Available online at www.sciencedirect.com
SCIENCE DIRECT®
Journal of Visual Languages and Computing
14 (2003) 363–386
Journal of Visual Languages & Computing
www.elsevier.com/locate/jvlc

ImageGrouper: a group-oriented user interface for content-based image retrieval and digital image arrangement

Munehiro Nakazato[a,*], Ljubomir Manola[b], Thomas S. Huang[a]

[a] *Beckman Institute for Advanced Science and Technology, University of Illinois at Urbana-Champaign, 405 N. Mathews Ave, Urbana, IL 61801, USA*
[b] *Faculty of Electrical Engineering, University of Twente, Netherlands*

Received 30 May 2002; received in revised form 30 April 2003; accepted 1 May 2003

Abstract

In content-based image retrieval (CBIR), experimental (trial-and-error) query with relevance feedback is essential for successful retrieval. Unfortunately, the traditional user interfaces are not suitable for trying different combinations of query examples. This is because first, these systems assume query examples are always added incrementally. Second, the query and the result display are done on the same workspace. Once the user removes an image from the query examples, the image may disappear from the user interface. In addition, it is difficult to combine the result of different queries.

In this paper, we propose a new interface for Content-based image retrieval named *ImageGrouper*. *ImageGrouper* is a Group-Oriented user interface in that all operations are done by creating groups of images. This approach has several advantages. First, the users can interactively compare different combinations of the query examples by dragging and grouping the images on the workspace (*Query-by-Group*). Because the query results are displayed on another pane, the user can quickly review the results and modify the query. Combining different queries is also easy. Furthermore, the concept of "Image Groups" is also applied for annotating and organizing many images. The *Annotation-by-Groups* method relieves the user of tedious task of annotating textual information on a large number of images. This method realizes a hierarchical annotation of the images as well as *Bulk Annotation*. The *Organize-by-Group* method lets the users manipulate the image groups as "Photo Albums" to organize the

*Corresponding author. Tel.: +1-217-244-1089; fax: +1-217-244-8371.
E-mail addresses: nakazato@uiuc.edu (M. Nakazato), l.manola@el.utwente.nl (L. Manola).

1045-926X/03/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S1045-926X(03)00034-X

images. Finally, the usability of the system is compared with the traditional user interfaces. By incorporating the lessons from the experiments, the usability of *ImageGrouper* is further improved.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: Content-based image retrieval; Digital photography; User interface; Information retrieval; Image database

1. Introduction

Many researchers have proposed systems to search images from large image databases. These systems can be divided into two types of interactions: *Browsing* and *Searching*. In the image browsing systems, the users look through the entire collections. In most image browsing systems, the images are clustered in a hierarchical manner and the user can traverse the hierarchy by zooming and panning [1–4]. In [4], browsing and searching are integrated so that the user can switch back and forth between browsing and searching.

Meanwhile, enormous amounts of research have been done for content-based image retrieval (CBIR) [5–7]. In CBIR systems, the user searches images by visual similarity, that is, low-level image features such as color [8], texture [9] and structure [10]. They are automatically extracted from the images and stored in the database. Then, the system computes the similarity between the images based on these features.

The most popular method of CBIR interaction is *Query-by-Examples*. In this method, the users select example images (as relevant or irrelevant) and ask the system to retrieve visually similar images. In addition, in order to improve the search further, CBIR systems often employ *Relevance Feedback* [6,11,12], in which the users can refine the search incrementally by giving feedback to the result of the previous query.

In this paper, we propose a new user interface for digital image retrieval and arrangement, named *ImageGrouper*. *ImageGrouper* is a *Group-Oriented User Interface*, in that all operations are done by creating groups of the images on the workspace. First, a new concept the *Query-by-Groups* is introduced for the CBIR. The users construct queries by making groups of the relevant and irrelevant images. The groups are easily created by dragging images on the workspace. Because the image groups can be easily reorganized, a flexible relevance feedback is achieved. Moreover, with the similar operations, the user can effectively annotate and organize a large number of images.

In the next section, we discuss how the groups are used for image retrieval. Then, the following two sections describe the use of the image groups for image annotation and organization. Then, the usability of the new user interface is compared with the traditional user interfaces in Section 5. Section 6 discusses in detail the implementation of the system. The future work and the conclusion are presented in Sections 7 and 8.

2. User interface support for content-based image retrieval

2.1. Current approaches: incremental search

Not many researches have been done regarding User Interface support for CBIR systems [4,13]. Fig. 1(a) shows a typical graphical user interface (GUI) for CBIR system that supports the *Query-by-Examples*. Here, a number of images are aligned in grids. In the beginning, the system displays randomly selected images. The effective ways to align images are studied in [14]. On some systems, the initial images are found by browsing or keyword-based search.

Under each image, a slide bar is attached so that the user can tell the system which images are relevant (or irrelevant). If the user thinks an image is relevant, s/he moves the slider to the right. If s/he believes the image is not relevant and should be avoided, s/he moves the slider to the left. The amount of slider movement represents the degree of relevance (or irrelevance.) In some systems, the user selects example images by clicking the images or the check boxes under the images (Fig. 1(b) [15]). In these cases, the degrees of relevance are not specified.

When the "Query" button is pressed, the system computes the similarity between the selected images and the database images, then retrieves the N most similar images. The images on the user interface are replaced with the newly retrieved images. These images are ranked based on the similarity.

If the user finds additional relevant images in the result, s/he selects them as new query examples. If a highly irrelevant image appears in the result set, the user can

(a) Slider-Based GUI (b) Click-Based GUI

Fig. 1. Typical GUI for CBIR Systems: Images are tiled in the workspace. In the Slider-based GUI, a slider is attached under each image so that the user can specify his/her interest.

select it as a negative example. Then, the user press "Query" again to obtain the refined search results. The user can repeat this process until s/he is satisfied or there is no additional image to add. This process is called *Relevance Feedback* [6,12]. Moreover, in many systems, the users are allowed to directly weight the importance of the image features such as color and texture.

In [16], Smeulders et al. classified *Query by Image Example* and *Query by Group Example* into two different categories. From the user interface viewpoint, however, these two are very similar. The only difference is whether the user is allowed to select multiple images or not. In this paper, we classify both approaches as the Query-by-Examples method. In stead, we use term "Query by Groups" to refer our new model of query specification method described later.

2.1.1. Disadvantages of the traditional approaches

The Query-by-Example approach has several drawbacks. First of all, these systems assumed that *The* More Query Examples are Available, the Better Result We Can Get. Therefore, the users were supposed to refine the search *incrementally* by adding new example images from the result of the previous query. However, this assumption is not always true. Additional examples may contain undesired features and degenerate the retrieval performance.

Fig. 2 shows an example of situations where more query examples could lead to worse results. In this example, the user is trying to retrieve pictures of cars. The upper row shows the query result when only one image of "car" is used as a query example. The bottom row shows the result of two query examples. The results are

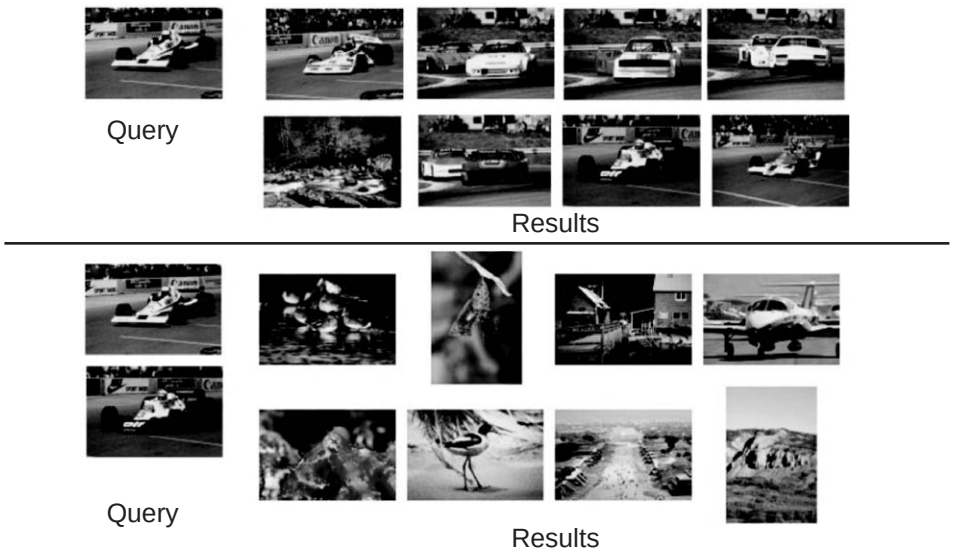


Fig. 2. Example of "More is not necessarily better". The upper row is the case of one example, the lower row is the case of two examples.

ordered based on the similarity ranks. In both cases, the same relevance feedback algorithm (Section 6.2. and [12]) was used and tested on Corel image set of 17,000 images. In this example, even if this additional example image looks visually relevant for human eyes, it introduces undesirable features into the query. Thus, no car image appears in the top 8 images. An image of car appears in the rank 13th for the first time.

This example is not a special case. It happens often in actual image retrieval scenarios and confuses the users. This problem happens because of *Semantic Gap* [13,16] between the high-level concept in the user's mind and the extracted low-level image features. Furthermore, finding good combinations of query examples is very difficult because the image features are numerical values that are impossible to be estimated by human. The only way to find the right combination is *trial-and-error*. Otherwise, the user can be trapped in a small part of image database [4].

Unfortunately, the traditional user interfaces were designed for "Incremental Search" and are not suitable for "Experimental (trial-and-error) Query." This is because in these systems, both the query specification and the result display use the same workspace. Thus, the images on the grid are replaced by the query result. Once the user removes an image from the query examples during relevance feedback loops, the image may disappear from the user interface for good unless the image is retrieved in the search again. Thus, it is awkward to bring it back later for another query.

Second, the traditional interface does not allow the user to put aside the query results for later uses. This type of interaction is desired because the users are not necessarily looking for only one type of images. The users' interest may change during retrieval. This behavior is known as *berry picking* [17] and has been observed for text documents retrieval by O'Day and Jeffries [18].

Moreover, because of *Semantic Gap* [13,16] mentioned above, the users often need to make more than one query to satisfy his/her need [17]. For instance, a user may be looking for images of "beautiful flowers." The database may contain many different "flower" images. These images might be completely different in terms of the low-level visual features. Thus, the user needs to retrieve "beautiful flowers" as a collection of different types of images.

Finally, sometimes, the user had better start from a generic concept of the objects and narrow down to more specific ones. For example, suppose the user is looking for images of "red cars." Because image retrieval systems use various image features [9,10] as well as the colors [8], even cars with different colors may have many common features with "red cars." In this case, it is better to start by collecting images of "cars of any color." Once enough number of car images are collected, the user can specify "red cars" as positive examples, and other cars as negative examples. Current interfaces for CBIR systems, however, do not support these types of query behavior.

Another interesting approach for the Query-by-Example has been proposed by Santini et al. [13]. In their *El Ninõ* system, the user specifies a query by the mutual distance between the example images. The user drags images on the workspace so that more similar images (in the user's mind) are located closer to each other. The

system then reorganizes the images' locations reflecting the user's intent. There are two drawbacks in El Ninõ system. First, it is unknown to the users how close similar images should be located and how much the negative examples should be moved at a distance from the good examples. It may take a while for the user to learn "the metric system" used in this interface.

The second problem of the El Ninõ system is that like traditional interfaces, the query specification and the result display are done on the same workspace. Thus, the user's previous decision (in the form of the mutual distance between the images) is overridden by the system when it displays the results. This makes *trial-and-error* query difficult. Given the analogue nature of this interface, *trial-and-error* support might be essential. Even if the user gets an unsatisfactory result, there is no way to redo the query with a slightly different configuration. Any experimental result is not provided in [13].

2.2. Query-by-groups: the ImageGrouper approach

We are developing a new user interface for CBIR systems named *ImageGrouper*. In this system, a new concept *Query-by-Groups* was introduced. The *Query-by-Groups* mode is an extension to the Query-by-Example mode described above. The major difference is that while the Query-by-Example handles the images individually, a "group of images" is considered as the basic unit of the query in the *Query-by-Group*.

Fig. 3 shows the display layout of *ImageGrouper*. The interface is divided into two panes. The left pane is the *ResultView* that displays the results of content-based retrieval, keyword-based retrieval, and random retrieval. This is similar to the traditional GUI except for there are no sliders or buttons under the images. The right pane is the *GroupPalette*, where the user specifies the image query information by creating image groups.

Fig. 4 shows the sequence of the typical image retrieval with the Query-by-Group. In order to create an image group, the user first drags one or more images from the *ResultView* into *GroupPalette* (Fig. 4(a)) Then, the user encloses the images by drawing a rectangle (box) as we draw a rectangle in drawing applications (Fig. 4(b)). All the images within the *group box* become the member of this group. Any number of groups can be created in the palette. The user can move the images from one group to another at any moment. In addition, the groups can be overlapped to each other (i.e. each image can belong to multiple groups.) To remove an image from a group, the user simply drags it out of the box. When the right mouse button is pressed over a group box, a popup menu appears so that the user can select query properties (positive, negative, or neutral) of the group (Fig. 4(b)). The properties of the groups can be changed at any moment. The colors of the corresponding boxes change accordingly. To retrieve images based on these groups, the user press the "Query" button placed at the top of the window (Fig. 3). Then, the system retrieves new images that are similar to the images in the positive groups while avoiding those images that are similar to the negative groups. The search results are displayed in the *ResultView* (Fig. 4(c)). If the user find new relevant images in the result, s/he can

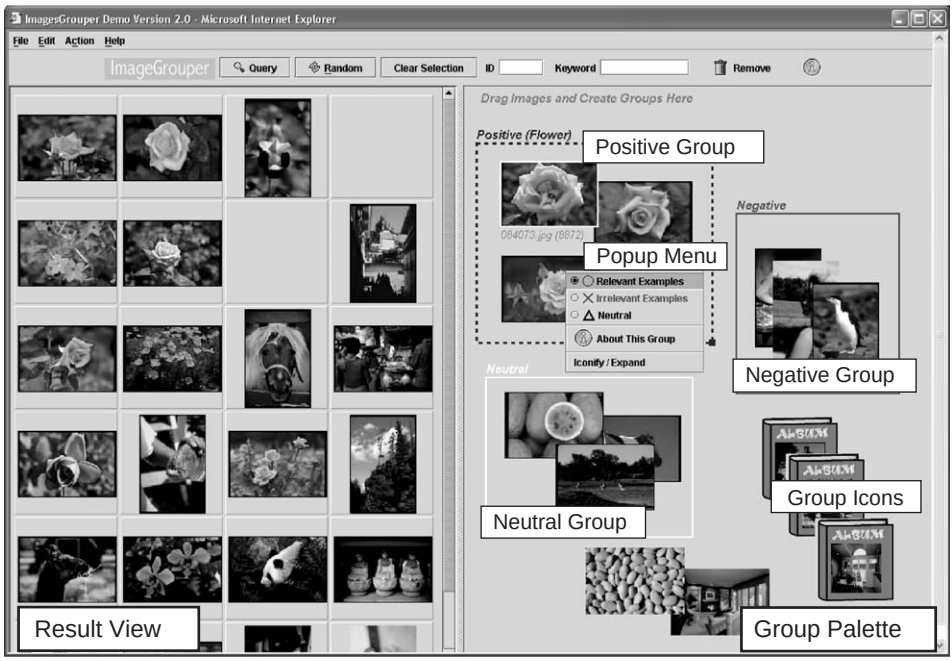
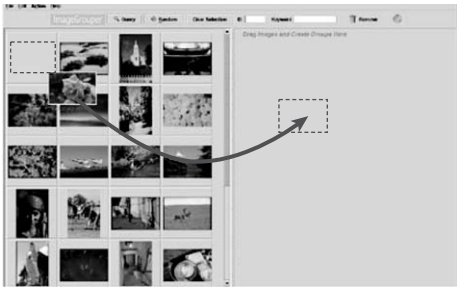


Fig. 3. The *ImageGrouper*: The users create an image groups by dragging images from the left pane to the right pane, then enclose them by enclosing them with a rectangle.

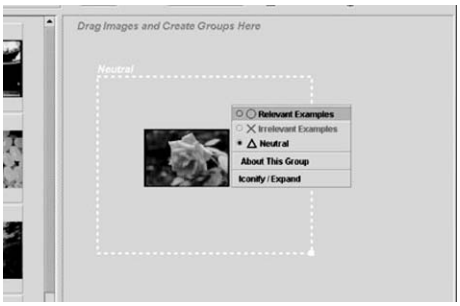
refine the search by dragging these images to the Palette, then press "Query" again (Fig. 4(d)). S/he can repeat until s/ he is satisfied or no additional relevant images can be found in the *ResultView*.

When a group is specified as *Neutral* (displayed as a white box), this group does not contribute to the search at the moment. This group can be turned to a positive or negative group for another search later. If a group is *Positive* (displayed as a blue box), the system uses common features among the images in the group. Meanwhile, if a group is given *Negative* (red box) property, the common features in the group are used as a negative feedback. The user can specify multiple groups as positive or negative. In this case, these groups are merged into one group (i.e. the union of the groups is taken.) The detail of the algorithm is described in Section 6.2.

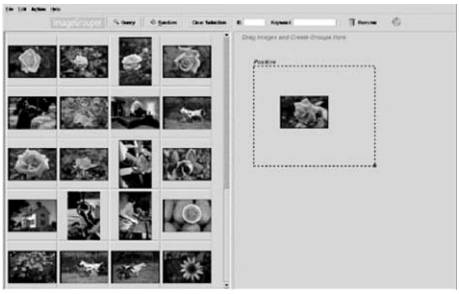
While the user created only one group in Fig. 4, the user can create multiple groups on the workspace. Fig. 3 shows an example of three groups. As in Fig. 4, the user is retrieving images of "flowers." On the *GroupPalette*, three flower images are grouped as a positive group. On the right of this group, a red box is representing a negative group that contains several "non-flower" images. Below the "flowers" group, there is a neutral group (white box), which is not used for retrieval now. The images can be moved outside of any groups in order to temporarily remove the images from the groups.



(a) Drag a Flower image to the palette.



(b) Draw a rectangle around the image, then chose "Relevant" from the popup menu.



(c) The result after the first query.

(d) More relevant images are being dragged into the group.

Fig. 4. The sequence of the basic query operation on *ImageGrouper*.

The gestural operations of *ImageGrouper* are similar to file operations of a Window-based OS. Furthermore, because the user's mission is to collect images, the operation "*Dragging Images into a Box*" naturally matches the user's cognitive state.

2.3. Flexible image retrieval

The main advantage of the Query-by-Groups is flexibility.

2.3.1. Trial and error query by mouse dragging

In *ImageGrouper*, the images can be easily moved between the groups by mouse drags. In addition, the neutral groups and space outside of any groups in the palette can be used as *storage area* [19] for the images that are not used at the moment. They can be reused later for another query. This makes *Trial and Error* of the relevance feedbacks easier. The user can quickly explore different combinations of query examples by dragging the images into or out of the box. Moreover, the query specification that the user made is preserved and visible in the palette. Thus, it is easy to modify the previous decision when the query result is not satisfactory.

2.3.2. Groups in a group

ImageGrouper allows the users to create a new group within a group (*Groups in a Group*). With this method, the user begins with collecting relatively generic images first, then narrows down to more specific images. Fig. 6 shows an example of *Groups in a Group*. Here, the user is looking for "Red Cars." When s/he does not have enough number of examples, however, the best way to start is to retrieve images of "cars with any color." This is because these images may have many common features with the red car images, though their colors features are different. The large white box is a group for "Cars with any colors." Once the user found enough number of car images, s/he can narrow down the search only for red cars. In order to narrow down the search, the user divides the collected images into two sub-groups by creating two new boxes for the red cars and the other cars. Then the user specifies the red car group as positive and the other cars group as negative (or neutral,) respectively. In Fig. 6, the left smaller (blue, i.e. positive) box is the group of red cars, and the right box (red, i.e. negative) is the group of the non-red cars. This *narrow down search* was not possible on the conventional CBIR systems.

2.4. Experiment on trial-and-error query

In order to examine the effect of *ImageGrouper's* experimental query support, we compared the query performance of our system with that of a traditional incremental approach (Fig. 1(a)). In this experiment, we used Corel photo stock that contains 17 000 images as the data set. For both interfaces, the same image features and relevance feedback algorithms (described in Section 6.2) are used.

For the traditional interface, the top 30 images were displayed and examined by the user in each relevance feedback. For *ImageGrouper*, the top 20 images were displayed in the *ResultView*. Only one positive group and one neutral group were created for this test. On both interfaces, no negative feedback was used. Feedback loops are repeated up to 10 rounds or until convergence.

We tested over eight classes of images as shown in Table 1. For each class, a query started from one image. In the case of the traditional interface, the query is repeated

Table 1
The number of feedback loops until convergence. 10 means that it did not converge until 10th round

Object	Traditional	Grouper
Red car	4	10
Tiger	3	10
Bird	4	8
Yellow flower	3	10
Citrus	2	4
Polar bear	5	10
Elephant	3	10
Swimmer	2	10

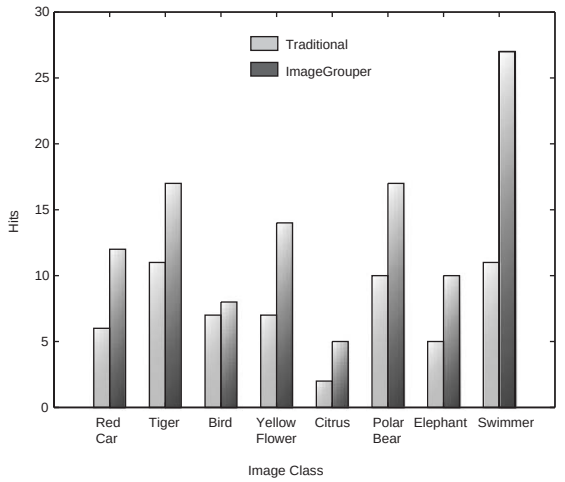


Fig. 5. The comparison on the number of hits until the 10th round or convergence.

by giving additional example from the result of previous query. When no more good example appears, the search stops (convergence). Meanwhile, for *ImageGrouper*, the search is refined incrementally at first. When the incremental search converges, the *trial-and-error* search is applied by moving some images out of the positive group into a neutral group (This means that the number of positive examples was temporarily decreased). Then, the search was refined incrementally again until another convergence occurred. The user repeats this until *trial-and-error* query has no effect.

Fig. 5 shows the number of correctly retrieved images after the convergence (or the 10th rounds.) This value is proportional to the recall rate. Thus, larger value means better retrieval performance. Table 1 shows the number of relevance feedback loops until convergence. The value 10 means the query did not converge before the 10th round.

Clearly, *ImageGrouper* can achieve better retrieval (higher recall) even if the underlying technologies (relevance feedback algorithm and visual features) are identical. In addition, the search with *ImageGrouper* is less likely to converge prematurely even when the search with the traditional interface converges into a small number of images after a few iterations. This result suggests the importance of support for *Trial-and- Error* query.

Meanwhile, the *Query-by-Group* can introduce a new problem. In an experimental query, the users need to decide which images should be included in the current query set. The search may be improved by adding or removing the images. This is not always easy to predict. Although *ImageGrouper* enables fast and easy *trial-and-error*, the user may have to try many different combinations until s/he obtains a better result. Unfortunately, it is difficult to automate this process because the judgement of the query results is subjective and depends on the user. However, it is still

desirable that the system suggests some promising query combinations. For example, the system can find an outlier in the current set of query examples.

3. Text annotations on images

The keyword-based search is a very powerful method for searching images. However, it works well only when all the images in the database are annotated with textual information. For commercial photo stocks, it may be feasible to enter keywords to every image manually. For home users, however, it is too tedious.

When the keyword search is integrated with CBIR like our system or [4], keyword-based search can be used to find the initial query examples for the content-based search. For this scheme, the user does not have to annotate all images. In any cases, it is very important to provide easy and quick ways to annotate text on many images.

3.1. Current approaches for text annotation

The most primitive way for text annotation is to select an individual image, then type in keywords. Because this interaction requires the user to use the mouse and keyboard repeatedly in turn, it is very frustrating for a large image database.

Several researchers have proposed smarter user interfaces for keyword annotation on images. In *Bulk Annotation* method of *FotoFile* [20], the user selects multiple images on the display, selects several attribute/value pairs from a menu, and then presses the "Annotate" button. Therefore, the user can add the same set of keywords on many images simultaneously. To retrieve images, the user selects entries from the menu, and then presses the "Search" button. Because of *visual and gestural symmetry* [20], the user needs to learn only one tool for both annotation and retrieval.

PhotoFinder [21] introduced *Drag-and-Drop* method, where the user selects a label from a scrolling list, then drags it directly onto an image. Because the labels remain visible at the designated location on the images and these locations are stored in the database, these labels can be used as "captions" as well as for the keyword-based search. For example, the user can annotate the name of a person directly on his/her portrait in the image, so that other users can associate the person with his/her name. When the user needs new words to annotate, s/he adds them to the scrolling list. Because the user drags keywords into individual images, bulk annotation is not supported in this system.

3.2. Annotation by groups

Most home users do not want to annotate their images one by one, especially when the number of images is large. Often, the same set of keywords is enough for several images. For example, a user may just want to annotate "My Roman Holiday, 1997" on all images taken in Rome. Annotating the same keywords repeatedly is painful enough to discourage him/her from using the system.

ImageGrouper introduces *Annotation-by-Groups* method where the keywords are annotated not on individual images, but on groups. As in the *Query-by-Groups*, the user first creates a group of images by dragging the images from the *ResultView* into the *GroupPalette* and drawing a rectangle around them. In order to give keywords to the group, the user opens *Group Information Window* by selecting "About This Group" from the pop-up menu (Fig. 3). In this window, arbitrary number of words can be added. Because the users can simultaneously annotate the same keywords on many images, the annotation becomes much faster and less error prone. Although the Annotation-by-Groups is similar to bulk annotation of *FotoFile* [20], there are several advantages described below.

3.2.1. Annotating new images with the same keywords

In the *Bulk Annotation* [20], once the user finished annotating keywords to some images, there is no fast way to give the same annotation to another image later. The user has to repeat the steps (i.e. select images, select keywords from the list, then press the "Annotate" button.) This is awkward when the user has to add many keywords. Meanwhile, in *Annotation-by-Group*, the system attaches annotations not on each image, but on the groups. Therefore, by dragging new images into an existing group, the same keywords are automatically given to it. The user does not have to type the same words again.

3.2.2. Hierarchical annotation with groups in a group

In *ImageGrouper*, the user can annotate the images hierarchically using *Groups in a Group* method described above (Fig. 6). For example, the user may want to add a new keyword "Trevi Fountain" to only a part of the image group that has been labeled "My Roman Holiday, 97." This is easily done by creating a new sub-group within the group and annotating only on the sub-group.

In order to annotate hierarchically on *FotoFile* [20] with bulk annotation, the user has to select a subset of images that are already annotated, and then annotate them again with more keywords. On the other hand, *ImageGrouper* allows the user to visually construct a hierarchy of the images on the *GroupPalette* first, then edit the keywords on the *Group Information Window*. This method is more intuitive and less error prone.

3.2.3. Overlap between images

An image often contains multiple objects or people. In such cases, the image can be referred in more than one context. *ImageGrouper* support this multiple reference by allowing overlaps between the image groups, i.e. an image can belong to multiple groups at the same time. For example, in Fig. 7, there are two image groups: "Cloud" and "Mountains." Because some images contain both cloud and mountain, these images belong to both groups (in overlapped region.) Once these two groups are annotated with "Cloud" and "Mountain" respectively, the images in the overlapped region are automatically referred as "Cloud and Mountain." This concept is not supported in other systems.

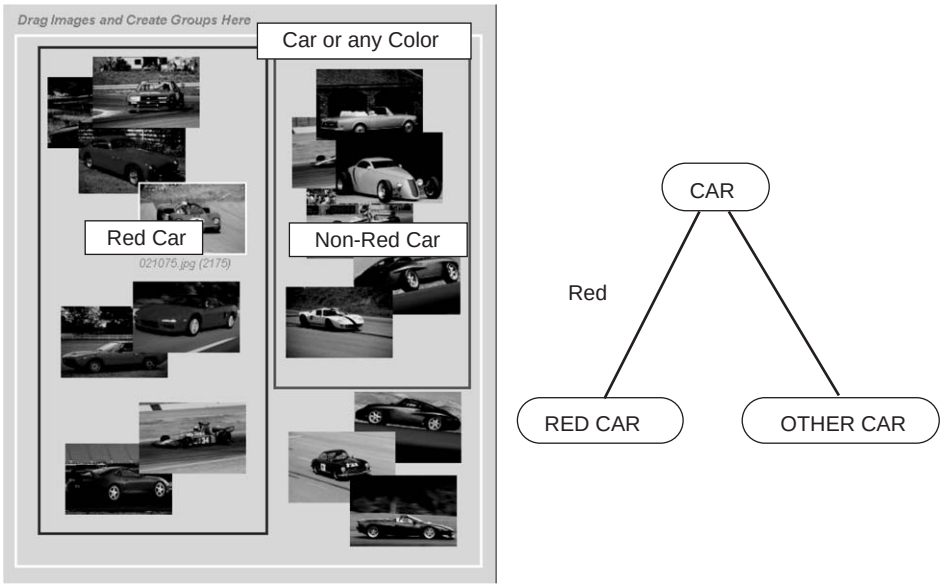


Fig. 6. *Groups in a group*. In *ImageGrouper*, the user can create a hierarchy of image groups. In this example, the entire group is "cars." Once the whole group is annotated as "car," the user only need to type "red" to annotate the red car group (smaller rectangle in the left).

4. Organizing images by groups

In the previous two sections, we described how *ImageGrouper* supports the content-based query as well as the keyword annotation. These features are closely related and complementary to each other. In order to annotate images, the user can collect visually similar images first, using the content-based retrieval with the *Query-by-Groups*. Then s/he can annotate textual information to the group of collected images. After this point, the user can quickly retrieve the same images using keyword-based search.

Conversely, the results of the keyword-based search can be used as a starting point for content-based search. This method is useful especially when the image database is only partially annotated.

4.1. Photo albums and group icons

As described above, *ImageGrouper* allows the groups to be overlapped. In addition, the user can attach textual information on these groups. Therefore, the groups in *ImageGrouper* can be used to organize the pictures as "photo albums [20]" Similar concepts are proposed in *FotoFile* [20] and Ricoh's *Storytelling* system [22]. In both systems, albums are used for "slide shows" to tell stories to the other users.

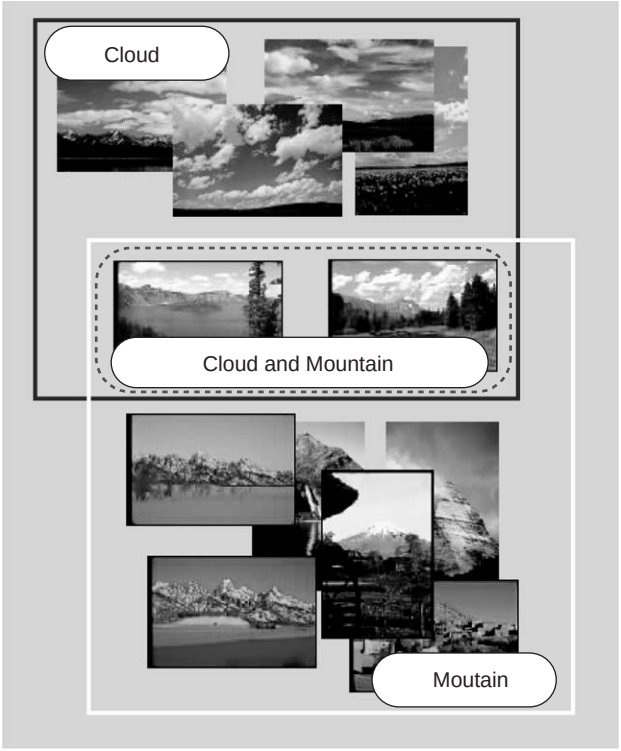


Fig. 7. Overlap between groups. Two images in the overlapped region contain both mountain and cloud. The user only needs to annotate two groups. Images in the overlapped region are automatically annotated with both keywords.

In *ImageGrouper*, the user can convert an image group into a *Group Icon* (Fig. 3). When the user selects "Iconify" from the popup menu (Fig. 3) images in the group disappear and a new icon for the group appears in the *GroupPalette*. When the group has an overlap with another group, images in the overlapped region remain in the display.

Furthermore, the users can manipulate those group icons as they handle individual images. They can drag the group icons anywhere in the palette. The icons can be even moved into another group rectangle realizing groups *in a group*.

Finally, the group icons themselves can be used as examples for the content-based query. A group icon can be used as an independent query example or combined with other images and groups. In order to use a group icon as a normal query group, the user right clicks the icon and opens a popup menu. Then, s/he can select "relevant," "irrelevant" or "neutral." On the other hand, in order to combine a group icon with other example images, the user simply draws a new rectangle and drags them into it.

Organize-by-Groups method described here is partially inspired by the Digital Library Integrated Task *Environment* (DLITE) [23]. In DLITE, each text document

as well as the search results is visually represented by icons. The user can directly manipulate those documents in a workcenter (*direct-manipulation*). In [19], Jones proposed another graphical tool for query specification, named *VQuery*. In VQuery, the user specifies the query by creating Venn diagrams. The number of matched documents is displayed in the center of each circle.

While DLITE and VQuery are designed for text document retrieval systems [24], the idea of *direct-manipulation* [23] is applicable more naturally to image databases. In the text document database, it is difficult to determine the contents of text documents from the icons. Therefore, the user has to open another window to examine the detail [23] (in DLITE, a web browser is opened). On the other hand, in the image databases, the images themselves (or their thumbnails) are the objects that the user operates on. Therefore, instant judgment by the user is possible on a single workspace [4,16].

5. Usability study

Even if *ImageGrouper* provides a powerful search ability, the users cannot take advantages of the system unless the system is easy to use. In order to compare the usability of *ImageGrouper* with the traditional graphical user interfaces (GUIs), we conducted usability tests. *ImageGrouper* was compared with two traditional GUIs: a simple *Click-based* Interface (Fig. 1(b)) and a *Slider-based* Interface (Fig. 1(a)). An example of the Click-based Interface is QBIC [5] and an example of the *Slider-based* Interface is the original MARS [6]. Each GUI employs different relevance feedback method and has different expressiveness. For example, the Click-based GUI is very simple and easy to use, but only positive feedback was allowed. The Slider-based GUI allows the user to specify the degree of relevance (from -1.0 to 1.0.) Both the Click-based and the Slider-based lose the previous query information when the search results are returned. Meanwhile, *ImageGrouper* requires drag-and-drop operations, but it realizes a more flexible query with *trial-and-error* query. Therefore, it is very difficult to compare the usabilities of these systems in the actual image retrieval scenarios. Instead, we compared the task completion time and the error rate of the image selection tasks in a simplified scenario as described below.

5.1. Experiment settings

5.1.1. Subjects

Ten people volunteered to participate in this experiment. All subjects were familiar with the commonly used widgets that require mouse operations such as sliders and check boxes. The ages ranged from 20s to 30s. One was a female and nine were male. Most subjects did not have experiences in any content-based image retrieval systems.

5.1.2. Apparatus

Both training and tests are conducted with a PC with Dual Intel Xeon 2 GHz Processors with 1 GB of Main Memory running Windows XP. The PC was located

in a small quiet room. The video card was nVIDIA. Quadro2 with 32 MB Video RAM. For the display, a 17 in LCD monitor set at 1280 by 960 in 32-bit color was used. All user operations were done with a Microsoft Optical USB Mouse.

Each system was implemented in Java2 (version 1.3.x) with Swing API. *ImageGrouper* requires slightly more graphics computation than the other two because the images have to be dragged over the workspace. Above specs, however, are more than enough to run each GUI smoothly, eliminating the differences in the computational overhead. Note that *ImageGrouper* does not require such high-spec machines. It runs smoothly on average PCs or workstations.

5.2. Scenarios

5.2.1. Experimental task

The task of this experiment is to find and select relevant images on each GUI. In each run, when a subject presses the "start" button, a sample image and its description (in text) are displayed on a separate window located to the left of the main GUI. At the same time, 16 different images are tiled on the main GUI. Then, they are asked to find and select all images that are "semantically similar" to the sample images. The correct images do not have to be visually similar to the sample image. For example, if a sample image is a picture of a standing penguin and the description is "Penguin," the subject has to choose all pictures of penguins even if the pictures look different from the sample image (for example a picture of two penguins lying on ice).

Each user interface requires different operations to select the images. In the case of the Click-based system, "Select" means simply clicking the pictures on the GUI. In the case of the Slider-based Interface, the user selects an image by moving the slider to the right. For *ImageGrouper*, the images are selected by dragging images from the *ResultView* to the *GroupPalette*. In any cases, the subjects can examine all images without scrolling. When the subject believes every relevant image is selected, s/he presses the "Next" button. Then the system displays the next problem (a new sample images and another set of 16 images on the GUI.) For both training and experiments, the task was repeated 10 times with different sample images such as "elephant," "flowers," "airplane," "cars," and "fireworks." The number of the correct images was different from one problem to another. The size of each image was fixed to 120 by 80 pixels (or 80 by 120).

Each subject was tested with all three GUIs one by one. The order of the GUIs was randomly chosen for each subject. For each interface, the subjects were first trained with the sample problems, then they were tested with the same interface immediately after the training.

5.2.2. Training

Before the training and testing of each GUI began, each subject individually received a brief instruction about the concept of the GUI (how to select and deselect images) and the procedure of the experiment. Then, s/he was asked to conduct the image selecting operation on the GUI with 10 training problems under the trainer's

supervision. Because the purpose of this training is to make him/her familiar with the GUI, the subject was allowed to ask the trainer assistance during the session.

5.2.3. Experiment

After the subject became familiar enough with the user interface, the subject began the same procedure with new problem sets. The number of images to be selected was not known to the subject and was different for each problem. Unlike the training phase, the subject was encouraged to conduct the operations as fast as possible and as accurate as possible. No question was allowed during the sessions. The completion time and the number of errors (the number of images missed and the number of image incorrectly selected) were recorded. The subjects were interviewed after the experiment.

5.3. The results

5.3.1. Error rate

To see the effects of the user interfaces on the accuracy of the tasks, we compared the number of missed images and the number of incorrectly selected images during the image selection tasks. Table 2 shows the total of 10 problems. The values are averaged over ten subjects. With each interface, the subjects rarely selected wrong images (at most once in ten problems.) The differences among the user interfaces was not significant ($F(2, 27) = 0.144, p = 0.254$).

Meanwhile, the subjects failed to select the correct images up to four times in ten problems (the average was less than 1.5 times per 10 runs in any interface). In each problem, most subjects missed at most one image. The differences in the total number of missed images were not statistically significant among the three interfaces ($F(2, 27) = 0.0859, p = 0.918$).

During the experiments, we informed the subjects that the tasks were timed and encouraged them to finish the tasks as fast as possible. Therefore, most subjects focused on the speed over the accuracy. The subjects, however, achieved fairly accurate image selection with these user interfaces.

5.3.2. Task completion time

The task completion time was measured by the duration from the time the subject pressed the start button to the moment the user pressed the finish button after s/he

Table 2
The number of missed and incorrectly selected images in ten problems

	Missed	Incorrectly selected	Total
Click	1.1	0.2	1.3
Slider	1.3	0.3	1.6
Grouper	1.1	0.5	1.6

The values are averaged over ten subjects. There is no statistically significant differences in the accuracy.

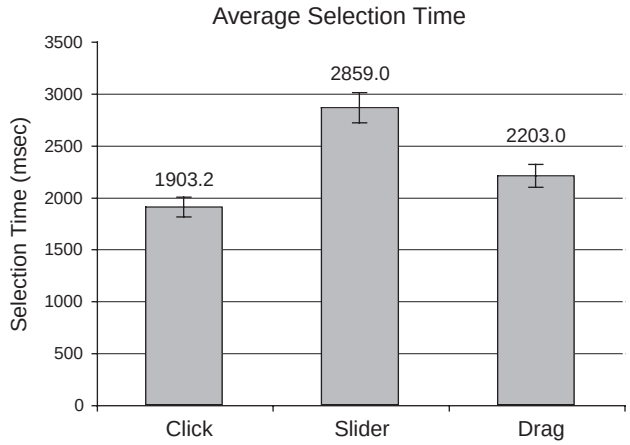


Fig. 8. The average selection time per image. The differences are statistically significant at $p<0.001$.

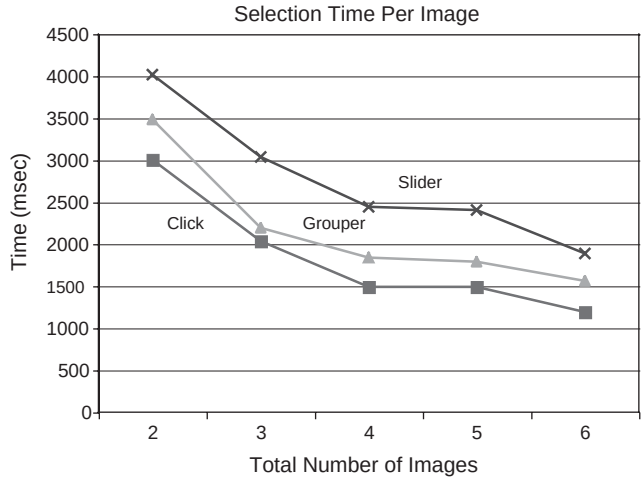


Fig. 9. The average time required to select each images in different problems. The X-axis is the total number of images selected during in the problems. The time per image decreases as the total number of images to be selected increases. The correct numbers to select were not known to the subjects.

completed the ten problems. The number of images to be selected varies from 2 to 6, depending on the problem. Fig. 8 shows the average time required to select each single image. The differences in completion time were significantly different among the user interfaces ($F(2, 27) = 21.05, p<0.001$). The Slider-based GUI was the slowest. Its average selection time was significantly slower than that of *ImageGrouper*

($F(1, 18) = 17.32(p < 0.001)$). The Click-based user interface achieved the shortest task completion time and significantly faster than *ImageGrouper* ($F(1, 18) = 4.455(p < 0.05)$). This result is not surprising since *ImageGrouper* requires *dragging* operations in addition to the mouse clicks on the images. The difference between the Click-based and Grouper was smaller than the difference between Grouper and the Slider-based GUI.

The number of images to be selected was different among the problems. The number ranged from two to six. Fig. 9 shows how the time required to select one image differs with the total number of images to be selected. Before the experiments, we expected the *Time Per Image* in *ImageGrouper* decreases significantly as the total number increases. This is because on *ImageGrouper*, as the user drags the images to the workspace, the number of images on the grid (ResultView) decreases, making it easier to examine the remaining images. Interestingly, the similar effects also occurred on the traditional user interfaces.

In the interview, the several subjects claimed they felt frustrated with the Slider-based GUI. In order to move the mouse pointer to the small slider handles and move them, the users need to focus their concentration on the handles instead of the tasks. Therefore, the Slider-based GUI is not suitable for repetitive image retrieval tasks with the relevance feedback.

5.4. Improving ImageGrouper based on the lesson we learned

As shown in the previous section, the Click-based GUI is the simplest and achieves the shortest task completion time for the simplified tasks. However, the Click-based interface is limited to positive-only relevance feedbacks, where the user can select only relevant images and cannot select irrelevant images as the negative feedback. In addition, as discussed in Section 2.1 the Click-based GUI is not suitable for the *trial-and-error* relevance feedback since it uses the same workspace both for the query creation and the result display. In order to incorporate the advantages of the Click-based GUI into *ImageGrouper*, we modified *ImageGrouper* so that it allows the user to select and moves the images by double clicking. When the user double clicks an image on the ResultView, the image is moved to the selected group on the GroupPalette. If no group is selected, the image is moved to the positive group that is created first. If there is no group on the GroupPalette, the system creates a new group and place the image in the new group. This is effective especially when many images have to be dragged to the GroupPalette. Of course, the user can also move the images by drag and drop.

6. Implementation

A prototype of *ImageGrouper* is implemented as a client-server system, which consists of *User Interface Clients* and *Query Server*. They are communicating via hyper-text transfer protocol (HTTP).

6.1. The user interface client

The user interface client of *ImageGrouper* is implemented as a Java2 Applet with Swing API (Fig. 3). Thus, the users can use the system through Web browsers on various platforms such as Windows, Linux, Unix and Mac OS X.

The client interacts with the user and determines his/her interests from the group information or keywords input. When the "Query" button is pressed, it sends the information to the server. Then, it receives the result from the server and displays it on the *ResultView*. Because the client is implemented in a multi-thread manner, it remains reactive while it is downloading the images. Thus, the user can drag a new image into the palette as soon as it appears in the ResultView.

Note that the user interface of *ImageGrouper* is independent of the relevance feedback algorithms [6,12] and the extracted image features (described below.) Thus, as long as the communication protocols are compatible, the user interface clients can access to any image database servers with various algorithms and image features. Although the retrieval performance depends on the underlying algorithms and the image features used, the usability of *ImageGrouper* is not affected by those factors.

6.2. The query server

The *Query Server* stores the image files and their low-level visual features. These visual features are extracted and indexed off line. When the server receives a request from a client, it computes the weights of the features and compares the user-selected images with the images in the database. Then, the server sends back the IDs of the k most similar images.

The server is implemented as a Java Servlet that runs on the Apache HTTP Server and Jakarta Tomcat Servlet container. It is written in Java and C++. In addition, the server is implemented as a *stateless server*: that is, the server does not hold any information about the clients. This design allows different types of clients such as the traditional user interfaces [25] (Fig. 1) and 3D Virtual Reality interfaces [26–28] can access to the same server simultaneously.

For the home users who want to organize and retrieve the images locally on their PCs' hard disks, *ImageGrouper* can be configured as a standalone application, in which the user interface and the query server reside on the same machine and communicate directly without a Web server.

6.2.1. Image features

As the visual features for content-based image retrieval, we use three types of features: *Color*, *Texture*, and *Edge Structure*. For the color features, HSV color space is used. We extract the first three moments from each of HSV channels [8]. Therefore, the total number of color features is nine. For the texture, each image is applied into *wavelet filter bank* [9] where the images are decomposed into 10 de-correlated sub-bands. For each sub-band, the standard deviation of the wavelet coefficients is extracted. Therefore, the total number of this feature is 10. For the edge structures, we used *Water-Fill edge detector* [10] to extract the image structures.

We first pass the original images through the edge detector to generate their corresponding edge maps. Then, 18 elements are extracted from the edge maps.

6.2.2. Relevance feedback algorithm

The similarity ranking is computed as follows. First, the system computes the similarity of each image for one of the three features. For each feature i, ($i = \{$color, texture, structure$\}$), the system computes a query vector $\vec{q}_i$ based on the positive and negative examples specified by the user. Then, it calculates the feature distance g_{ni} between each image n and the query vector as follows:

$$g_{ni} = (\vec{p}_{ni} - \vec{q}_i)^{\mathrm{T}} W_i (\vec{p}_{ni} - \vec{q}_i) \tag{1}$$

where $\vec{p}_{ni}$ is the feature vector of image n regarding the feature i. For the computation of the distance W_i, matrix , we used biased discriminant analysis (BDA.) The detail of BDA is described in [29]. After the feature distances are computed, the system combines each feature distance g_{ni} into the *total distance* d_n. The total distance of image n is the weighted sum of each g_{ni},

$$d_n = \vec{u}^{\mathrm{T}} \vec{g}_n \tag{2}$$

where $\vec{g}_n = [g_{n1}, \ldots, g_{nI}]$. I is the total number of the features. In our case, I is 3. The optimal solution of the feature weighting vector $\vec{u} = [u_1, \ldots u_I]$ is solved by Rui et al. [12] as follows:

$$u_i = \sum_{j=1}^{l} \sqrt{f_j / f_j} \tag{3}$$

where $f_i = \sum_{n=1}^{N} g_{ni}$, and N is the number of the positive examples. This gives a higher weight to that feature whose total distance is small. This means that if the positive examples are similar with respect to a certain feature, this feature gets higher weight. Finally, the images in the database are ranked by the total distance. The system returns the k most similar images.

7. Future work

We plan to evaluate our system further with respect to both usability and query performance. Especially, we will investigate the effect of *Groups in a Group* query described in Section 2.3. As mentioned in [30], the traditional precision/recall measure is not very suitable for the evaluation for interactive retrieval systems. Therefore, we may need to consider suitable evaluation methods for the system [16,31].

Next, in the current system, when more than one group is selected as positive, they are merged into one group, i.e. all images in those groups are considered as positive examples. This method, however, does not take advantages of the new user interface. We are investigating a scheme where different positive groups are considered as different classes of examples [32].

In addition, for the advanced users, we are going to add support for the group-wise feature selection. Although our system automatically determines the feature weights, the advance users might know which features are important for their query. Thus, we will allow the users to specify which features are supposed to be considered for each group. Some groups might be important in terms of the color features only, while others might be important in terms of the structures. Finally, because the implementation of *ImageGrouper* does not depend on the underlying retrieval technologies, it can be used as a benchmarking tool [31] for various image retrieval systems.

8. Conclusion

In this paper, we presented *ImageGrouper*, a new group-oriented user interface for digital image retrieval and organization. In this system, the users search, annotate, and organize digital images by groups. *ImageGrouper* has several advantages regarding image retrieval, text annotation, and image organization.

First, in the content-based image retrieval (CBIR), predicting a good combination of the query examples is very difficult. Thus, the *trial-and-error* is essential for successful retrieval. However, the previous systems assumed only *incremental* relevance feedback and do not support *trial-and-error* search. On the other hand, the *Query-by-Groups* concept of *ImageGrouper* allows the user to try different combinations of query examples quickly and easily. We showed this simple operation helps the users to achieve higher recall rate. Second, with the *Groups in a Group* configuration, the user can search images by *narrowing down* the scope of the search from the general concept to more specific concept.

Next, typing text information to many images is very tedious and time consuming. *Annotate-by-Groups* method eases the users of this task by allowing them to annotate multiple images simultaneously. *Groups in a group* method realizes a hierarchical annotation, which was difficult in the previous systems. Moreover, by allowing the groups to overlap to each other, *ImageGrouper* further reduces typing.

In addition, our concept of image groups is also applied for organizing the image collections. A group in the *GroupPalette* can be shrunk into a small icon. These icons can be used as "photo albums" which can be directly manipulated and organized by the users.

Furthermore, we compared the usability of our system with those of two traditional GUIs in a simple scenario. The lesson we learned from the experiments helped us to improve the usability of our system further while preserving the greater flexibility of the system.

Finally, these three concepts: *Query-by-Groups*, *Annotation-by-Groups and Organize-by-Groups* share the similar gestural operations: that is, *dragging images and drawing a rectangle surrounding them*. Thus, once the user learned one task, s/he can easily adapt herself/himself to the other tasks. The operations in *ImageGrouper* are also similar to the file operations used in Windows and Macintosh computers as well as most drawing programs. Therefore, the user can easily learn to use the system.

The prototype of *ImageGrouper* is available at http://www.ifp.uiuc.edu/~nakazato/grouper/.

Acknowledgements

This work was supported in part by National Science Foundation Grant CDA 96-24396.

References

[1] B.B. Bederson, Quantum Treemaps and Bubblemaps for a Zoomable Image Browser. HCIL Tech Report #2001-10, University of Maryland, College Park, MD 20742.
[2] J.-Y. Chen, C.A. Bouman, J.C. Dalton, Heretical browsing and search of large image database, IEEE Transactions on Image Processing 9 (3) (2000) 442–455.
[3] J. Laaksonen, M. Koskela, E. Oja, Content-based image retrieval using self-organization maps. In: Proceedings of 3rd International Conference in Visual Information and Information Systems, Amsterdam, The Netherlands, 1999.
[4] Z. Pecenovic, M.-N. Do, M. Vetterli, P. Pu, Integrated browsing and searching of large Image collections, In: Proceedings of Fourth International Conference on Visual Information Systems, Lyon, France, November 2000.
[5] M. Flickner, H. Sawhney, et al., Query by image and video content: the QBIC system, IEEE Computer 28 (9) (1995) 23–32.
[6] Y. Rui, T.S. Huang, M. Ortega, M. Mehrotra, Relevance feedback: a power tool for interactive content-based image retrieval, IEEE Transaction on Circuits and systems for Video Technology 8, 1998, pp. 644–655.
[7] J.R. Smith, S.-F. Chang, VisualSEEk: a fully automated content-based image query system, ACM Multimedia '96, Boston, Massachusetts, USA, 1996.
[8] M. Sticker, M. Orengo, Similarity of Color Images, In: Proceedings of SPIE, Storage and Retrieval of Image and Video Databases III, Vol. 2420, SPIE Press, WA, USA, 1995, pp. 381–392.
[9] J.R. Smith, S.-F. Chang, Transform features for texture classification and discrimination in large image databases. In: Proceedings of IEEE International Conference on Image Processing, Austin, Texas, USA, 1994.
[10] X.S. Zhou, T.S. Huang, Edge-based structural feature for content-base image retrieval, Pattern Recognition Letters (Special issue on Image and Video Indexing) 22 (5) (2001) 457–468.
[11] Y. Ishikawa, R. Subrammanya, C. Faloutsos, MindReader: Query database through multiple examples, In: Proceedings of the 24th VLDB Conference, New York City, NY, USA, 1998.
[12] Y. Rui, T.S. Huang, Optimizing learning in image retrieval, In: Proceedings of IEEE CVPR '00, 2000.
[13] S. Santini, R. Jain, Integrated browsing and querying for image database, IEEE Multimedia 7 (3) (2000) 26–39.
[14] K. Rodden, W. Basalaj, D. Sinclair, K. Wood, Does organization by similarity assist image browsing? In: CHI '01. 2001.
[15] I.J. Cox, M.L. Miller, T.P. Minka, T.V. Papathomas, P.N. Yianilos, The Bayesian image retrieval System, PicHunter: theory, implementation, and psychophysical experiments, IEEE Transactions on Image Processing 9 (2000) 20–37.
[16] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, R. Jain, Content-based image retrieval at the end of the early years, IEEE PAMI 22, (2000).
[17] M.J. Bates, The design of browsing and berrypicking techniques for the on-line search interface, Online Review 13 (5) (1989) 407–431.

[18] V.L. O'Day, R. Jeffries, Orienteering in an information landscape: how informationseekers get from here to there, INTERCHI '93, 1993.
[19] S. Jones, Graphical query specification and dynamic result previews for a digital library, In: UIST '98, San Francisco, CA, USA, 1998.
[20] A. Kuchinsky, C. Pering, M.L. Creech, D. Freeze, B. Serra, J. Gwizdka, FotoFile: A consumer multimedia organization and retrieval system, In: CHI '99, 1999.
[21] B. Shneiderman, H. Kang, Direct annotation: a drag-and-drop strategy for labeling photos, In: Proceedings of the IEEE International Conference on Information Visualization (IV'00), Hague, The Netherlands, 2000.
[22] M. Balabanovic, L.L. Chu, G. J. Wolff, Storytelling with digital photographs, In: CHI '00, 2000.
[23] S.B. Cousins, et al., The digital library integrated task environment (DLITE), In: 2nd ACM International Conference on Digital Libraries, Philadelphia, Pennsylvania, USA, 1997.
[24] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison Wesley, Reading MA, 1999.
[25] M. Nakazato, et al., UIUC Image Retrieval System for JAVA, available at http://www.ifp.uiuc.edu/~nakazato/cbir/.
[26] A. Hiroike, Y. Musha, Visualization for similarity-based image retrieval systems, IEEE Symposium on Visual Languages, Tokyo, Japan, 1999.
[27] A. Hiroike, Y. Musha, A. Sugimoto, Y. Mori, Visualization of information space to retrieve and browse image data, In: Proceedings of Visual '99: Information and Information Systems, Amsterdam, The Netherlands, 1999.
[28] M. Nakazato, T.S. Huang, 3D MARS: Immersive virtual reality for content-based image retrieval, In: Proceedings of IEEE International Conference on Multimedia and Expo 2001.
[29] X. Zhou, T.S. Huang, A generalized relevance feedback scheme for image retrieval. In: Proceedings of SPIE Vol. 4210: Internet Multimedia Management Systems, 6–7 November, Boston, Massachusetts, USA, 2000.
[30] E. Lagergren, P. Over, Comparing interactive information retrieval systems across sites: the TREC-6 interactive track matrix experiment, ACM SIGIR '98, 1998.
[31] H. Müller, et al., Automated benchmarking in content-based image retrieval. In: Proceedings of IEEE International Conference on Multimedia, Expo 2001, August, 2001.
[32] X.S. Zhou, N. Petrovic, T.S. Huang, Comparing discriminating transformations and SVM for learning during multimedia retrieval, In: ACM Multimedia '01, Ottawa, Ontario, Canada, 2001.