIST*
return *dist*

Figure 5: Distance Calculation.

- Once the user determines that a particular image is not a match, this image can be removed from further consideration by the user.

The *Eff*^{2} prototype has three main components: client API, image server, and client interface, which were defined to support precisely these interactions. Figure 6 gives an overview of these aspects of our prototype, which are described in the following.

3.1 Client API

The image retrieval client communicates with the image server via the client API. The API is defined and implemented in Java and invoked via RMI. As the server is implemented in C++, the API uses JNI to gain access to C++ functions that implement the functionality of the API. The operations of the client API are explained below:

int Start(char *image) In the current version, the data structure submitted to the server is an array of query descriptors; in future versions, an image may be supplied instead.* The data structures for the search are initialized and a new search process is spawned. An identifier for the search is returned, which is used in subsequent calls to identify the search.

char *Status(int searchid) This function returns the status of the given search. In order to get the intermediate results, the vote arrays must be summarized and ranked.

* Note that the char * data type is generally used to pass complex information, as it is easier to pass through the layers of the API than a complex data structure.

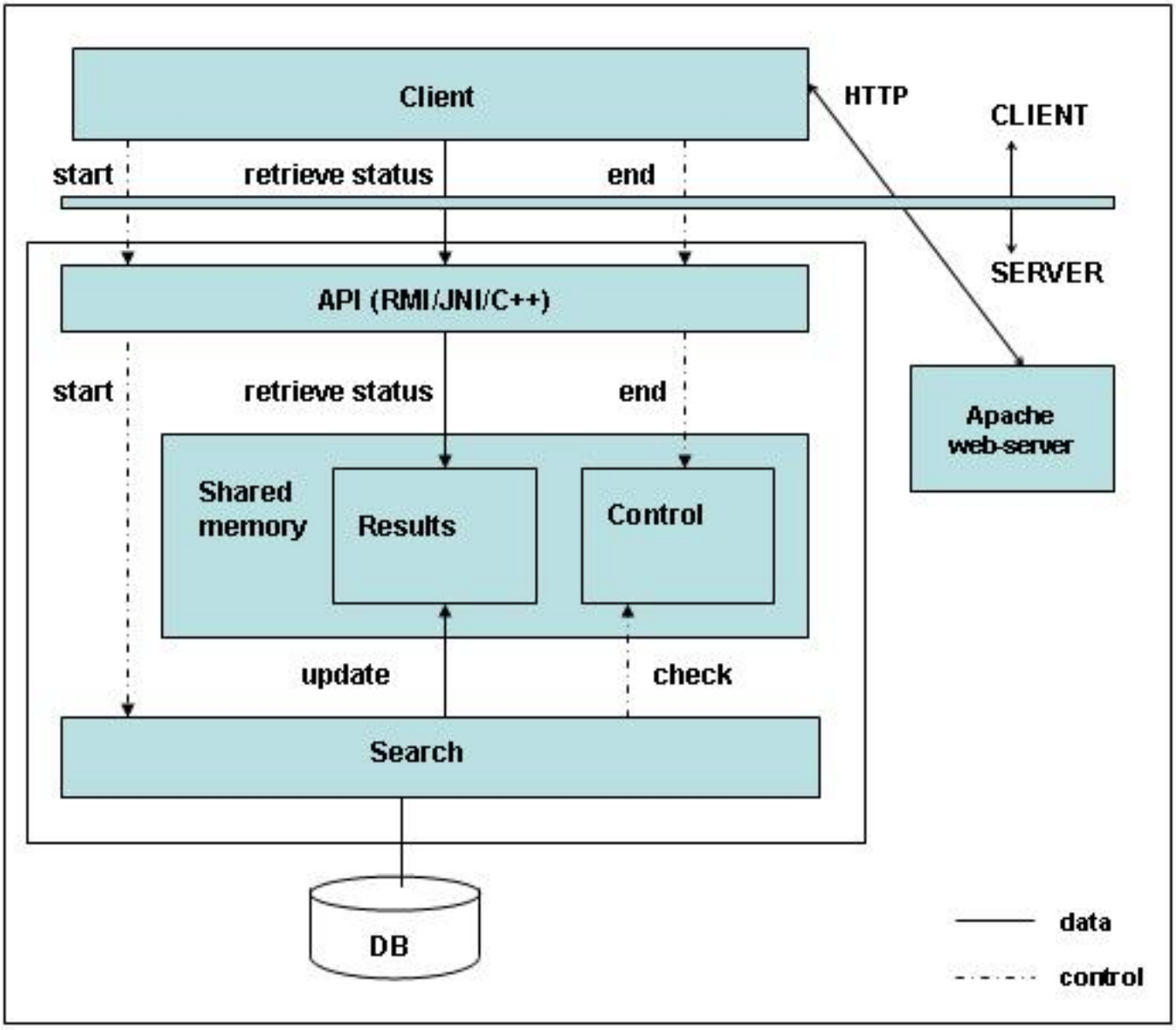

Figure 6: Architecture of the *Eff*^{2} Prototype.

Observe that although the search process may be modifying the results at the same time as the status is taken, there is no need for mutual exclusion, as at most one vote might be corrupted at any time.

void End(int searchid) This function ends the given search, by instructing the search process to stop, regardless of whether the search found a match or not.

void OmitIds(int searchid, char *idlist) This method is used to notify the search process that the user does not want to see these images in the result anymore.

3.2 Image Server

The server uses shared memory to communicate the intermediate results from the search process to the client API. When the server is initialized, a large chunk of shared memory is allocated and subsequently managed via a simple memory manager. All data structures of the search (described in Section 2.2.2) are stored in the shared memory, and pointed to (indirectly) via the search identifier shared by the API calls. A special control mechanism is implemented to stop the search from exploring the database when the user has requested to abort the current search.

3.3 Client Prototype

Most of the functionality of the client prototype is available through the main screen, which is shown in Figure 7. The selected query image is shown on the right, and the six images that are ranked highest in the intermediate result at each time are shown on the left (the screen is refreshed every five seconds).

The identifier of the image and its score are shown beneath each result image. The user can utilize the score to assess the relative merits of images in the result. Additionally, there are three buttons associated with each image.

The check-mark is used by the user to indicate that this image is indeed the image sought after. Pressing such a button ends the search.

The question-mark is intended to initiate a closer comparison of the images. Currently, the query image and the result image are simply shown side by side, but it is easy to foresee all kinds of actions to identify a match, such as stretching the images, rotating and overlaying them—essentially, undoing the potential actions of an image pirate.

The minus button indicates that that particular image is definitely not a match, and should not be shown again. Pressing this button results in a call to the OmitIds() function of the client API.

Additionally, there are several buttons on the right side of the interface. Pressing the **Pause**-button instructs the interface to stop polling the server for updates to the (intermediate) results (when pressed, the **Pause**-button is replaced by a **Play**-button). The **Update**-button can then be used to manually retrieve updated results. The **Stop**-button indicates that no image is found and that query processing should be halted. The **Skip**

Figure 7: The user interface.

all-button instructs the server to never show any of the current top images again. Finally, the **Quit**-button exits the prototype.

3.4 Upcoming Features

The *Eff*^{2} prototype is still quite raw. We expect future versions to be implemented in CORBA, with multiple threads implementing the server (eliminating the need for shared memory and the memory manager). Additionally, we expect to make multiple changes to the client prototype, such as:

- Increasing the number of images shown in the intermediate result, and adding a slider to examine lower ranked images.
- Allowing the user to view the "history" of the search, for instance by showing a graph of the changes in the scores (or ranks) of the images.
- Allowing a precise comparison of the query image and the result images.
- Allowing configuration of the update time.
- Allowing additional query modes, such as selecting a random picture, cropping the query picture, and allowing for query refinement.
- Making adjustments to the interface, such as adding tooltips and a 'New search' button.

Of course, new search algorithms developed through our research efforts will also be integrated with the prototype.

4 Experiments

We performed two experiments to examine the benefits of allowing user interaction with the search. First, we measured the dynamic quality of the results, to quantify the potential benefits of presenting intermediate results. Second, we studied the overhead of the search due to the presentation of intermediate results.

The image collection consisted of 52,273 images taken from video sequences of various television shows. These images were described by a total of 5,017,298 descriptors, which were stored in a random order; as each descriptor needs 100 bytes of storage, the size of the descriptor database on disk was about 500MB. The experiments were run on a Dell workstation with a 1.6GHz Pentium 4 CPU, 512MB of main memory, and an 18GB ATA disk.

We ran queries using 30 randomly chosen images from the collection, with a very varying number of descriptors. As the overall behavior of all 30 images was very similar, we have chosen two of those images to present in detail, while also showing results for all 30 images. These two images are equal in size but the system creates more descriptors for one of them. The *small* image contained 52 descriptors and the *large* image 562 descriptors. They were chosen as fairly extreme candidates in terms of number of query descriptors.

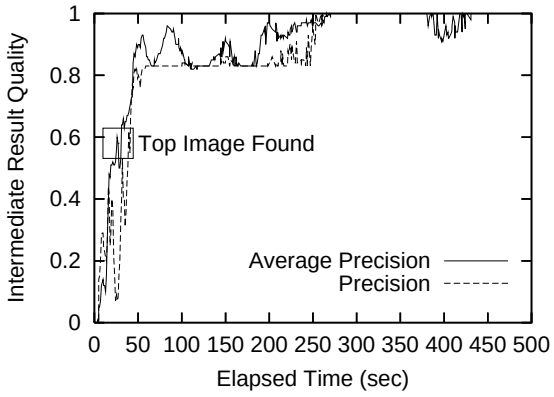

Figure 8: Intermediate result quality for large image

4.1 Quality of Intermediate Results

The dynamic quality of the answers was measured relative to the final outcome of the search algorithm. Two metrics were used. Precision determines how many of the eventual top six images are present in the current top six images at any time, independent of the order of images. Average precision over the top six images, on the other hand, applies more importance to correctly ranking the top image(s):

$$average\ precision = \frac{\sum_{i=1}^{6} precision_i}{6}$$

The precision and average precision were calculated once every second. The search was performed 30 times for each image and the results averaged.

Figure 8 shows the changes in the quality of the intermediate results for the large image, as time elapses during the search. The complete execution of the search takes about 475 seconds (or almost 8 minutes). As expected, the initial results contain only random images and the final result contains the correct six images. The figure shows that the quality of the result rises very rapidly early in the search, and after only about 60 seconds (or roughly 15% of the search time) five out of the top six images are already being presented to the user.†

Figure 8 also shows the time point at which the eventual top image is first shown as the intermediate top image. This occurs after only 27 seconds, or 6% of the search time. This is a promising result, as it indicates that a search process that quickly retrieves 6% of the *correct votes*, will have a good chance of presenting meaningful results to the user. Since

† The "flickering" of the result quality seen in the range from 70 seconds to 250 seconds—and even later—happens because a few images receive a similar number of votes, and thus a single new vote can change their relative position.

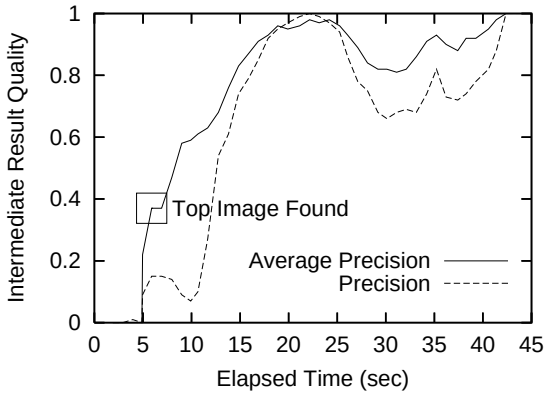

Figure 9: Intermediate result quality for small image

there are, with the large image, $562 \times 30 = 16,860$ votes to be shared, it may be possible to get a good result by reading as few as 1,000 descriptors in the best case!

Figure 9 shows the changes in the quality of the intermediate results for the small image, as time passes during the search. As the figure shows, the response time is much shorter, due to fewer query descriptors. Overall, the observations above still hold, although now the search process must search about 15% and 50% of the collection to find the first image and top five images, respectively. In this case, 15% of $52 \times 30 = 1,560$ votes is about 250 votes, or even fewer than before.

Additionally, more "flickering" of the results is seen in Figure 9. The higher proportions and the flickering here are due to the fact that with fewer query descriptors, individual votes have proportionately more effect on the outcome of the search. For high-quality images with many descriptors, such as the large image, this is not an issue.

Figure 10 outlines the quality of the intermediate results for all 30 query images. Two lines are shown in the figure, the time required to run the query until completion and the time that passes until the top image is first shown in the top position. Figure 10 indicates that the time required to find the top image is generally quite low and relatively independent of the number of descriptors in the image.

4.2 Server Overhead

In our architecture, the search process may be interrupted while the user retrieves the status of the search. Although mutual exclusion is not required, the two processes can interfere with each other. We have therefore made measurements to determine the additional time required to conclude the search due to interference from the client.

Figure 11 shows the response time of the system for the two selected images, as the polling interval is increased (the dashed lines show the response time without polling).

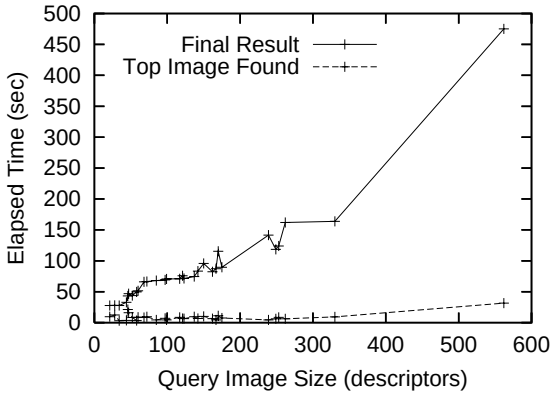

Figure 10: Intermediate result quality for 30 images

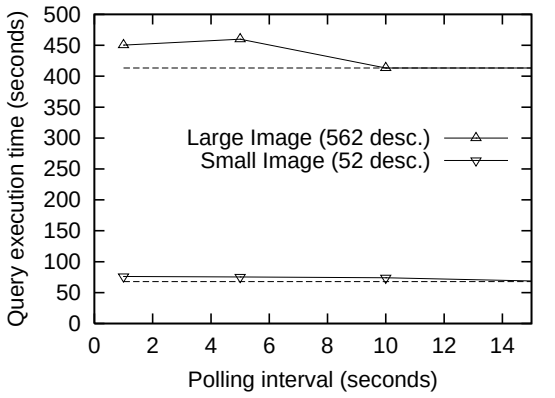

Figure 11: Server overhead

On one extreme, a single second passes from the receipt of one intermediate result to the request for another. On the other extreme, the results are obtained once query processing is completely finished. As no significant changes were seen beyond a polling interval of 10 seconds, the x-axis is truncated. Again, each data point is the average of 30 runs.

As the figure shows, there is some overhead associated with frequent polling, but never more than 12%. Given the large gains in response time, due to presentation of intermediate results, this overhead is clearly acceptable.

5 Related work

Much work has been done on content-based image retrieval (e.g., see [RHC99, VT01]) and on user interaction with image retrieval systems in particular (e.g., see [BCP04, ACP04]). Most of these works, however, have focused on global description schemes, which are known to suffer problems of "disappearance of discriminability" as the image collection grows. While our approach may be applied to such global descriptors, it is the first work we are aware of that addresses user interaction with such an advanced image description scheme.

The work most related to ours from the database area is online aggregation [HHW97]. Online aggregation is a query processing scheme aimed at interactive processing of expensive aggregate queries, such as calculating the average salaries across departments. In online aggregation the dynamic interface quickly lists approximate averages, along with indicators of the quality of the estimates and progress bars. The user can halt processing of the query, once the desired accuracy is reached.

6 Conclusions

This report has described the *Eff*^{2} image retrieval system prototype, which is designed to facilitate user interaction with a search process using a very advanced local description scheme [AG01]. The prototype is based on three main components: a client API, an image server and a client interface. We have demonstrated, through experimental results, that intermediate results can indeed have significant value to the users and that compared to the gains of presenting intermediate results, the overhead is small.

References

[ACP04] M. Albanese, C. Cesarano, and A. Picariello. A multimedia data base browsing system. In *Proceedings of the First International Workshop on Computer Vision meets Databases (CVDB)*, Paris, France, 2004.

[AG01] L. Amsaleg and P. Gros. Content-based retrieval using local descriptors: Problems and issues from a database perspective. *Pattern Analysis and Applications*, 4, 2001.

[BAG03a] S.-A. Berrani, L. Amsaleg, and P. Gros. Approximate searches: k-neighbors + precision. In *International Conference on Information and Knowledge Management (CIKM)*, New Orleans, LA, 2003.

[BAG03b] S.-A. Berrani, L. Amsaleg, and P. Gros. Robust content-based image searches for copyright protection. In *Proceedings of the First ACM International Workshop on Multimedia Databases (MMDB)*, New Orleans, LA, 2003.

[BCP04] I. Bartolini, P. Ciaccia, and M. Patella. The PIBE personalizable image browsing engine. In *Proceedings of the First International Workshop on Computer Vision meets Databases (CVDB)*, Paris, France, 2004.

[EGJA05] S.H. Einarsson, R.Ý. Grétarsdóttir, B.Þ. Jónsson, and L. Amsaleg. The *Eff*^{2} image retrieval system prototype. In *Proceedings of the IASTED Conference on Databases and Applications (DBA)*, Innsbruck, Austria, 2005.

[FBF+94] C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic, and W. Equitz. Efficient and effective querying by image content. *Journal of Intelligent Information Systems*, 3, 1994.

[FRKV94] L.M.J. Florack, B.M. ter Haar Romeny, J.J. Koenderink, and M.A. Viergever. General intensity transformations and differential invariants. *Journal of Mathematical Imaging and Vision*, 4(2), 1994.

[HHW97] J.M. Hellerstein, P.J. Haas, and H.J. Wang. Online aggregation. In *Proceedings of the ACM SIGMOD Conference on Management of Data*, Tuczon, AZ, 1997.

[HKM+97] J. Huang, R. Kumar, M. Mitra, W.-J. Zhu, and R. Zabih. Image indexing using color correlograms. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, San Diego, CA, 1997.

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of International Conference on Computer Vision (ICCV)*, Corfu, Greece, 1999.

[MPE] Moving Picture Experts Group. MPEG home page. http://www.chiariglione.org/mpeg/.

[PSR+01] F.A.P. Petitcolas, M. Steinebach, F. Raynal, J. Dittmann, C. Fontaine, and N. Fatès. A public automated web-based evaluation service for watermarking schemes: Stirmark benchmark. In *Proceedings of electronic imaging, security and watermarking of multimedia contents III*, San Jose, CA, 2001.

[RHC99] Y. Rui, T.S. Huang, and S.-F. Chang. Image retrieval: current techniques, promising directions and open issues. *Journal of Visual Communication and Image Representation*, 10(4), 1999.

[SS94] M.A. Stricker and M.J. Swain. The capacity of color histogram indexing. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, Seattle, CA, 1994.

[VT01] R.C. Veltkamp and M. Tanase. Content-based image retrieval systems: A survey. http://www.aa-lab.cs.uu.nl/cbirsurvey/, 2001.

REYKJAVÍK UNIVERSITY

HÁSKÓLINN Í REYKJAVÍK

Department of Computer Science
Reykjavík University
Ofanleiti 2, IS-103 Reykjavík, Iceland
Tel: +354 599 6200
Fax: +354 599 6201
http://www.ru.is
ISSN 1670-5777