Group-based Interface for Content-based Image Retrieval

Munehiro Nakazato
Beckman Institute
University of Illinois
at Urbana-Champaign
nakazato@uiuc.edu

Ljubomir Manola
School of Electrical Engineering
University of Belgrade
manola@yubc.net

Thomas S. Huang
Beckman Institute
University of Illinois
at Urbana-Champaign
huang@ifp.uiuc.edu

ABSTRACT

In Content-based Image Retrieval (CBIR) systems, the Query-by-Example (QBE) approach is commonly used. However, because of inevitable "semantic gaps" between visual features and the user's concepts, trial-and-error query is essential for successful retrieval. Unfortunately, traditional user interfaces are not suitable for trying different combinations of query examples. This is because in these systems, query specification and 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. In our system, the users can interactively compare different combinations of query examples by dragging and grouping images on the workspace (Query-by-Group.) Because the query results are displayed on another pane, the user can quickly review the results. Combining different queries is also easy. Furthermore, the concept of "image groups" is also applied to annotating and organizing a large number of images. Because the gestural operations of our system is similar to file operations of modern window-based operation systems, users can easily learn to use the system.

Categories and Subject Descriptors

H.5.2 [**Information Interface and Presentation**]: User Interfaces - *graphical user interfaces (GUI), interaction styles*.

General Terms

Design, Human Factors.

Keywords

Content-based image retrieval, image database, information retrieval, digital photography.

1. INTRODUCTION

More and more people are enjoying Digital imaging these days [9][16]. New inexpensive and high quality digital cameras hit the market every month. Even camera-equipped mobile phones have appeared [26]. We can take plenty of digital pictures without worrying about the price of films. We can easily edit pictures, as we like. We can share the images with our family or friends by E-mail and World-Wide Web.

The problem is that searching and organizing digital images are not as easy as the traditional photos. Because we can take pictures as many as the memory is available and as long as the battery remains, we have to manage a large number of image files. Moreover, those image files may not have any understandable names (they are usually named like "DSCF0052.JPG" and so on.)

Many researchers have proposed ways to find an image from large image databases. We can divide these approaches into two types of interactions: *Browsing* and *Searching*. In image browsing, the users look through the entire collections. In most systems, the images are clustered in hierarchical manner and the user can traverse the hierarchy by zooming and panning [4][5][14][21]. In [21], browsing and searching are integrated so that the user can switch back and forth between browsing and searching.

Image searching can be further divided into *Keyword-based* and *Content-based*. The advantages of Keyword-based approaches are that the user can retrieve images with high-level concepts of images such as object names in the image or the location where the picture was taken. However, in order to make a keyword-based approach effective, the users have to annotate all images manually. While this might make sense for commercial photo stocks, it is extremely tedious tasks for home users.

Meanwhile, enormous amount of research have been done for *Content-Based Image Retrieval* (CBIR) [10][23][30]. In CBIR systems, the user searches image by visual similarity, i.e. low-level image features such as color [31], texture [29] and structure [32]. They are automatically extracted from all images and indexed 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-Example*. In this method, the users select example images (as positive or negative) and ask the system to retrieve visually similar images. In addition, in order to improve the retrieval further, CBIR systems often employ *Relevance Feedback* [11][23][24], 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 novel user interface for digital image retrieval and organization, named *ImageGrouper*. In *ImageGrouper*, a new concept *Query-by-Groups* is introduced for Content-based Image Retrieval (CBIR.) The users construct a query by making groups of images. The groups are easily created by dragging images on the interface. Because the image groups can be easily reorganized, flexible retrieval was achieved. Moreover, with the similar interaction methods, the user can effectively annotate and organize a large number of images.

In the next section, we present a new concept of user interface for CBIR. Then, the following sections describe new paradigm for image annotation and organization introduced in our system.

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 Content-based Image Retrieval (CBIR) systems [21][25]. Figure 1 shows a typical GUI for CBIR system that supports *Query-by-Example*. 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 [22]. In some cases, they are images 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. If the user thinks an image is relevant, s/he moves a slider to the right. If s/he thinks an 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 check boxes or by clicking on the images [8]. In these cases, the degrees are not specified.

When the "Query" button is pressed, the system computes the similarity between selected images and database images, then retrieves the *N* most similar images. The grid images are replaced with the retrieved images. These images are ordered based on the degree of similarity.

If the user finds other relevant images in the result set, s/he selects them as new query examples. If a highly irrelevant image appears in the result set, the user can select it as a negative example. Then, the user press "Query" again. The user can repeat this process until s/he is satisfied. This process is called *relevance feedback* [11][23][24]. Moreover, in many systems, there are interfaces to weight the importance of image features such as color and texture.

Figure 1: Typical GUI for CBIR Systems

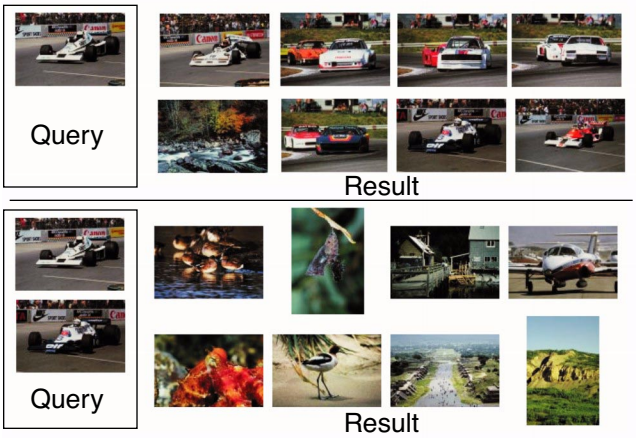

Figure 2: Example of "More is not necessarily better". The top row shows the query result of only one example, while the bottom row shows the result when an additional example was added. The cone example case achieved better result. In one example query, 7 of the top 8 images are "cars" while in two example case, there is no car images in the top 8.

In [28], Smeulders et al. classified *Query by Image Example* and *Query by Group Example* into different categories. From 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 Query by Examples method. In stead, we use term "Query by Groups" to refer our new model of query specification method described later.

This type of traditional GUI has several drawbacks. First of all, these systems assume that *the more query examples are available, the better result we can get*. Therefore, the users are supposed to search images *incrementally* by adding new example images from the result of the previous query. However, this assumption is not necessarily true. Additional examples may contain undesired visual features and degenerate the retrieval performance.

Figure 2 shows an example of situations when more query examples could lead to worse results. In this example, the user is trying to retrieve pictures of cars. The first row shows the query result when only one image of "car" is used as a query example. The second row shows the result of two query examples. The results are ordered based on the similarity ranks. In both cases, the same relevance feedback algorithm (Section 5.2 and [24]) was used and tested on Corel image set of 17000 images. In this example, even if this additional example image looks visually good for human eyes, it introduces undesirable features into the query. Thus, no car image appears in 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 image retrieval and confuses the users. This problem happens because of *semantic gap* [25][28] between the high-level concept in the user's mind and the extracted features of images. Furthermore, finding good combinations of query examples is very difficult because image features are numerical values that are impossible to be estimated by human. Only way to find the right combination is *trial and error*. Otherwise, the user can be trapped in a small part of image database [21].

Unfortunately, the traditional user interfaces were designed for *incremental search* and are not suitable for *trial and error query,* if not impossible. This is because in these systems, query

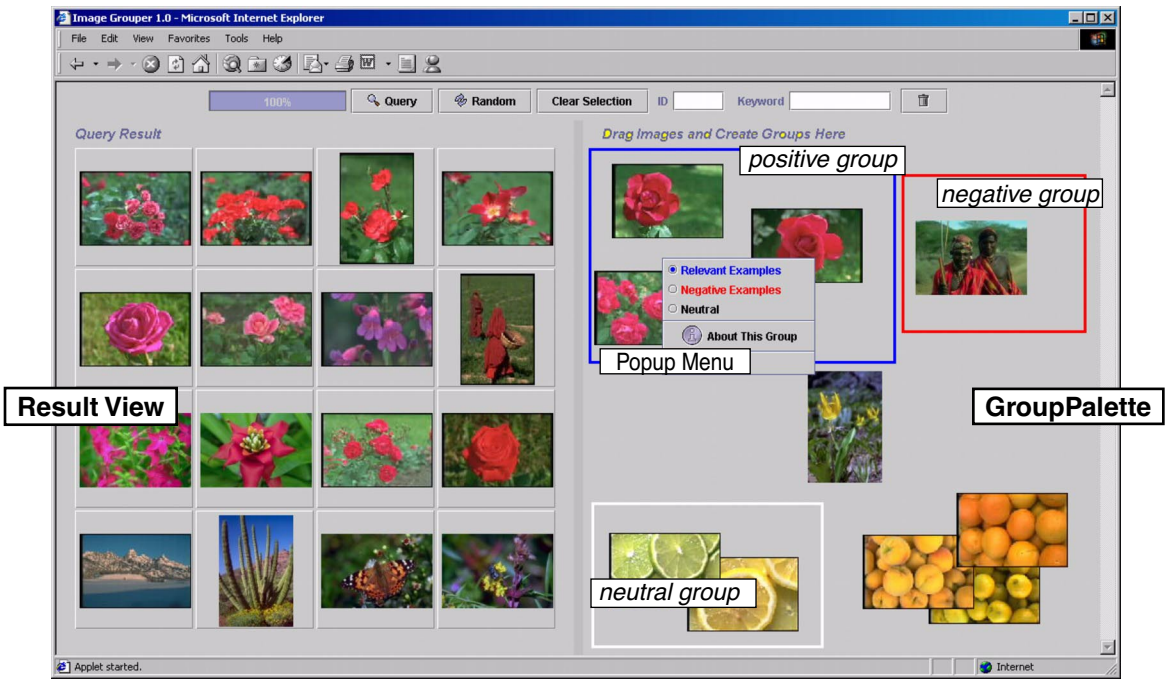

Figure 3: The ImageGrouper

specification and result display must be done on the same workspace. Once the user removes an image from the query examples during relevance feedback loops, the image may disappear from the user interface. 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 of CBIR are not necessarily looking for only one type of images. The users' interest may change during retrieval. This behavior is known as *berry picking* [3] and have been observed for text documents retrieval by O'Day and Jeffries [20].

Moreover, because of *semantic gap* [25][28] mentioned above, the users often need to make more than one query to satisfy their need [3]. 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 low-level visual features. Thus, the user needs to retrieve "beautiful flowers" as a collection of different types of images.

Finally, in some case, the user had better start from a general concept of objects and narrow down to specific ones. For example, suppose the user is looking for images of "red cars." Because image retrieval systems use various image features [29][32] as well as colors [31], 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 Query by Examples was proposed by Santini et.al [25]. In their El Ninõ system, the user specifies a query by mutual distance between example images. The user drags images on the workspace so that the 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 significant drawbacks in El Ninõ system. First, it is unknown to the users how close similar images should be located and how far negative examples should be apart from good examples. It may take a while for the user to learn "the metric system" used in this interface. The second problem is that like traditional interfaces, query specification and result display are done on the same workspace. This makes trial and error query difficult. Given the analogue nature of the 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 the paper.

2.2 Query-by-Groups: The ImageGrouper Approach

Figure 3 shows the display layout of *ImageGrouper*. In this system, a new concept *Query-by-Groups* was introduced. Query-by-Groups mode is an extension to Query-by-Example mode described above. The major difference is that while Query-by-Example handles the images individually, in *Query-by-Group*, a "group of images" is considered as a unit of the query.

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*, in which the user manages each image and image groups. In order to create an image group, the user first drags one or more images from the *ResultView* into *GroupPalette*, then encloses the images by drawing a rectangle (box) as we draw a rectangle in drawing applications. All the images within the *group box* become the member of this group. The user can create multiple groups in the palette. In addition, 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 just drags it out of the box.

When the right mouse button is pressed on a group box, a popup menu appears so that the user can give query properties (positive, negative, or neutral) to the group. The properties of groups can be changed at any moment. The Color 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 (Figure 3.) Then, the system retrieves new images that are similar to images in positive groups while avoiding images similar to negative groups. The result images are displayed in the *ResultView*.

When a group is specified as *neutral* (displayed as a white box), this group does not affect the result of the retrieval. This group can be turned to a positive or negative group later. If a group is *positive* (displayed as a blue box), the system uses common features among the images in the group. On the other hand, if a group is given *negative* (red box) property, the common features in the group are used as negative feedbacks for the similarity matching. The user can specify multiple groups as positive or negative. In this case, these groups are merged into one group, i.e. AND of groups are taken. The detail of the algorithm is described in Section 5.2.

In the example shown in Figure 3, the user is retrieving images of "flowers." In 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 consists of only one image. Below the "flowers" group, there is a neutral group (white box), which is not used for retrieval at this moment. Images can be moved outside of any groups in order to temporarily remove images from the groups

2.3 Flexible Image Retrieval

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

2.3.1 Trial and Error Query by Mouse Dragging

First of all, images can easily enter or leave a group by mouse drags. It makes trial and error of relevance feedbacks easier. The user can explore different combinations of query examples by dragging images into or out of the box. Images that are not used for query can be kept in the outside of the boxes in the palette and can be reused for another query later.

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.

Figure 4 shows an example of *Groups in a Group*. Here, the user is looking for pictures of "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 red car images, though their colors features are different. The large white box in Figure 4 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 divide the collected images into two sub-groups by creating two new boxes for red cars and other cars. Then the user specifies the red car group as positive and the other cars group as negative, respectively. In Figure 4, the left smaller (blue, i.e. positive) box is the group for red cars and the right box (red, i.e. negative) is the group for cars with other than red. This *narrow down search* was not possible on the conventional CBIR systems.

2.4 Experiment on the Trial and Error Query

In order to examine the effect of ImageGrouper's flexible retrieval, we compared the query performance of our system with that of a traditional incremental approach. In this experiments, we used Corel photo stock that contains 17000 images as the data set. For both interfaces, the same image features and relevance feedback algorithms (described in Section 5.2) are used.

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

We tested over eight classes of images (Table 1). For each class, a query starts from one image example. In case of the traditional interface, the query is repeated 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, 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 is temporarily decreased.) Then, the search is refined incrementally again until another convergence occurs. The user repeats this until trial and error query has no effect.

The results of the experiments are shown in Table 1. "Hits" means the number of retrieved images after the convergence (or 10 rounds.) This value is proportional to the Recall rate. Thus, larger value means better retrieval performance. "Round" means the number of relevance feedback loops until convergence. Round=10 means the query did not converge before the 10th round.

Clearly, ImageGrouper can achieve better retrieval (higher recall) even if 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 small number of images after a few iterations. This result suggests the importance of support for *trial and error* query.

Object	Incremental Search		ImageGrouper	
	Hits	Round	Hits	Round
Red Car	6	4	12	10
Tiger	11	3	17	10
Bird	7	4	8	8
Yellow Flower	7	3	14	10
Citrus	2	2	5	4
Polar Bear	9	5	17	10
Elephant	5	3	10	10
Swimmer	11	2	21	10

Table 1: Comparing the traditional search and trial and error search. Tested on eight classes of images

3. TEXT ANNOTATIONS ON IMAGES

Keyword-based search is a very powerful method for searching images. The problem is that it works well only when all the images are annotated with textual information. For commercial

photo stocks, it may be feasible to enter keywords to all images manually. For home users, however, it is too tedious.

When keyword search is integrated with CBIR like our system and [21], keyword-based search can be used to find the initial query examples for 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 a large number of images.

3.1 Current Approaches for Text Annotation

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

Several researchers have proposed smarter user interfaces for keyword annotation on images. In *bulk annotation* method of *FotoFile* [13], 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 at the same time. To retrieve images, the user selects entries from the menu, and then presses the "Search" button. Because of *visual and gestural symmetry* [13], the user needs to learn only one tool for both annotation and retrieval.

PhotoFinder [27] 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 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.

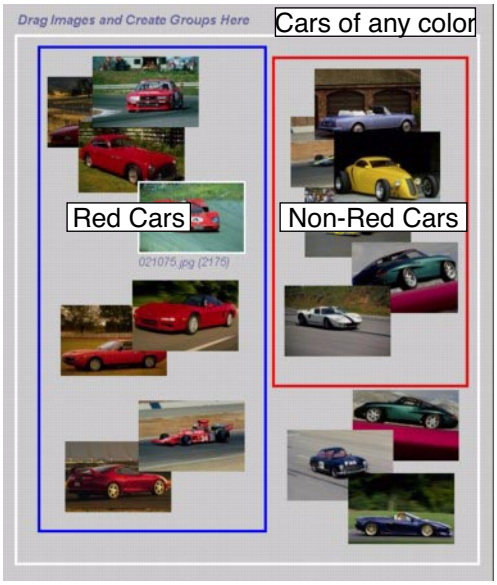

Figure 4: *Groups in a group*. Here, the user is looking for images of "Red Cars." S/he start with retrieval of "Cars with any color" (white box) Then, the search is narrowed down by selecting "red cars" as a positive group, "cars with colors other than red" as a negative group, respectively.

3.2 Annotation by Groups

Most home users do not want to annotate images one by one, especially when the number of images is large. In many cases, 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 keywords are annotated not on individual images, but on groups. As in *Query-by-Groups*, the user first creates a group of images by dragging images from *ResultView* into *GroupPalette* and drawing a rectangle around them. In order to give keywords to the group, the user opens *Group Information Window* (Figure 5) by selecting "About This Group" from the pop-up menu (Figure 3). In this window, arbitrary number of words can be added. Because the users can simultaneously annotate the same keywords on a number of images, annotation becomes much faster and less error prone. *Group Information Window* is also used to browse the information of individual images in the groups ("Members" tab in Figure 5.)

Although *Annotation-by-Groups* is similar to bulk annotation of *FotoFile* [13], there are several advantages.

3.2.1 Annotating New Images with the same keywords

In *bulk annotation* [13], 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 same steps (i.e. select images, select keywords from the list, then press "Annotate".) This is awkward when the user has to add a large number of keywords. Meanwhile, in *Annotation-by-Group*, the system attaches annotations not on each images, but on 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 by Groups in a Group

In *ImageGrouper,* the user can annotate images hierarchically using *Groups in a Group* method described above (Figure 4). For example, the user may want to add new keyword "Trevi Fountain" to only a part of the image group that has been labeled "My

Figure 5: *Group Information Window*: Group Properties tab is for Query and Keyword Annotation, Members List tab is for investigating individual images

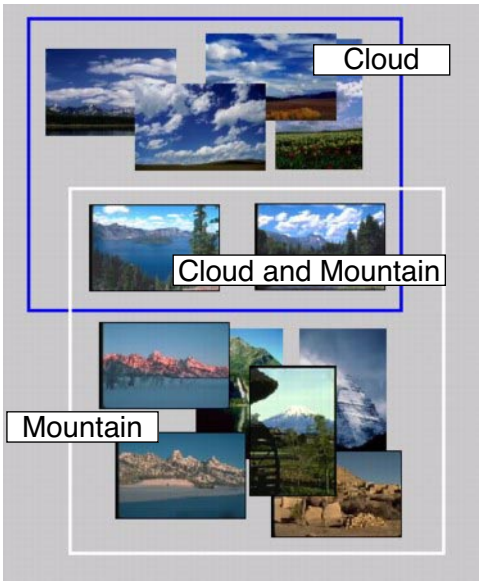

Figure 6: Overlap between groups. There are two groups for "Mountain" and "Clouds." Two images in the overlapped region contain both mountain and cloud.

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* [13] with bulk annotation, the user has to select some 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 on the *GroupPalette* first, then edit 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 references* by allowing overlaps between image groups, i.e. an image can belong to multiple groups at the same time. For example, in Figure 6, there are two image groups: "Cloud" and "Mountains." Because some images contain both cloud and mountain, these images belong to both groups. They are automatically referred as "Cloud and Mountain." This concept is not supported in other systems.

3.3 Summary of Text Annotation

ImageGrouper introduced three new concepts for text annotation on images. *Annotation-by-Groups* dramatically reduces the number of typing required and enables quick and easy annotation. Moreover, to annotate new images with the same keywords as an existing group, the user simply drags the new images into the group.

Next, *Groups in a Group* allows the user to annotate images hierarchically. The hierarchy is visualized on the display. Finally, *Group Overlapping* makes it possible to refer an image in different contexts. These techniques also reduce the number of typing and provide the users with greater flexibility. Meanwhile, the simplicity of *Annotation-by-Groups* is still preserved.

4. ORGANIZING IMAGES BY GROUPS

In the previous two sections, we described how *ImageGrouper* support content-based query as well as keyword-based 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 content-based retrieval with *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 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 or when the user is searching images based on visual appearance only. First, the user retrieves some images by keywords. If there are interesting ones in the results, s/he can create a query group from them as the first examples. Then, the user refines the search by *Query-by-Groups* method.

4.1 Photo Albums and Group Icons

As described above, *ImageGrouper* allows groups to be overlapped. In addition, the user can attach textual information on these groups. Therefore, groups in *ImageGrouper* can be used to organize pictures as "photo albums." Similar concepts are proposed in *FotoFile* [13] and Ricoh's *Storytelling* system [1]. In both systems, albums are used for "slide shows" to tell stories to the other users.

In *ImageGrouper,* the user can convert a group into a *group icon*. When the user selects "Iconify" from the popup menu (Figure 3,) images in the group disappear and a new icon for the group appears in *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 box realizing *groups in a group*.

Finally, group icons themselves can be used as examples for 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) [7]. In DLITE, each text documents as well as the search results are visually represented by icons. The user can directly manipulate those documents in a workcenter (*direct-manipulation*.) In [12], Jones proposed another graphical tool for query specification, named *VQuery.*

While DLITE and VQuery were systems for text documents, the idea of *direct-manipulation* [7] is applicable more naturally to image databases. In 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 investigate the detail [6] (in case of DLITE, a web browser is opened.) On the other hand, in image databases, images themselves (or their thumbnails) can be used for direct-manipulations. Therefore, instant judgement by the user is possible [21][28].

5. IMPLEMENTATION DETAILS

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

5.1 The User Interface Client

The user interface client of *ImageGrouper* is implemented as a Java2 Applet with Swing API (Figure 3). Thus, the users can use the system through Web browsers on various platforms such as Windows, Linux, Unix and Mac OS X. Because we chose HTTP as a communication protocol, even the users within Firewalls can access to the servers.

The client interacts with the user and determines his/her interests from the group information or keywords input. When "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* of the interface. The user interface client can access to image databases at remote locations as well as on the local machine.

Note that the user interface of *ImageGrouper* is independent of relevance feedback algorithms [11][23][24] 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 image features used, the usability of *ImageGrouper* is not affected by those factors.

5.2 The Query Server

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

The server is implemented as a Java Servlet that runs on the Apache Web Server and Jakarta Tomcat Servlet container. It is written in Java and C++. In addition, the server is implemented as a *stateless server*, i.e. the server does not hold any information about the clients. This design allows different types of clients such as the traditional user interface [18] (Figure 1) and 3D Virtual Reality interface [19] can access to the same server simultaneously.

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

5.2.1 Image Features

As the visual features for content-base image retrieval, we use three types of features: *Color*, *Texture*, and *Edge Structure*.

For *color* features, HSV color space is used. We extract the first two moments (mean, and standard deviation) from each of HSV channels [31]. Therefore, the total number of color features is six. For *texture*, each image is applied into *wavelet filter bank* [29] 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. Finally, for *edge structures*, we used *Water-Fill edge detector* [32] to extract image structures. We first pass the original images through the edge detector to generate their corresponding edge maps. From the edge map, eighteen (18) elements are extracted from the edge maps.

5.2.2 Relevance Feedback Algorithm

The similarity ranking is computed as follows. First, the system computes the similarity of each image with respect to only one of the features. For each feature $i = \{\text{color, texture, structure}\}$, 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. For the computation of the distance, we used *Biased Discriminant Analysis* (BDA.) The detail of BDA is described in [33].

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 a weighted sum of each g_{ni},

$$d_n = \vec{u}^T \vec{g}_n \quad (1)$$

where $\vec{g}_n = [g_{n1}, \ldots, g_{nI}]$. I is the total number of 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.[24] as follows,

$$u_i = \sum_{j=1}^{I} \sqrt{f_j / f_i} \quad (2)$$

where $f_i = \sum_{n=1}^{N} g_{ni}$, and N is the number of positive examples. This gives higher weight to that feature whose total distance is small. Which 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.

6. 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 [15], traditional precision/recall measure is not very suitable for evaluation for interactive retrieval systems. Therefore, we may need to consider appropriate evaluation methods for the system [17][28].

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. We are investigating a scheme where different positive groups are considered as different classes of examples [33].

In addition, for the advanced users, we are going to add support for 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 color features only, while others might be important in terms of structures. We believe this will extend the expressiveness of our system.

Finally, because the implementation of ImageGrouper does not depend on underlying retrieval technologies, it can be used as a benchmarking tool [17] for various image retrieval systems.

7. CONCLUSION

In this paper, we presented *ImageGrouper*, a new 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 content-based image retrieval (CBIR), predicting a good combination of query examples is very difficult. Thus, *trial-and-error* is essential for successful retrieval. However, the previous systems are assuming *incremental* search and do not support *trial-and-error* search. On the other hand, *Query-by-Groups* concept in *ImageGrouper* allows the user to try different combinations of query examples quickly and easily. We showed this lightweight operation helps the users to achieve higher recall rate.

Second, with *Groups in a Group* configuration, the user can start by searching general concepts and then narrow down the search to more specific concepts. This method helps the user find both positive and negative examples, and provides him/her with more choices.

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

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

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

8. ACKNOWLEDGEMENT

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

9. REFERENCES

[1] Baeza-Yates, R. and Ribeiro-Neto, B. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] Balabanovic, M., Chu, L.L. and Wolff, G.J. Storytelling with Digital Photographs. In *CHI'00*, 2000.

[3] Bates, M.J. The design of browsing and berrypicking techniques for the on-line search interface. *Online Review*, 13(5), pp. 407-431, 1989.

[4] Bederson, B.B. Quantum Treemaps and Bubblemaps for a Zoomable Image Browser. *HCIL Tech Report #2001-10*, University of Maryland, College Park, MD 20742.

[5] Chen, J-Y., Bouman, C.A., and Dalton, J.C. Heretical Browsing and Search of Large Image Database. In *IEEE Transaction on Image Processing*, Vol. 9, No. 3, pp. 442-455, March 2000.

[6] Card, S.K., Mackinlay, J.D., and Shneiderman, B. (Editors) *Readings in Information Visualization: Using Vision to Think,* Morgan Kaufmann, 1999.

[7] Cousins, S.B., et al. The Digital Library Integrated Task Environment (DLITE). In *Proceedings of the 2nd ACM International Conference on Digital Libraries*, 1997.

[8] Cox, I.J. et al. The Bayesian Image Retrieval System, *PicHunter*: Theory, Implementation, and Psychophysical Experiments. In *IEEE Transactions on Image Processing*, Vol. 9, No. 1, January 2000.

[9] Gonzales, D. Digital Cameras are no longer just for the Digerati. *New York Times*, November 25, 2001.

[10] Flickner, M., Sawhney, H. and et al. Query by Image and Video content: The QBIC system. In *IEEE Computer*, Vol. 28, No.9, pp. 23-32, September 1995.

[11] Ishikawa, Y., Subrammanya, R. and Faloutsos, C. MindReader: Query database through multiple examples. In *Proceedings of the 24th VLDB Conference*, 1998.

[12] Jones, S. Graphical Query Specification and Dynamic Result Previews for a Digital Library. In *UIST'98*, 1998.

[13] Kuchinsky, A., Pering, C., Creech, M.L., Freeze, D., Serra, B. and Gwizdka, J. FotoFile: A Consumer Multimedia Organization and Retrieval System. In *CHI'99*, 1999.

[14] Laaksonen, J., Koskela, M. and Oja, E. Content-based image retrieval using self-organization maps. In *Proceedings of 3rd International Conference in Visual Information and Information Systems,* pp. 541-548, 1999.

[15] Lagergren, E. and Over, P. Comparing interactive information retrieval systems across sites: The TREC-6 interactive track matrix experiment. In *Proc. of ACM SIGIR'98,* 1998.

[16] Miles, S. Camera buyers increasingly focus on digital. *CNET NEWS.com,* September 26, 2000.

[17] Müller, H et al. Automated Benchmarking in Content-based Image Retrieval. In *Proceedings of IEEE International Conference on Multimedia and Expo 2001,* August, 2001.

[18] Nakazato, M. et al., UIUC Image Retrieval System for JAVA, available at http://chopin.ifp.uiuc.edu:8080.

[19] Nakazato, M. and Huang, T.S. 3D MARS: Immersive Virtual Reality for Content-based Image Retrieval. In *Proc. of IEEE International Conference on Multimedia and Expo 2001,* August, 2001.

[20] O'Day V. L. and Jeffries, R. Orienteering in an information landscape: how informationseekers get from here to there. In *Proceedings of the INTERCHI '93,* 1993.

[21] Pecenovic, Z. et al. Integrated Browsing and Searching of Large Image Collections. In *Proceedings of Fourth Intl. Conf. on Visual Information Systems*, November, 2000.

[22] Rodden, K., Basalaj, W., Sinclair, D. and Wood, K. Does Organization by Similarity Assist Image Browsing? In *CHI'01*. 2001.

[23] Rui, Y., Huang, T. S., Ortega, M. and Mehrotra, M. Relevance Feedback: A Power Tool for Interactive Content-Based Image Retrieval. In *IEEE Transaction on Circuits and Video Technology*, Vol. 8, No. 5, Sept. 1998.

[24] Rui, Y. and Huang, T. S., Optimizing Learning in Image Retrieval. In *Proceedings of IEEE CVPR '00*, 2000.

[25] Santini, S. and Jain, R. Integrated Browsing and Querying for Image Database. *IEEE Multimedia*, Vol. 7, No. 3, 2000, pp. 26-39.

[26] Sharp. J-SH07. http://www.j-phone.com/f-e/j/products/j-sh07/back.html and http://www.sharp.co.jp/products/jsh07/index.html

[27] Shneiderman, B. and Kang, H. Direct Annotation: A Drag-and-Drop Strategy for Labeling Photos. In *Proceedings of the IEEE Intl. Conf. on Information Visualization (IV'00)*, 2000.

[28] Smeulders, A.W.M., Worring, M., Santini, S., Gupta, A. and Jain, R. Content-based Image Retrieval at the End of the Early Years. In *IEEE Pattern Analysis and Machine Intelligence* Vol. 22, No. 12, December, 2000.

[29] Smith, J.R. and Chang S-F. Transform features for texture classification and discrimination in large image databases. In *Proceedings of IEEE Intl. Conf. on Image Processing,* 1994.

[30] Smith, J.R. and Chang S-F. VisualSEEk: a fully automated content-based image query system. In *Proc. of ACM Multimedia'96*, 1996.

[31] Sticker, M. and Orengo, M., Similarity of Color Images. In *Proceedings of SPIE*, Vol. 2420 (Storage and Retrieval of Image and Video Databases III), SPIE Press, Feb. 1995.

[32] Zhou, X. S. and Huang, T. S. Edge-based structural feature for content-base image retrieval. *Pattern Recognition Letters, Special issue on Image and Video Indexing*, 2000.

[33] Zhou, X. S., Petrovic, N. and Huang, T. S. Comparing Discriminating Transformations and SVM for Learning during Multimedia Retrieval. In *Proc. of ACM Multimedia'01,* 2001.