Content-Based Queries in Image Databases

Uri Shaft
Raghu Ramakrishnan

Technical Report #1309

March 1996

Paper Number 5024

Content-Based Queries in Image Databases

Uri Shaft and Raghu Ramakrishnan
(uri, raghu) @cs.wisc.edu
Computer Sciences Department
Univ. of Wisconsin-Madison

February 21, 1996

Abstract

Current image retrieval systems have many important limitations. Many are specialized for a particular class of images and/or queries. The more general systems support relatively weak querying by content (e.g., by color, texture or shape, but with no deeper understanding of the structure of the image). Few (if any) have addressed the issue of truly large collections of images, and how the underlying techniques scale.

There are many aspects to a DBMS supporting image retrieval by content. In this paper, we focus on a *data model* and give an example of a *data definition language* (DDL) for image data, and demonstrate the gains to be had by incorporating such a DDL in a general-purpose image DBMS. Specifically, we make contributions in five areas:

(1) A proposal for a data model for images. (2) The use of DDL definitions for guiding automatic feature extraction, using a constraint-based scheduling algorithm that calls upon a library of standard and specialized image analysis routines. (3) The use of extracted features (based on the data model and DDL definitions) in representing and indexing large sets of images, and in query formulation and evaluation. (4) A system architecture that supports the use of specialized feature extraction algorithms, which may be independently developed in various important application domains, and may rely upon domain-specific image analysis techniques.

To our knowledge, this is the first proposal for the use of a non-trivial data model (coupled with an image description language) for processing large sets of images in a DBMS. We discuss the impact of the data model and the DDL on various aspects of the system, and experimentally demonstrate some major benefits of this approach. In particular, we show how very large image sets can be effectively queried — using meaningful, domain-specific restrictions on the attributes and relationships of objects contained in images — with users providing input only on a per-collection, rather than a per-image, basis. We show that the approach is scalable, and demonstrate that content-based querying of very large collections of images using a domain-independent image DBMS is a viable goal.

1 Introduction

Large collections of images (and videos!) are becoming an increasingly common part of users' data, and the problem of storing and retrieving images by content is becoming a bottleneck in effectively utilizing this information. As is well known from the experience with image processing in the fields of pattern matching, computer vision etc., image analysis and classification are very hard problems. Specialized feature extraction and image understanding algorithms have been developed for specific image domains,

but completely general solutions are not available. Most current image retrieval systems, therefore, are specialized to a given domain of images.

The few general purpose image retrieval systems support relatively weak querying by content (e.g., by color, texture or shape, but with no deeper understanding of the structure of the image). In addition, users are expected to provide considerable input on a per-image basis (e.g., outlining interesting objects), queries cannot really exploit the structure of images, and the underlying storage and indexing techniques do not scale well as the number of images increases.

As the number of images increases, the problem becomes even harder since we must now consider storage on disk or tape and efficient indexing and retrieval; a traditional DBMS problem. Clearly, some new techniques are required to address the latter concerns, but it is important to be able to apply the known techniques for image understanding whenever possible.

These observations suggest that in order to support a rich class of content-based queries over large collections of images, some simplifying restrictions are necessary, and that application specific information must be exploited. From a database perspective, this suggests that:

- The data should be organized into collections of *similarly structured* images. For example, a collection of faces, a collection of airplane profiles, a collection of birds, a collection of satellite photographs of cloud cover, a collection of fingerprints, etc., all exhibit similarity of structure. (Note that we refer here to similarity of the *images to be stored*, rather than similarity of the *underlying objects*. For example, the degree of similarity amongst a collection of frontal "passport-style" photos is much larger than amongst a collection of photos of faces that are randomly oriented.)

- The user should be responsible for choosing an appropriate level of abstraction for the prototypical image in a given collection (e.g., does hair color matter? Does distance between an individual's eyes matter?) and for providing guidance to the system in segmentation, classification and indexing of the images.

- *The user's input should be on a per-collection basis, not on a per-image basis.* For each collection, a user provides a description of a 'protypical image'.

In this paper, we describe mechanisms that enable users to describe a given image set in terms of a *data-model for images*. The DBMS features that support such descriptions are analogous to the *data-definition* commands in a conventional DBMS. Given such a description, the system should be able to to tailor its feature extraction and indexing algorithms for efficient analysis and retrieval of images. Thus, there is a tension in the image DDL design: it must be sufficiently expressive to cover a broad range of image domains, yet simple enough that the system can automatically use a description to tailor some of its basic algorithms.

Our main contribution in this paper is a proposal for a data model and DDL for describing images, and experimental results based on an actual implementation that demonstrate the benefits to be had. We demonstrate that very large image sets can be effectively queried using meaningful, domain-specific restrictions on the attributes and relationships of objects contained in images. The results go significantly beyond what has been achieved in other image retrieval systems in three important areas: *scalability* with respect to the number of images, *expressiveness* of the queries, and *generality* with respect to the domain of images.

To our knowledge, this is the *first* proposal for the use of a non-trivial image data model (coupled with an image description language) for processing large sets of images in a DBMS. The specific DDL that we discuss is an initial design, and we expect that it will be refined as we gain experience working with image sets from a variety of application domains.

We do not discuss details of how to do *color indexing*, or *shape indexing*, or *edge detection*, or other similar problems that are basic to extracting features from individual images. We assume that a library of routines is available for such fundamental tasks, and this is orthogonal to our focus, which is how such "building-block routines" should be used to analyze and organize a large set of images. Indeed, we have designed our system to be *extensible* — if specialized feature extraction algorithms are available of identifying different kinds of objects in a collection of images, the DDL and our system interfaces support the use of such algorithms in a clean way, in addition to the built-in library of basic extraction and image processing algorithms.

It is important to note that we assume the existence of some commonality of logical structure among images in the same collection. Our approach exploits such commonalities in image analysis and in the image representation. We believe that many potential applications (such as *faces* and *fingerprints* for police agencies or *houses* for a real estate agency) have such commonality in their collections. It is our hope that the basic algorithms in the system will support the definition and extraction of most objects of interest, but it is likely, especially in complex image domains (e.g., finger prints) that some domain-specific algorithms will also be required. Even so, as long as there is commonality in the images that is of interest (i.e., the user wants to ask questions about these common aspects), our framework provides valuable benefits: the need for specialized algorithms is reduced, specialized algorithms from different domains can be used together, and the extracted information is cleanly organized, stored and indexed by utilizing the DDL description.

Some examples with no commonality among images can be found (e.g. art collections); we can do little with such data sets. Various techniques (e.g., color indexing, several transforms) can be used to find *similar* objects, and we do support this functionality, as well as storage and indexing of the extracted information, but it is not clear how queries based on the objects contained in the images, and their relationships, can be answered effectively.

This work is part of the PIQ query-by-image-content project at UW-Madison, and the overall effort also includes implementation of other aspects of an image DBMS (e.g., multidimensional indexing), and is being built on top of the Shore storage manager.

1.1 Overview of Our Approach

It is perhaps easiest to present our approach in terms of the steps that we envisage a user will carry out to store and query a collection of images.

- **Data Definition:** The user describes what a typical image is (in a collection of similar images) in terms of *features*. The front-end can make this process more natural, but the key technically is the DDL. DDL definitions give a user the ability to customize the database an application by specifying what aspects of the image are important, what needs to be extracted and stored, and what may need to be extracted (but need not be stored and indexed upon) for certain queries.

- **Data Entry:** The user then loads a collection of images of a defined type, for example a collection of faces. The system analyzes each face, extracts features using the DDL specification of the *face* schema for guidance on what to look for, enters the image into the database, and updates any indexes on the collection of faces.

- **Querying:** The user poses a query, in a suitable front-end, using the collection of features in the definition of the *face* type and some system-supported idioms for concepts like *similar*. The system responds to queries by using indexes to retrieve a minimal number of candidates whenever possible.

- **Post-query Image Processing:** The basic queries supported by the system will typically produce a "first-cut" set of answers (hopefully, a small set). A typical application will then analyze the set of returned images using more sophisticated image analysis.

In the rest of this paper, we discuss several important aspects of our approach in more detail. In Section 2, we present the data model and an exmaple of a first-cut data definition language. In Section 3, we discuss how DDL definitions can be used to guide image analysis. In Section 4, we consider the problem of how to represent image data sets and how to index them. In Section 5, we consider some issues in query formulation and evaluation. In Section 6, we experimentally explore different kinds of similarity queries. In Section 7, we present experimental results on complex content-based queries. We consider the scalability of our approach in Section 8. The previous three sections use synthetically generated data in order to control the complexity of carrying out and analyzing the experiments. In Section 9, we present experimental results on a real dataset to demonstrate that the approach can be applied in meaningful application domains. We discuss related work in Section 10, and in Section 11, we discuss the results and future work.

2 The Data Model and DDL for Images

In this section, we present our data model and DDL for describing a 'prototypical image' in a collection of images. It is natural to ask how the description of a set of images differs from a corresponding abstract data type definition. For example, how does a DDL description of faces differ from an ADT definition of the data type **face**? First, we use the DDL definition to guide feature extraction on an input image. Second, the DDL definition forms the basis for similarity specification in queries. Third, the result of the extraction process typically contains many (implementation specific) attributes that are not directly specified in the DDL description. For example, an attribute of type *shape* might be represented internally using the first several moments of an appropriate transform. Clearly, these points differentiate a DDL definition of a prototype image from an ADT definition.

However, the result of feature extraction on an image can be stored as an ADT object. Whether this is appropriate depends upon how ADTs are implemented in the underlying DBMS; we will not discuss this point further, since it is incidental to this paper.

2.1 The Data Model for Images

For each image in a collection (e.g., a face in a collection of faces), our objective is to be able to automatically generate a *summary* of the image, guided by the description of the 'prototypical image' for the collection. The summaries are stored and indexed, and queries are evaluated based on the summaries; only images whose summaries match the query are retrieved.

The summary of an image has a tree structure. All images in the same collection have summary trees with a similar structure. Each node in the tree corresponds to an *object* in the image. An object in an image (e.g., a nose in a face image) always has a region it corresponds to. Thus, every node in a summary tree has at least one attribute called *primary-region*. In addition, each node may contain several attributes of the following types:

Number: A numeric value that is either a real number or an integer.

String: An array of characters.

Matrix: A matrix of numbers.

Region: A collection of pixels.

Arbitrary: The model does not assume anything about this type. It is used as a mechanism for extending the basic data model for specific uses.

Set: A set of values from one of the above four types.

For example, a simple representation of an airplane may be a tree whose root corresponds to the entire airplane. The primary region of the root would be all the pixels in the image that are part of the airplane. The root may have three childern, the first child for the wings, the second child for the body and the third for the tail. We may represent the wings as a single object or we may represent them as two objects by having two children of the wings node coresponding to the left and right wings. Each node in the tree may have different attributes other than the primary region. For example, the wings' node may have a numeric attribute corresponding to the wing-span. There are a few attributes that are always available for each node because they can be computed from the primary region. These attributes are:

Size: the number of pixels in the primary region.

Position: the center of mass of the primary region (a vector of two components).

Orientation: the angle between the X axis and the diameter of the primary region.

Length: the diameter of the primary region.

Bounding Box: the bounding box of the primary region. This box is parallel to the main axis and is represented by a 2×2 matrix.

We define *classes* for each type of node in the tree. Continuing with the airplanes example, we may have the classes *plane*, *body*, *tail*, *wings* and *wing*. A node of class *plane* has three children from classes *body*, *tail* and *wings*. Nodes of classes *body* and *tail* have no children while a node from class *wings* may have any number of children of class *wing* (airplanes may have more than two wings and some of them may not be seen in the image so the number of wings that are represented may vary).

2.2 Overview of the DDL

A specification of an image is a series of statements in the DDL. We present a textual DDL; in a real system, a graphical user-interface is likely to be much more effective, even if the same underlying functionality is supported. The DDL constructs described in this section are a subset of the constructs that we have currently implemented in the PIQ query-by-image-content system. We expect that the constructs described here will be refined, and additional constructs added, as we gain experience using the system on data sets from different application domains. Nonetheless, the constructs presented here already allow us to demonstrate the major benefits of our approach.

Our DDL for images has several components. Most importantly, we can describe *a class of objects* by defining a prototypical object in the class.

2.3 Class Types and Definitions

The DDL uses *class types* for defining classes. We make a distinction between *simple class types* and *complex class types*. The simple class types that we have implemented in the PIQ system are:

Color: The search algorithm looks for regions with a specific color. The color is defined as an argument of the specific class of this class type.

Uniform Color: The search algorithm looks for regions with a uniform color within some tolerance that is an argument of the specific class of this class type.

Shape: The search algorithm looks for regions that have a specific shape defined as an argument of the specific class of this class type.

Example: The search algorithm looks for regions that are similar to an example region given as an argument of the specific class of this class type. The example region may is another image of much smaller size.

The complex class types in the PIQ system are:

Union: The semantics of a union class type is that the primary region of a node of a class of union class type is the union of the regions of the children of that node. Note that a region is a set of pixels so a union of regions is a union of sets. When we say that class A is a union of classes B, C and B this means that each tree of class A has exactly three children, the first and the third are trees of class B and the second is a tree of class C. The union of the primary regions of the roots of the three trees is the primary region of the root of the tree of class A.

Difference and intersection: The semantics is the same as the union class type but using difference and intersection instead of union.

Set: When we say that class A is a set of class B we mean that a summary tree of class A is a tree in which all the children of the root are trees of class B and the primary region of the root of the tree of A is the union of the primary regions of its children.

Connected-set: The semantics is the same as the *set* class type but a further constraint is that the primary region of the root of a tree is a connected set of pixels.

Or: When we say that class A is B or C or D we mean that the root of a summary tree of class A has one child which belongs to either class B, C or D. The primary region of the root is the primary region of its child.

Each simple class type has an associated *search algorithm* that takes as input an image and specific arguments of the class type and returns a set of summary tree nodes with no children but with some attributes. For example, a *uniform-color* class type has a search algorithm that takes an image and returns a set of nodes that correspond to regions with a constant color in them (within some tolerance level). The algorithms are standard image processing algorithms such as *pixel classification* (for color searches), *Hough transforms* (for shape searches) and *pattern matching* (for example searches).

Similarly, each complex class type has a search algorithm. The result of the search is a set of summary trees and not a set of nodes. For example, a *union* class type will use the search algorithms of each of its chidren as subroutines and will combine their results to obtain a set of summary trees. The search algorithms are described in Section 3.

The DDL has statements for defining classes:

- "class *class-id* belongs to class type *type-id* (*arguments*)"
  This statement defines a new class that can be referred to using *class-id*. The type of this class is *class-type* and the *arguments* are the specific arguments for the search algorithm associated with this

class type. For example, the statement:
"class *round* belongs to class type *shape* (*circle*)"
defined a new class called *round* with a search algorithm that looks for circles. We explain how arguments are used by search algorithms in Section 2.5.

- "class *class-id* is union of classes $C_1, .., C_n$"
  This class belongs to the union class type and its children are classes $C_1, .., C_n$ that were defined earlier. For example:
  "class *plane* is union of classes *body, tail, wings*"

Similar statements with the obvious meaning are:

- "class *class-id* is set of class C"
- "class *class-id* is connected set of class C"
- "class *class-id* is either $C_1, .., C_{n-1}$ or C_n"

2.4 Attributes and Expressions

Every class has the predefined attributes *primary-region, size, position, orientation, length* and *bounding-box*. Classes that have a simple class type may have other predefined attributes. For exmaple, if class A is of class type *shape* it has an attribute called *shape-description* which is a description of the specific shape (the description is a vector of some non-centered moments of the shape). Additional attributes can be defined for a class using the following DDL statement:
"attribute *attr-id* of class *class-id* is *expression*"
This statement means that class *class-id* has an attribute called *attr-id*. The value of *attr-id* can be calculated using *expression*. The expression may use attributes that are already defined for class *class-id* and for any of its children (if it has any).

In addition to being used for calculating attribute values, expressions can be used to restrict the summary trees allowed for a given class using the following DDL statement:
"class *class-id* is restricted by *expression*"
This statement means that any for any summary tree of class *class-id* the given expression must evaluate to "true" given the values of the attributes in the summary tree.

Expressions are represented by expression trees in the PIQ system. The leaves (or *atoms*) of an expression are either attributes or constants. The internal nodes are functions or operations performed on the children. The DDL syntax for expressions is similar to other programming languages (e.g. C or Pascal). For example we can restrict class A by using the statement: "class A is restricted by $A.size < 0.05 * image.size$". This means that the region of class A must be more than 5% of the pixels in the image. The atoms *image.size* and *image.bbox* can be viewed as constants. The operators we have implemented are the usual numeric operators, comparisons (which return a boolean value) and some functions. Additional details about functions are presented in Section 2.5.

2.5 Extending the System

The PIQ system is designed so that new feature extraction algorithms can be inserted to the system with minimal effort. PIQ is under constant development and such a design makes it easy extend the system. More importantly, many specialized algorithms for feature extraction have been reported in the image

processing literature. Some of them are for specific domains and others are more general. We want to make it possible for users to utilize such algorithms with minimal effort.

The set of available search algorithms can be extended in PIQ through the mechanism of defining new simple class types. A new class type must be *registered* by giving the system its name, a list of predefined attribute names with their types and the source code for the following three functions:

Search: This is the feature extraction algorithm. Its input is an image, and the arguments for the class. It must return values for all the predefined attributes that were listed in the registration process and the primary region for the node (of the given class in the given image).

Argument-input: This function is used for creating arguments for a specific class of the registered class type. For example, if the *shape* class type were not built into PIQ, we might want to add it. If so, this function would create the arguments for a specific shape class (e.g., circles). The input to this function is ...

Cost-estimate: This function will estimate the cost of running the feature extraction algorithm given specific values for the arguments and the image. It is used by the search optimizer (see Section 3).

PIQ also supports the addition of functions to be used in expressions. One can register a new function by specifying the function name, the return type, the names and types of the arguments and the source code for evaluating the function given constant values for its arguments. The PIQ extensibility features are discussed further in Section 9.1.

2.6 Using the Summary Trees

It may not be practical to save all the attributes of a summary tree in the database since the size of a summary tree (including attribute values) may be very large. Further, some attributes may be used for computing other attributes of interest, but may not be of interest themselves. The DDL therefore includes constructs to specify which attributes should be saved in the database. It also contains constructs for specifying which attributes should be indexed upon. We evaluate queries using only the saved attributes and specified indexes. Index creation is very similar to index creation in a conventional DBMS, and should be guided by queries that users expect will be asked frequently.

3 The Image Analysis Algorithm

A major challenge is to utilize information in the DDL description of a prototypical image while processing a collection of images. Our approach consists of interleaving feature extraction with constraint simplification (over constraints derived from the DDL description and prior steps). The feature extraction algorithm exploits the information in DDL definitions to guide the steps in the extraction, thereby making it potentially much more efficient than a "blind" feature extraction. It also produces a detailed summary of the image in the form of a tree of objects that "instantiates" the DDL definition, and this summary serves as the basis for expressing sophisticated queries about the content of an image. This algorithm is one of our main contributions; our experimental results demonstrate that it is indeed very effective in terms of both the quality of extraction and the utility of the resulting summary trees.

The feature extraction is based upon a library of fairly conventional image processing routines, which can be enriched further. The library that we have implemented in PIQ includes support for tasks like *uniform color searching, shape searching, searching for example images* etc.

We utilize a standard package (based on Simplex) for constraint simplification. We use it in a way that gives us bounding intervals for some of the variables. This is very useful for restricting size, position and other attributes so that image processing searches become less costly.

3.1 Problem Definition

We are given as input a class description in the DDL and a set of images. The output should be one instantiation of the class for each image in the set. Consequently we need an algorithm that given a class definition and an image will find an instantiation if one exists.

We can think of the class definition as a tree whose leaves are simple objects, each corresponding to a search method. The internal nodes correpond to operations that can be done over the image regions that are found for the leaves. An instantiation is another kind of tree, with leaves that correspond to image regions found for the class definition tree, and internal nodes that correspond to the (image regions found to match the) internal nodes of the class definition tree. The shape of the instantiation tree can be different from the shape of the class definition tree. For example, take the definition: "**class** A **is** either B or C". The root of an instantiation tree of class A would have a single child that corresponds either to an instantiation of class B or of class C. As another example, consider the definition: "**class** A **is set of class** B ". If we find three instances of class B in the image then the root of the instantiation tree of class A would have three children corresponding to these three instances.

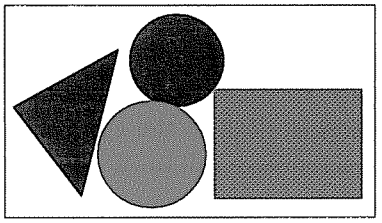

Figure 1: Example of image

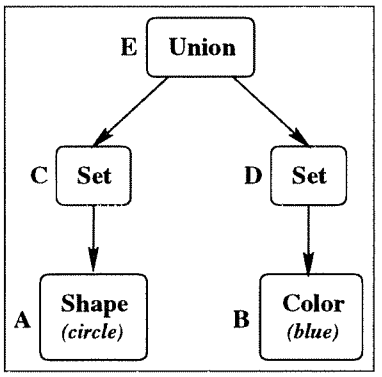

Figure 2: Example of class

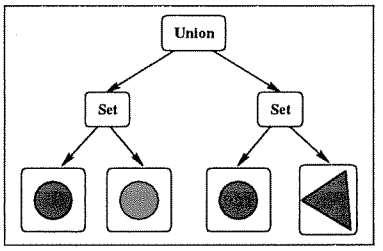

Figure 3: Example of instance

The goal of the analysis algorithm is to find a valid instantiation, given an image and a class definition. Figure 1 shows an example of a very simple image. In Figure 2 we show an example class definition without any attributes and restrictions. Figure 3 shows a valid instantiation of this class for the image in Figure 1.

A textual description of the class definition in Figure 2 is:

- **class** A **belongs to class type** *shape(circle)*
- **class** B **belongs to class type** *color(blue)*
- **class** C **is set of class** A
- **class** D **is set of class** B
- **class** E **is union of classes** $C.D$

Class E is instantiated in Figure 3.

3.2 Main Algorithm

The algorithm maintains a schedule of *tasks*. Each task is a partial solution to the problem. A task is a tree structure corresponding to the class definition tree, where some of the leaves are instantiated to regions in the image. Every internal node of a task whose children are (all) instantiated is instantiated as well. An iteration of the algorithm consists of selecting a task from the schedule, and searching the image for instances of one leaf (of that task tree) that is not already instantiated. (Note that the search parameters are typically constrained by previously instantiated leaves.) For each such instance of the leaf, the algorithm creates a new task tree by copying the original task tree and instantiating the leaf node to the newly found instance.

The new task trees are analyzed to see if they meet all the restrictions in the class definition, and those that do not contradict the restrictions are inserted into the schedule (if they are not fully instantiated) or into a list of final results (if they are fully instantiated). The analysis stage following each iteration evaluates all attributes of instantiated nodes. This means that a fully instantiated tree has all its attributes evaluated.

The algorithm starts with exactly one task that is a tree with no instantiated nodes. The algorithm is done when the schedule is empty. Note that any number of results may be returned by the algorithm.

3.2.1 Simple Class Definitions

Suppose the class definition does not have any **set** or **or** nodes. For each "simple" leaf, the algorithm essentially calls the search routine of the class type to which the leaf belongs and that routine searches the image for instances of the leaf.

3.2.2 Complex Class Definitions

When the class definition has **set** or **or** nodes, the algorithm must collect the results for each instantiated child, rather than create new tasks. We use the algorithm recursively, treating **set** nodes and **or** nodes as if they were leaves. Whereas the search algorithm for "simple" leaves is a call to the appropriate feature extractino routine the search algorithm for the "complex" leaves is a recursive call to the search algorithm itself. For example ,suppose we have the following class definition:

> **class** *Foo* **is union of classes** *Reds*, *Blue*.
> **class** *Reds* **is connected set of class** *Red*.

And *Red*, *Blue* are simple classes with red and blue colors respectively. For the purposes of extraction, we consider this definition as two separate trees. The lower tree has *Reds* as the root and one leaf : *Red*. The upper tree has *Foo* as the root, and it has the lower tree and *Blue* as "leaves". The extraction algorithm performs the following steps:

1. Choose a leaf of the upper tree for instantiation. Suppose the lower tree is chosen (as a leaf).
2. Search for instantiations of the lower tree. This is a recursive call to the extraction algorithm with the lower tree as a class definition.
3. Take all the results from the instantiation and create partially instantiated trees from them. Each such tree would be a copy of the upper tree with the lower tree leaf replaced by one of the extracted instances.

4. Evaluate attributes and restrictions for each partial result. Put all the successful results in the schedule of tasks.

5. For each result in the schedule do:

   (a) Search for instantiations of the *Blue* leaf.

   (b) For each instance that is found create a copy of the tree while replacing the *Blue* leaf with the instance.

   (c) Evaluate attributes and restrictions and report all successful results as results of the entire extraction.

3.2.3 Scheduling a Search

An important issue is how the schedule of tasks is managed, and how the order in which tasks are considered can be optimized to minimize the overall extraction effort. The schedule is simply a priority queue, and priorities are assigned based on considerations such as the following:

How low is the time required for the task? It is somewhat hard to estimate the cost of a task since it involves both one search for a simple node and then the analysis of the results. The one search is estimable but the analysis depends on the size of the result which is usually unknown in advance.

How restrictive is the time required for the next instantiation? Determining appropriate scheduling criteria is one of the open problems in this work, and we have thus far used simple heuristics. We also want to know how restrictive it would be to instantiate a leaf. The naive metric would be the change in the size of the schedule as a result of the instantiation. This naive metric could be very useful because the size of the schedule is a good measure of how much work needs to be done in the next iteration. Some simple guidelines can be applied here. It is obvious that if we know the color of a uniform colored object then there are fewer possible instantiations than in the case where we don't know the color.

In general, the more restrictive the search space (position, orientation, scale, color etc.), the fewer the instantiations we **expect** to find in the image. Another point is that the more restrictions we have involving the attributes of the leaf with attributes of other nodes, the more we **expect** that an instantiation would restrict the possibilities for those other nodes. In this sense we can construct some sort of **expectation** metric for how restrictive an instantiation of a leaf can be. Therefore, we need to avoid searching for independent leaves as much as possible.

4 Representation and Indexing of Image Sets

The feature extraction phase produces an instantiated clas-definition tree for each image. We now consider how this information can be stored and indexed in order to support complex content-based queries efficiently.

In a relational DBMS, a popular indexing technique is the B+ Tree, and a typical index retrieval is "Find all *Employee* records where *salary* is between 40K and 50K". The user who created the *Employee* relation must anticipate such queries and create a B+ Tree index on the *salary* field. By analogy, we want to be able to create indexes on a set of images. Typical queries might be "Find all images similar to this image", "Find all airplanes (from a collection of airplanes) where the ratio of wingspan to length is less than 1", "Find all images that contain a spherical object", etc.

While the analogy to B+ Trees is useful—it suggests that the user must provide input on what features to index, for example—the problem is clearly much more difficult. Typical query conditions are now based

on some notion of *similarity*, not precise arithmetic comparisons. How can we build access methods that can support retrievals based on such complex conditions? A natural idea is to break the problem into two parts: first, techniques for extracting sets of "feature vectors" from a given image, and second, techniques for storing and retrieving sets of such vectors. Intuitively, each vector of k dimensions should capture the essence of an image object in such a way that two nearby vectors in k-dimensional space correspond to similar objects. Along with each vector, we store a pointer to the corresponding image and object.

This idea (with only one vector per image) was proposed by Jagadish [14], and is very promising. Our approach can be seen as a generalization of this idea in which an image is used to generate several vectors. There are several relations associated with each class definition, and each vector generated from an image in this class is stored as a tuple in one of those relations, which we call *summary relations*. Clearly, summary relations are likely to have many (at least in the tens, possibly more) attributes.

4.1 Generating Vectors from Images

We now examine how *summary relations* are generated. This is based upon the techniques for feature extraction discussed in Section 3, and is guided by the DDL description of a given collection of images. The main idea is to have a relation corresponding to each node in a class definition tree and a tuple for each instantiation of a node in the appropriate relation for that node. The attributes of a node are the attributes of its relation, some identifying attributes and pointers to its children (if any).

The identifying attributes consist of *image-id* and *object-id*. We need the identifying attributes to be a key so that we can refer to a specific tuple in the relation. Thus, we implement a pointer as a foreign key. We get *object-id* from a numbering of all tuples of this relation for a specific image, so together with the *image-id* it becomes a key. For example, suppose that we have a class *Reds* that is a set of class *Red*. Class *Red* is a simple class defined by the color red. Suppose that for class *Red* we define an attribute *size* and for class *Reds* we define attributes *length*, *position*. The relations corresponding to the classes would be:

- *Reds*(*iid* : image-id, *oid* : number, *children* : set-of-number, *length* : number, *position* : [number,number]).
- *Red*(*iid* : image-id, *oid* : number, *size* : number).

Suppose we have an instantiation of this class for image I with three instances of *Red* then the relations would have tuples :

$$Reds(I, 1, \{1, 2, 3\}, 57, [30, 60]) \qquad Red(I, 1, 20) \qquad Red(I, 2, 10) \qquad Red(I, 3, 100)$$

The attribute values here are chosen at random. The point to note is that the combination of image-id and object-id creates a key for each relation so we know these three tuples for *Red* are indeed the children of the *Reds* tuple.

All attribute types except *region* and *arbitrary* can be represented directly in any of the relations. If we do choose to save a region attribute we save the region separately as an object and use its object ID as a relation's attribute. We can do the same with attributes of *arbitrary* type.

In order to capture the notion of similarity, several ideas have been proposed. Although many sophisticated techniques have been developed for similarity measurement [24], no attempts have been made to experimentally evaluate their efficacy in image databases. A very interesting research direction is to explore the use of DDL descriptions to define similarity. We return to this point in Section 5.

4.2 Multi-Dimensional Access Methods

An important component of our approach is *multi-dimensional indexing*. The vectors generated by analysing an image must be stored, and *range queries* must be answered efficiently by retrieving only qualifying vectors. One important direction of our research will be to investigate how to improve current multidimensional indexing techniques, especially for data sets with high dimensions. The design of multi-dimensional access methods is outside the scope of this paper, and we only discuss the potential use of such methods (which we have implemented in PIQ, and are currently evaluating). The fact that our framework allows us to use such methods on the summary information produced by the feature extraction phase is very important, since even the summary information is very large. When the class definitions are complex and involve color histograms and shape summaries the entire summary for a single image may involve dozens or even hundreds of numbers. As the number of images increases, scalability demands that efficient indexing techniques should be applicable.

Typically, users want to ask several different queries on a given image set. A single index may be able to support several kinds of queries, but it is likely that in general we will need to build multiple indexes on a given image set. For example, on a collection of faces, we may want to support queries like "Find all people with eyes similar to these", "Find all people with dark hair", "Find all people with square faces or long necks".

A first step in supporting such queries is to develop DDL descriptions of the images that direct the system to extract information about the aspects of interest to the user (e.g., the color of the hair, the length of the neck). The next step is to recognize that in fact there may be several vectors derived from a single image, each capturing some features of interest. This means we create several indexes on the same set of images. The problem of how we choose a suitable collection is an important one. Users are expected to provide this information, but it is clearly necessary to develop good guidelines first, and to understand the issue in the context of the DDL definitions that generate the "summary" relations that are to be indexed.

5 Query Formulation and Evaluation

After the object extraction phase is complete the database is populated with one instantiation for each image in a specific class. These instantiations take the form of trees which are represented by several relations in the way discussed in Section 4.1. A query consists of operations on the extracted instantiations, i.e., the summary relations.

Suppose we have a class *Airplanes* which is defined as a set of class *Plane*. *Plane* would be a complex class, but for this example we can ignore the details of how *Plane* is defined in terms of other classes. We can assume that we have attributes *position, orientation, size, length, shape* and *color* for class *Plane*. An instance of class *Airplanes* would be a tree whose root is a node with no attributes and whose children are instances of class *Plane* (exactly all the instances that are found in the image). Now we can ask queries of the following kinds:

- Find all images of class *Airplanes* in which there are two planes pointing to the same direction (i.e., having the same orientation).
- Find all images of class *Airplanes* in which there is a black plane seen above a white and red plane.
- Find all images of class *Airplanes* in which the largest plane is white.

We would not be able to ask a query of the form: "Find all images of class *Airplanes* in which a black airplane is seen on a blue background" since the class definition does not contain any reference to the

backgroud color distribution.

We see that having semantic information about the content of the image can help to formulate queries that are beyond the scope of any "pixel-level similarity" query. On the other hand, we can always form a query of the form "Find all images of class *Airplanes* that are similar to image I" by first extracting an instance of the class definition *Airplanes* from image I and then comparing the resulting instance to all other instantiations of that class. The only problem is the meaning of similarity. If the user does not wish to specify the notion of similarity then the system can use some (necessarily) ad-hoc decisions for comparing instantiations of the same class. On the other hand a user may wish to elaborate. For example, the user can state that the number of planes in the image has the most importance or that the color of planes has more importance than the number of planes, etc. We also want the result of the query to be an ordered list of results where the order comes from the "amount" of similarity found.

In contrast to other current image retrieval systems, the power of our query facility is evident. We can utilize spatial relationships among objects, for example, not just colors and shapes. We can define any notion of similarity that we want as long as the information that we use is included in the extracted attributes.

6 Similarity Queries

In this section, we present the results of a series of experiments that explore the concept of similarity between images. The purpose of the similarity experiments is to test whether a similarity query based on some similarity notion that is internal to the system can generally yield acceptable results. The alternative is to allow for user-defined similarity queries so that a user has more control over what kind of images should be retrieved.

The *data model* that the system uses to represent information about images plays a key role in the kinds of queries that can be supported. We have designed experiments with the purpose of evaluating the power of using a rich data model. We use one set of images for all the similarity queries. Each experiment consists of two stages:

Extraction: The first part is to extract instantiations for a specific class from a set of images. We note that this part required *no* user intervention.

Queries: We ask similarity queries by choosing an image from the set and retrieving the most similar images, using a given notion of similarity based on the image data model, from the set of images.

These experiments are done in a very controlled setting. The images are synthetic and rather simple. The data set is small (1000 images) so each query can be manually verified. The images are randomly generated. Each image consists of between 1 to 8 geomtric figures (circles, squares, triangles), with random colors and a dark background. We included three small sets of images in these 1000 images. One set consists of images that have a single white circle in them. The second set consists of images with two adjacent circles (one white and one red). The third set consists of images that have 3 disjoint squares in them (red, white and blue). These sets are used as controls for the similarity queries. We make the assumption that the triangles in the images are ignored. This assumption is made so we can stress the point that in general only parts of the images are important to us and maintaining information about the rest of the image is not only unnecessary but also a waste of resources.

After the 1000 images were generated we defined 3 different class definitions (using the DDL) that correspond to different "image data models". For each such class definition we extracted instantiations for the images and we asked the same similarity queries. The notion of similarity differs from class to class but

the general principle is to try to incorporate all the information available in the underlying DDL definition while computing similarity. The images for the similarity queries were chosen from the three small control groups, so it is easy to verify how good the answers are.

In the following sections we present a description of the experiments for each data model and an analysis of some of the results. Section 6.1 uses the data model that comes with the PIQ system. In contrast we tried two other data models that are similar to what is available in the QBIC [1] system. These are discussed in Sections 6.2, 6.3 We show the results of the same similarity query for each of the data models. Section 6.4 describes two different similarity metrics that also use the PIQ data model. Its purpose is to show the advantage of being able to change the similarity metric, even with the same underlying data model. (We note that we carried out additional experiments using different similarity queries and different data sets, with very similar results.)

6.1 Class Definition Using the Full PIQ Data Model

The class definition uses the full data model supported by the PIQ system, given the following DDL definition:

class *Blocks* **is set of class** *Block*.
class *Block* **is connected set of class** *Shape*.
class *Shape* **is either** *Circle* **or** *Square*.

Circle and *Square* are simple classes corresponding to the appropriate shapes. The attributes *size, length, position, orientation, color* and *shape* are extracted for all classes. For the simple classes, we omit the *shape* attribute since it is clear from the class definition. The intuition here is that we get all circles and squares in the image, group them into connected regions (called blocks) and retrieve information about each circle, square and block. A tree description of this class definition can be seen in Figure 4.

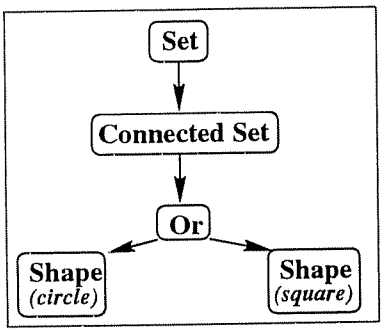

Figure 4: Class definition for the similarity queries with the full PIQ data model.

After the extraction is finished we have a set of instantiations that can be viewed as a set of trees. The notion of similarity here has two major parts. The first part tries to match two trees in terms of the type of objects they contain. This means that two trees match only if they have the same number of *Blocks* (or instances correponding to class *Block*). There must be a 1-1 matching function between blocks of one image and blocks of the other. A *Block* of two circles can only match another *Block* of two circles, etc. After a match is found we create a similarity metric that for each attribute matches it with a corresponding attribute in the other image and we incorporate all these matches into one number. This is a somewhat ad-hoc way of defining similarity but as will be seen later, there may not be a good way of defining similarity anyway.

The results of the similarity query for the full PIQ data model are shown in Figure 10. In all queries the results are given in sorted order. The order in the figures is first from left to right and then from top to bottom, so the best result is in the upper left corner and the worst (among those that are shown) is in the bottom right corner. Note that the best result is always the image used for the query, so all images in a figure should be compared to the one in the upper left corner of the same figure. The similarity query involves an image with two adjacent circles in it and no other objects, Figure 10 shows the best 16 results (out of 28) for that query. For the PIQ data model, the definition of

similarity takes into account *size, position* and *color* for this query and all of them carry about the same weight.

Since the query image has two adjacent circles in it then all the results have exactly two adjacent circles and no squares. Even in this simple case it is already difficult to see why the images are placed in that order. There are a lot of attributes to consider, such as the color distribution between the two circles, the orientation of the two circles as a single object, etc. On the other hand, it is hard to know what the user would consider to be "similar" in this case. Should the position of the objects matter? Should the fact that the white circle is above the red one matter? It is clearly desirable to find some notion of similarity that an "average user" may agree upon, but it is equally important to allow users to specify what they mean by "similar" when they need to do so.

6.2 Class Definition with Only Color

The DDL class definition here is influenced by the data model of the QBIC [1] system. In the QBIC system all objects have the same type and the features that can be extracted for an object are limited to *color, shape* and *texture*. There is also a *sketch* attribute for entire images. The other limitation is that a user must define the objects in the image for the system. We defined two data models that correspond to the QBIC data model. This section deals with the case where no user is available to assist in feature extraction. (Section 6.3 deals with the case where a user is willing to assist in feature extraction.) In this case we only do feature extraction for the entire image. We do not have texture in our data set and shape is irrelevant for entire images. Therefore the class definition has one simple class that defines the entire image and extracts only a color attribute (which is a color histogram of the image).

The only similarity metric we can use is the comparison of color histograms since that is the only available data. We call this class definition "Control 1".

The results of the similarity query for the "Control 1" data model are shown in Figure 11. The figure shows the best 20 results out of 1000. The reason that there are 1000 results is that we have no data that allows us to exclude any image in the database from being similar (to some degree) so a similarity query for this data model just creates an ordering of the entire database. Again, the order in which the results are retrieved is from left to right and from top to bottom. Some results are grouped together by a blue outline around the group. The order in each such group is random since the query returned the same numerical similarity value for all images in that group. For example, in Figure 11 the groups are: $\{2,3\}, \{4,5,6,7,8\}, \{9,10\}.\{14,15,16\}$ and all the other results are not grouped. Within each group the similarity metric returned the same value. Note that the similarity metric compares only the color of the entire image and the background is therefore taken into consideration. That is why an image that contains squares instead of circles can be retrieved as the 11th result. We can see that some images that do not appear to have a lot in common with the query image are retrieved (e.g., the 11th, 13th, 19th and 20th results). The fact that some of these images have colors in them that do not appear in the query image is not an error. The similarity takes into account the fact that the amount of the other colors is low. This query stresses the point that a general purpose similarity metric cannot always work satisfactorily.

6.3 Class Definition with Color and Objects

The class definition here tries to mimic the QBIC data model assuming that there is a user willing to identify objects in each image. In this way, we can identify circles and squares and extract their color. (We assume that the user knows that triangles are not important.) The class definition is:

class *Objects* **is set of class** *Object*.
class *Object* **is either** *Circle* **or** *Square*.

We only extract the color of cirlces and squares, and additionally record the fact that they are circles and squares. The notion of similarity here is as follows: first match the number of circles and squares in both images. If they are the same, then compute similarity based on matching the colors of circles to circles and of squares to squares. We call this class "Control 2".

The result of the similarity query for the "Control 2" data model are shown in Figure 12. The Figure shows the best 30 results (out of 70). The similarity metric in this case causes the system to retrieve images with two circles in them without checking the adjacency of the circles. That is why results 26-29 are retrieved although the circles in them are not adjacent. Note that in this case the background color is not relevant since it is not represented in the data.

6.4 Other Similarity Metrics for the PIQ Data Model

The similarity metric that we used for the PIQ data model (Section 6.1) was ad-hoc. We tried two other similarity metrics in order to contrast them with the result for that ad-hoc similarity metric, and the "Control 2" data model similarity metric.

The first metric we used ignores all attributes of objects except *color*. We call this the "Limited PIQ" similarity since it limits itself to a small part of the available data. This similarity metric first compares two instances to see if they have the same form (e.g., if they have two adjacent circles or only a single circle, etc.) If the two instances have the same form it tries to match colors of all objects. This includes colors of complex objects (e.g., the union of two circles).

The results of the similarity query using the Limited PIQ similarity metric are given in Figure 13. The figure shows the best 27 out of 28 results. We clearly see that only images with two adjacent circles are retrieved. The degree of similarity among the first 25 results depends only on the color distribution in the union of the circles (i.e., the ratio between the area of the red circle and the area of the white circle).

The second metric we used ignores even more data. It ignores all attributes except the color of simple objects, and also ignores adjacency of objects. Computing similarity involves counting the numebr of circles and squares first, and if the numbers match, then we try to match colors of simple objects. This similarity metric uses the exact same type of data that the "Control 2" similarity metric uses. The results for all queries with this similarity metric are identical to the "Control 2" queries so they are not shown in figures. The purpose of this similarity metric was to show that we can define essentially the same similarity metrics on different data models, and that automatic matching of the class definition tree of objects is not a required step in similarity queries in PIQ.

7 Complex Content-Based Queries

In section 6 we saw how a data model can help perform similarity queries. The full power of the data model is best demostrated by posing queries that can **not** be posed to a system that does not support a rich data model. We call such queries "Image-Specific Content-Based Queries", to emphasize that these queries specify conditions that depend upon the given image or class of images, and not just conditions (such as color) that are common to all images. The following sections demonstrate the power of the PIQ data model by posing such queries.

7.1 Content Based Queries for the Simple Data Set

To demonstrate the power of the PIQ data model we took the same image set discussed in Section 6 and used the instances obtained by the extraction algorithm for the PIQ data model. Once the database is populated with those instances, we posed three queries and the results for these queries are shown in Figures 14,15,16 . We verified all those results by doing a manual search in the database (all 1000 images) and the results for all three queries were accurate.

The first query was: "Find all images in which there are circles and squares and in which all the circles are to the left of all the squares". We imposed a constraint that the order in which the results are retrieved is a decreasing order of the width of the horizontal gap between the set of circles and the set of squares. Figure 14 shows the best 12 out of 75 results.

The second query was: "Find all images in which there is a pair of adjacent circles such that the ratio between the size of the large one and the size of the small one (of the same pair) is at least 3". We imposed a constraint that the order in which the results are retrieved is a decreasing order of that ratio. Figure 15 shows the best 12 out of 24 results. Due to the dithering of the images during printing some colors that were very similar are blended into a single color. In the 5th result there is a small bright circle inside the yellow circle, so that result is not an error. This 5th result demonstrated to us that a user may not be always able to identify the objects. We needed to look at a magnified version of the image so we could see that the bright circle is there. For certain real image sets (e.g., in the medical community) automated feature extraction system are known to be more reliable than a human and not just more efficient. Being able to perform such sophisticated feature extraction in a general-purpose image DBMS is a challenge (and perhaps unrealistic to expect to achieve), but nonetheless, there is a significant advantage to doing automatic extraction whenever possible, from the standpoint of reliability as well as efficiency.

The third query was: "Find all images in which there are circles and squares and in which all circles are smaller than all the squares". We imposed a constraint that the order in which the results are retrieved is a decreasing order of the difference between the size of the smallest square and the size of the largest circle. Figure 16 shows the best 12 out of 68 results. Note that in the 9th result there is a large dark circle that is partially occluded by the green square. The search algorithm that we use allows for some occlusion of objects, but we set a limit to the amount of occlusion that is allowed in a retrieved object and this circle did not meet that standard. Therefore, from the system's point of view this image constains only one circle and one square.

All these queries demonstrate the use of relationships between objects. To the best of our knowledge none of these queries can be posed to current general-purpose image retrieval systems.

7.2 A More Complex Data Set

All the examples shown in Sections 6,7.1 were designed to be as simple as possible, in order to make the results easy to analyze. We designed another experiment that still uses a synthetic data set, but a more complex one, and therefore the class definition and the queries can also be more complex. The data set contains 1000 images in which there are two squares, one blue triangle and several circles. The position, size and color of the circles is random, the position and size of the triangle is random, the color and size of the squares is random, and the position of the squares has some random elements in it. As in the previous experiments we first defined a class, then automatically extracted instances for each image, and finally posed queries that use the information in the instances.

The class definition is as follows:

Class *Complex* **is union of classes** *Left-Square, Right-Square, Blue-Triangle, Circles.*

Class *Circles* **is set of class** *Circle.*

Classes *Left-Square* and *Right-Square* are simple classes that have the same definition (square shape). Class *Blue-Triangle* is a simple class defined as a blue object that is restricted to have a shape similar to a triangle. Class *Circle* is a simple class defined by the circle shape. We imposed some constraints on the main class. One constraint requires that *Left-Square* be to the left of *Right-Square*. The other constraint requires that *Left-Square* and *Right-Square* be at about the same height. (The actual constraint was that they could not be separated by a horizontal line.)

We extracted the following attributes for each class: *size, shape, color, position*, etc., as usual, and also defined three special attributes using the DDL :

attribute *A* **of class** *Complex* **is** *distance(Left-Square.prim-reg, Blue-Triangle.prim-reg).*
attribute *B* **of class** *Complex* **is** *distance(Right-Square.prim-reg, Blue-Triangle.prim-reg).*
attribute *C* **of class** *Complex* **is** *distance(Circles.prim-reg, Blue-Triangle.prim-reg).*

The next step was to pose queries and check the results. The first query was: "Find all images in which there are at least three circles, all the circles are below the squares and the blue triangle is closer to a square than to any of the circles". The results are in increasing order of the difference between the distance of the triangle to the nearest circle and the distance of the triangle to the nearest square. All eleven results are shown in Figure 17.

The second query was the same as the first but with a different ordering. The results are in increasing order of the distance of the triangle to the nearest square. Figure 18 shows all eleven results.

The third query was: "Find all images in which the square that is closest to the blue triangle is green, the other square is not green and there is at least one red or orange circle". The order is a descending order on an approximation of the number and brightness of red and orange circles. (A bright red circle scores three times as much as any other red or orange circle.) With color histograms, colors and brightness can be roughly approximated but not accurately evaluated. Figure 19 shows the best 12 results (out of 25 results).

Note that the data model helped us to automatically extract the data from the images. From 1000 images in the database, 444 images did not meet the standard set by the DDL so the database was populated with 556 instances of *Complex* objects. Using a DDL can help us determine which images should belong to a specific class and which should not. Once we have populated the database, we can pose complex queries that use all that information to further search the extracted instances.

8 PIQ Scalability Testing

An important quality of a DBMS is scalability. The system should be able to handle growth in dataset size gracefully. We tested the scalability of the PIQ system with several experiments that determine some of the costs involved in populating and querying a large database.

8.1 A Much Larger Database

The first scalability experiment involves populating the database with 50,000 images, automatically extracting instances for a specific class definition, and then posing content-based queries. The images were generated in the same way as the data set discussed in Section 6 The class definition we used for this experiment is :

Class *Blocks* **is set of** *Block*.
Class *Block* **is connected set of** *Not-Black*.

Where *Not-Black* is a simple class defined as a uniform patch of color that is not dark.

An example query for this experiment is: "Find all images in which the largest block of objects is mostly blue". Results are in increasing order by the number of objects in the image and then in increasing order by the size of the blue object. The best 16 results (out of 498) are shown in Figure 5. This query demonstrates that the PIQ system can handle a large data set and pose real content-based queries against it.

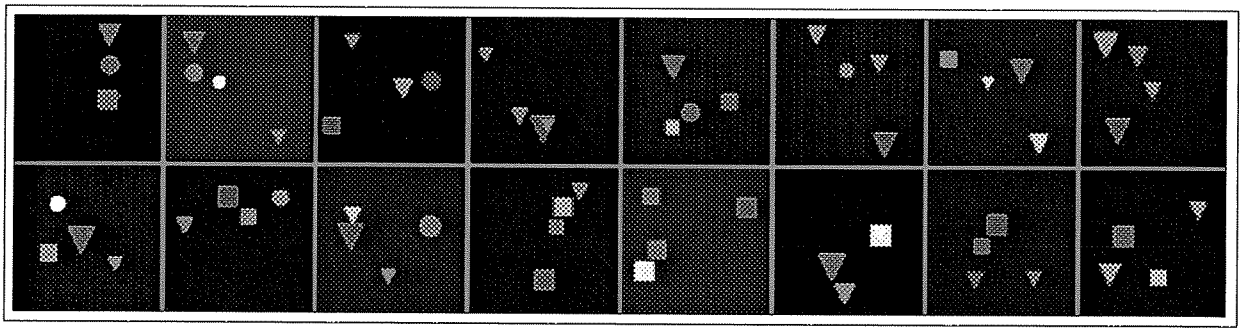

Figure 5: Result of query: Find all images in which the largest object is blue. Order the results first by the number of objects in the image and then by the size of the blue object. Best 16 results out of 498 results.

8.2 Time Cost Analysis

An important factor that we tested is the time that it takes to extract features from the images. If we consider a very homogeneous image set we might expect that the extraction time would be linear in the number of images. To test this we measured the extraction time for sets of 1000 images , 10000 images and all the 50000 images. The results are shown in figure 6, and clearly show that the extraction cost scales linearly.

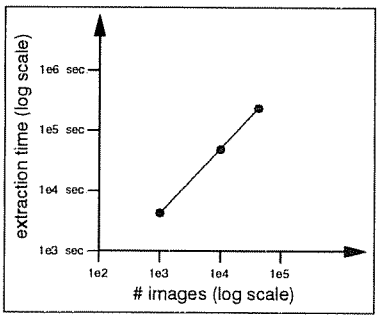

Figure 6: Time of feature extraction as a function of the number of images for the large data set.

To further test this hypothesis we measured the times it took to do feature extraction for each image separately (for 1000 images) and took the standard estimates for mean and variance. The result has estimated mean 3.95 seconds and variance 0.463 seconds, confirming that the time taken for extraction is likely to grow linearly with the number of such (uniform) images.

A DDL description of an image tells us what should be ignored as much as what should be extracted. Suppose that the class definition becomes more restrictive in the sense that it ignores more parts of the image. Would the cost of extraction be lower? To answer that, we take the small data set discussed in Section 6 and measure the average extraction time if we are interested in only circles, only squares or both circles and squares. The result is:

image class	only circles	only squares	both circles and squares
average extraction time (sec)	6.44	7.45	12.87

This clearly shows that the time cost of extracting both circles and squares together is about the sum of the cost of extracting circles alone and squares alone. (It is slightly lower than the sum since some image analysis routines such as edge-detection are run once for both circles and squares.)

9 Using Images from a Real Data Set

In order to test the feasibility of using the PIQ system on meaningful problems, we carried out experiments with a real data set. The set contains fifty images taken from the NASA Dryden Research Aircraft Photo Archive. [1] All the images contain one or more aircraft that can be extracted from the image using an analysis of foreground and background in the images.

We defined a class for those images as follows:

class *Airplanes* **is set of class** *Plane*.

Class *Plane* is defined as a connected set of simple classes that describe uniform color patches in the image with various shape, size and color constraints. The class definition is not intuitive, but it basically performs an extraction of the foreground objects from the background and then uses restrictions on the obtained instances to exclude things that obviously do not look like an airplane. For example, Figure 8 shows an aircraft and the moon. The moon is not extracted as an airplane (although it is indeed extracted as a foreground object) because its shape does not fit the class definition. Figure 7 shows an image for which the extraction failed. The failure is due to the busy background; also, a part of the airplane has similar colors to some background objects. This image was not included in the set we posed queries on.

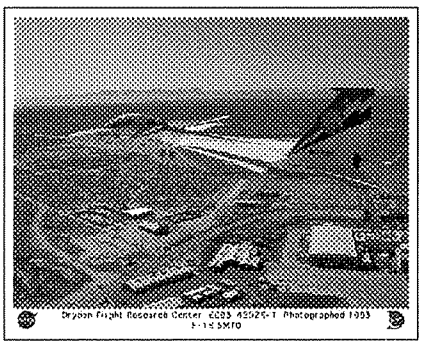

Figure 7: An airplane image not from the queried data set

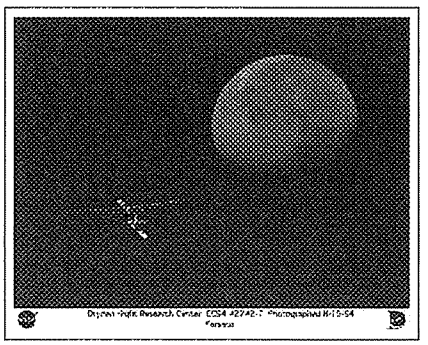

Figure 8: An airplane image from the queried data set

We defined attributes for the *Plane* class that include *color*, *shape*, *length*, *orientation*, *size* and *position*. After the feature extraction was completed we posed image-specific content-based queries and similarity queries.

The first content-based query was: "Find all images with two airplanes in them and order them by how much the top airplane is black". The four results for this query are shown in Figure 20. The first three results had a very small difference the retrieved "blackness" value but the fourth result had a significantly lower value.

The second content-based query was: "Find all images with at least two airplanes and retrieve in increasing order of the size of the bottom airplane". The five results are shown in Figure 21. Notice that

[1] The NASA Dryden Research Aircraft Photo Archive is accessible on the WWW at http://www.dfrf.nasa.gov/PhotoServer/photoServer.html

the first result has three airplanes in it, which were all extracted from the image so it couldn't appear as a result of the first query.

We defined a notion of similarity that incorporates only shape, color and size of airplanes. Then we posed similarity queries. An example is shown in Figure 22. The query image is the SR-71 airplane in the upper left corner of the figure. It seems that the first few results are very close to the query image and from the 7th image onward it becomes less clear that the result is similar. When we see as a contrast the worst five results (in Figure 23) it is clear that this query was successful. Incidentally, note that images 12 and 13 are identical. The archive happened to contain two copies of the same image. We noticed this when we saw the result of the query!

9.1 Using Specialized Feature Extraction Algorithms

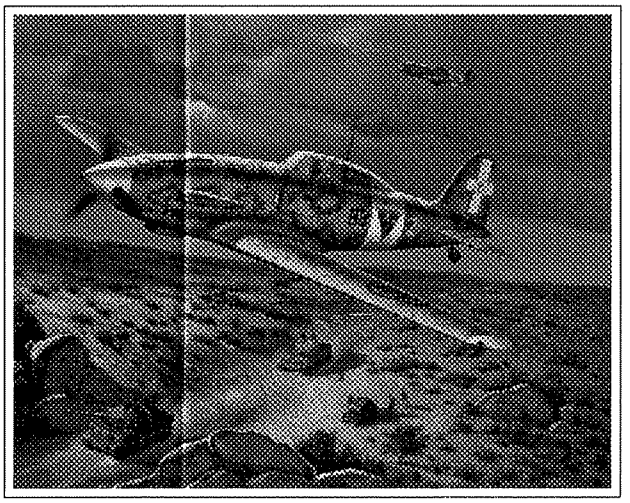

Figure 9: An example picture of military vehicles and airplanes.

Consider the problem of dealing with a large collection of images containing tanks and other military land vehicles. We want to record information about the location, orientation, number of vehicles, their relative position etc. in the image summary, and must capture this in the DDL description of a prototypical image in such a collection.

If sophisticated software for detecting tanks and other military vehicles in an image is available, we want to be able to use it. The first step in doing this is to register a new simple class type, say *military-vehicles*. Next, we define specific classes, say *tanks* and *armored-cars*, of this new class type. Then, we can define a prototypical image by defining a corresponding class, say *military-scene*. For example, we may define *military-scene* as a union of classes *airplanes*, which is perhaps defined entirely using the PIQ DDL without any use of new simple classes, and *tanks*. As images in the *military-scenes* collection are entered into the database, PIQ automatically extracts the summary tree of each image and stores it.

Since we rely upon pre-existing, specialized feature extraction algorithms for recognizing tanks, what have we gained from our use of PIQ, in particular, the DDL definition? (1) The extensible PIQ framework allowed us to analyze each image with the specialized extraction code for tanks in conjunction with built-in PIQ techniques for airplanes; we can obviously use other specialized code for, say, *missile-launchers* as well. (2) The query facility can be used for queries involving many objects from different classes and can refer to relationships among them. Figure 9 is an example for an image in such a collection. It would be retrieved for the query "*Find all images (in the collection) with at least two planes flying above at least one tank*". It would not be retrieved for the query "*find all images in which there is only one type of vehicle*". (We assume that the military vehicle software identifies the type of the vehicle as well as other attributes such as *position* etc.) Both queries are examples of queries that to the best of our knowledge can not be posed in other present image database systems.

9.2 Discussion

The experiments with the real data set highlighted two important points. The first point is that once we populate the database with summaries of a collection of images based on the DDL description, we are able to support a rich class of content-based queries. We demonstrated that using our data model can help us answer queries that are very difficult or impossible to pose in most other image database systems. We also illustrated how the data model allows us to define "similarity" in many different ways.

The second point is that our automated extraction using only built-in simple classes in PIQ is not flexible enough to handle very complex data sets satisfactorily; the extensibility features of PIQ are therefore very important. For example, we could not handle the entire collection of images from the NASA Dryden source, since it involves a deep understanding of what an aircraft is. One way of dealing with such difficult cases is to try automated extraction as much as possible and ask for a user's help when the extraction fails. The extensibility features in PIQ allow a user to provide such help by registering domain-specific extraction algorithms, which can then be used naturally in DDL definitions.

The system design allows for several types of enhancements. If we find some sophisticated algorithm for identifying objects of some image class, we can easily incorporate it into the system as another "simple" class. Our main extraction algorithm would invoke the sophisticated extraction algorithm when an instance of that class is to be searched for. The querying facility doesn't need to know about the sophisticated algorithm, but uses the attributes that it extracts.

The system already supports simple class definitions using images as *examples*, although we have not discussed it here. Exploring how to use and to enhance this feature for defining complex images is an important direction for future work. We note that even if, in an extreme situation, a user has to assist in extracting an object, a DDL definition guides us in deciding what feature information to extract and store, in order to subsequently answer queries of interest.

The PIQ system is in a very early stage of development. We expect to refine it as we gain experience by working with a variety of data sets. While we have thus far concentrated on a declarative DDL, it may well be that this must be complemented by a more procedural DDL in certain situations.

10 Related Work

Perhaps the most sophisticated system supporting query by content over general image sets is QBIC [1]. QBIC automatically extracts the following attributes for images and for user-identified objects within images: *color, shape, sketch* and *texture*.

We have followed QBIC with respect to how we handle color and shape attributes. A sketch in QBIC is a very reduced resolution edge-map. It is computed only for images and the only query possible is similarity to some user defined sketch.

Identifying an object in the image is left to the user. It is usually done by using a set of tools that allow users to identify an image using the mouse. The drawback here is that users have to put in some effort *per image*, and this can be a prohibitive limitation for very large collections of images. The second major limitation is that queries are limited to the extracted information, which consists of just color, shape, texture and sketches. Thus, queries about the relationships between component objects and their features cannot be posed. In PIQ, we have sought to address these two limitations of QBIC.

A notable example of a system that is tailored to a specific domain of images is VIMSYS [11]. Although VIMSYS is specialized to a single domain (faces), a design objective was to collect all the domain-specific details into one module. The system performs well for the domain of faces. However, it needs a special

module for each data domain, and the data model for each domain is rigid (i.e., it cannot be changed after the code is written). Thus, VIMSYS remains essentially a specialized system. The VIMSYS goals and designs have influenced PIQ. We have tried to provide a higher-level mechanism for customization to different image domains through the use of a DDL, rather than through the use of a (well-specified) domain-specific library of routines.

There are many published results on techniques for similarity searches based on color, shape, etc., but we do not discuss these since we regard them as largely orthogonal to the DDL itself. These results essentially provide a richer set of choices for the image procesing library used by our main feature extraction algorithm.

11 Conclusion

We have described the query by image content facility being developed in the PIQ project, which is still in an early stage. The main contribution is the observation that the use of an image DDL can assist in automatic extraction of image features, and, perhaps more importantly, in retrieving and structuring the information necessary for answering image-specific conten-based queries. We have presented the results of several experiments designed to test and validate this claim. Future work will be focused on refining the DDL framework and developing more powerful mechanisms for defining similarity, querying and indexing.

References

[1] R. Barber, W. Equitz, C. Faloutsos, M. Flickner, W. Niblack, D. Petkovic, and P. Yanker, "Query by Content for Large On-Line Image Collections", IBM Technical Report RJ-9408, Yorktown Heights, NY, 1993

[2] Beckmann, N., Kriegel, H.P., Schneider, R., Seeger, B. "The R* Tree: An Efficient and Robust Access Method for Points and Rectangles", Proc. ACM SIGMOD Int. Conf. on Management of Data, 1990.

[3] Carey, M., DeWitt, D., Franklin, M., Hall, N., McAuliffe, M., Naughton, J., Schuh, D., Solomon, M., Tan, C., Tsatalos,O., White, S., Zwilling, M. "Shoring up Persistent Objects", Proc. ACM SIGMOD Int. Conf. on Management of Data, 1994.

[4] Chiueh, T., "Content-Based Image Indexing," Proc. 20th Int. Conf. on Very Large Data Bases, 1994.

[5] DeWitt, D., Kabra, N., Luo, J., Patel, J., Yu, J. "Client-Server Paradise", Proc. 20th Int. Conf. on Very Large Data Bases, 1994.

[6] Freeston, M.W., "The Bang File: A New Kind of Grid File," Proc. ACM Sigmod Int. Conf. on Management of Data, 1987.

[7] Goldstein, J., Ramakrishnan, R. and Yu, Jie-bing "Using Constraints to Query R* Trees", Manuscript.

[8] Grosky, W.I., "Toward a Data Model for Integrated Pictorial Databases," Computer Vision, Graphics, and Image Processing, Vol. 25, No. 3, pp. 371–382, 1984.

[9] Grosky, W.I., and Mehrotra, R., "Index-based Object Recognition in Pictorial Data Management," *Computer Vision, Graphics, and Image Processing*, vol. 52, pp. 416–436, 1990.

[10] Grosky, W.I., "Multimedia Information Systems," IEEE MultiMedia Magazine, Vol. 1, No. 1, Spring 1994.

[11] A. Gupta, T. Weymouth, and R. Jain, "Semantic Queries with Pictures: the VIMSYS Model," Proc. of the 17th Int. Conf. on Very Large Data Bases, September 1991.

[12] Guttman, A. "R Trees: A Dynamic Index Structure for Spatial Searching," Proc. ACM SIGMOD Int. Conf. on Management of Data, 1984.

[13] Hellerstein, J.M., "Practical Predicate Placement," Proc. ACM Sigmod Int. Conf. on Management of Data, 1994.

[14] Jagadish, H.V. "A Retrieval Technique for Similar Shapes," Proc. ACM SIGMOD Int. Conf. on Management of Data, 1991.

[15] Ramesh Jain, R. Kasturi, and B. G. Schunck. *Machine Vision.* McGraw Hill, 1995.

[16] Ramesh Jain, "NSF Workshop on Visual Information Management Systems," SIGMOD Record, Vol. 22, No. 3, pp.57-75, 1993.

[17] Kurokawa, M., "An Approach to Retrieving Images by Using their Pictorial Features," Proc. of the ICIP, Singapore, September 1989, ICIP.

[18] King-Ip Lin, H.V. Jagadish, and Christos Faloutsos, "The TV-Tree - an Index Structure for High-Dimensional Data", VLDB Journal, Vol. 3, pp. 517-542, 1994.

[19] D. Lomet and B. Salzberg, "The hB-Tree: A Multi-attribute Access Method with Good Guaranteed Performance," ACM TODS Vol. 15, No. 4, Dec. 1990.

[20] Nievergelt, J., Hinterberger, H., Sevcik, S.C. "The Grid File: An Adaptable, Symmetric Multikey File Structure," Readings in Database Systems, Morgan Kaufmann, 1988.

[21] F. Rabitti and P. Savino, "Query processing on Image Databases," 2nd Working Conf, on Visual Database Systems, pp. 174–88, Budapest, Hungary, April 1991, IFIP WG 2.6.

[22] Ramakrishnan, R., Srivastava, D., and Sudarshan, S. "CORAL: Control, Relations and Logic," Proc. of the Int. Conf. on Very Large Databases, 1992.

[23] Samet, H., "The Quadtree and Related Hierarchical Data Structures," Computing Surveys, June 1984.

[24] Santini, S. and Jain, R., "Similarity Matching", Submitted to IEEE Trans. on PAMI, 1995.

[25] Sellis, T., Roussopoulos, N., Faloutsos, C., "The R+ Tree: A Dynamic Index for Multi-Dimensional Objects," Proc. 13th Inf. Conf. on VLDB, 1987, pp. 507-518.

[26] Seshadri, P., Livny, M. and Ramakrishnan, R., "SEQ: A Model for Sequence Databases," Proc. of the Int. Conf. on Data Engineering, Taiwan, 1995.

[27] Stonebraker, M., Frew, J., Gardels, K. and Meredith, J., "The Sequoia 2000 Benchmark," Proc. ACM SIGMOD Int. Conf. on Management of Data, 1993.

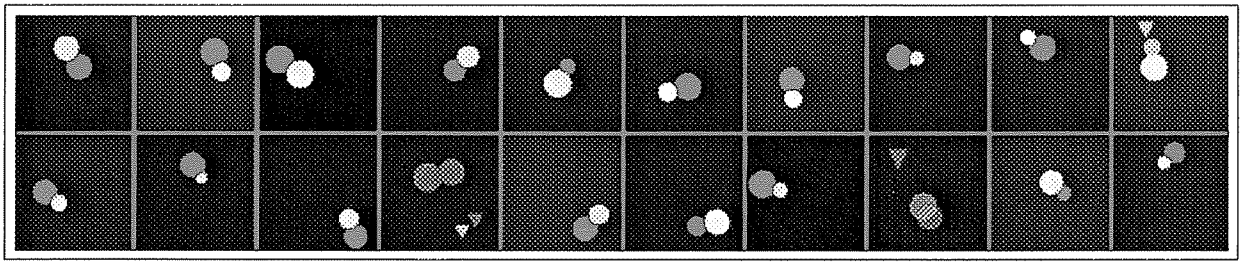

Figure 10: Similarity query. PIQ data model.

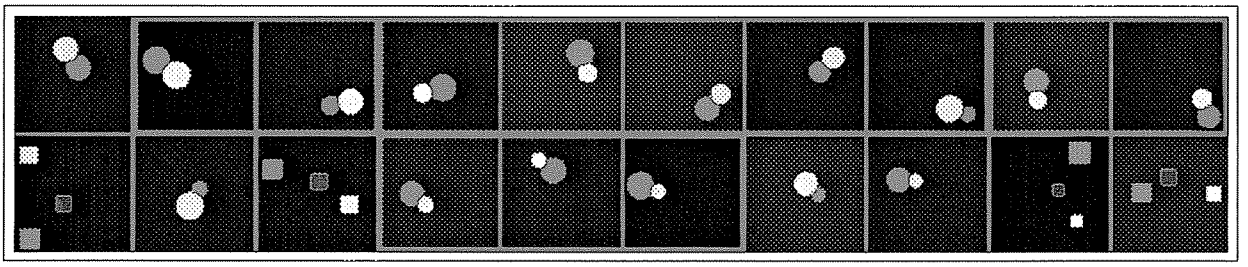

Figure 11: Similarity query. "Control 1" data model.

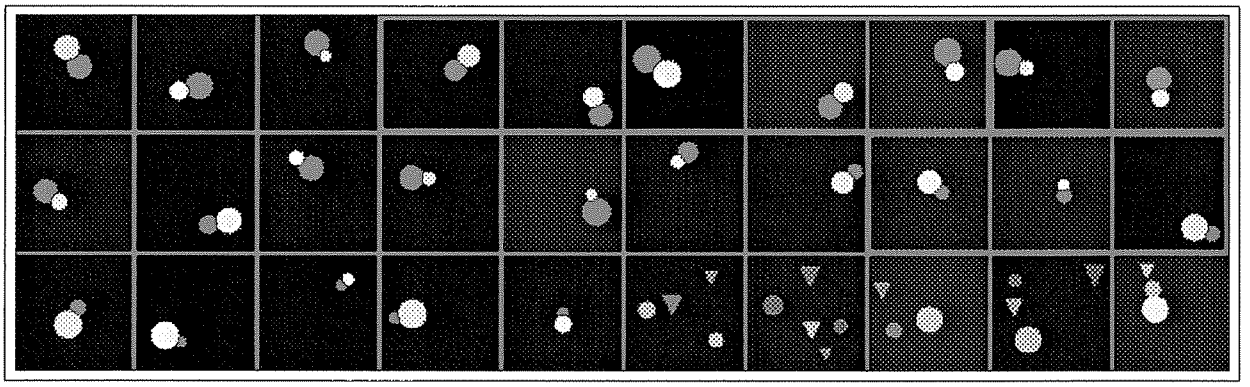

Figure 12: Similarity query. "Control 2" data model.

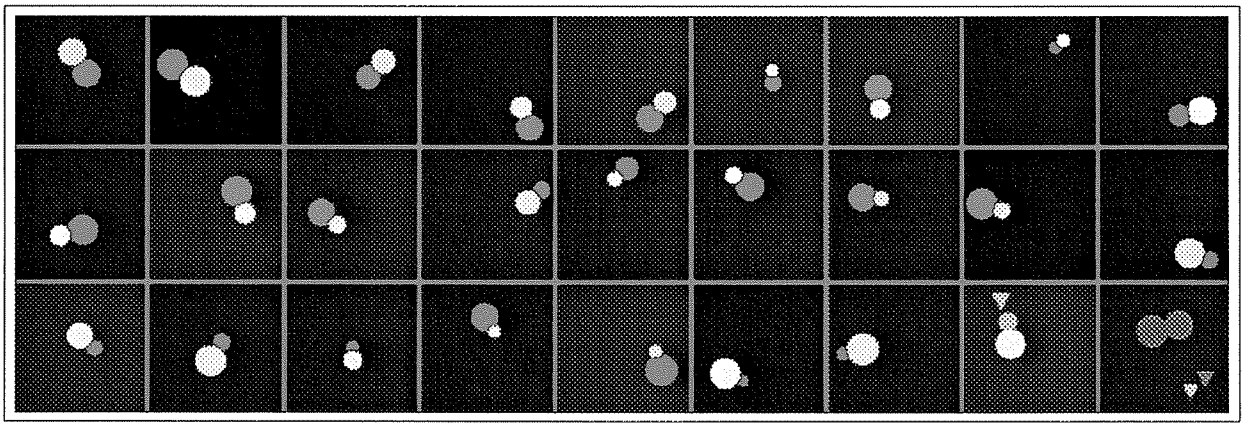

Figure 13: Similarity query. "Limited PIQ" similarity.

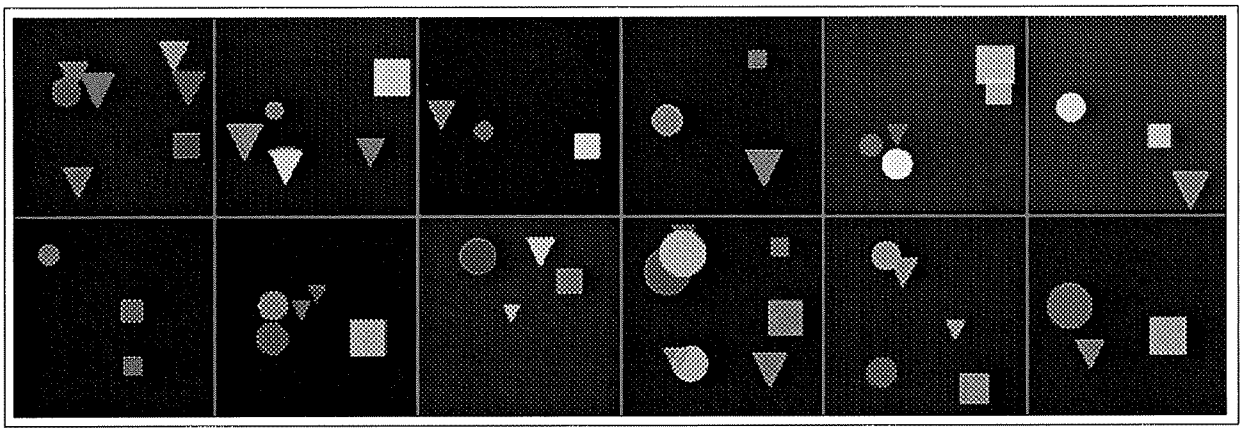

Figure 14: Result of query: "Images in which all the circles are to the left of all the squares". Results are ordered by the width of the gap between circles and squares.

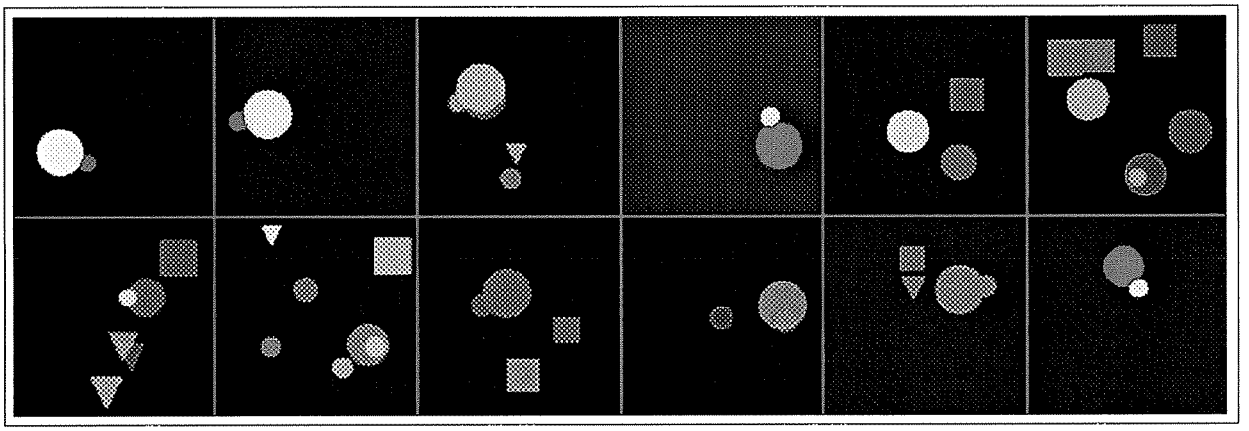

Figure 15: Result of query: "Images in which there is a pair of adjacent circles , the ratio of size between the large and smaller circle is at least 3". Results are ordered according to that size ratio.

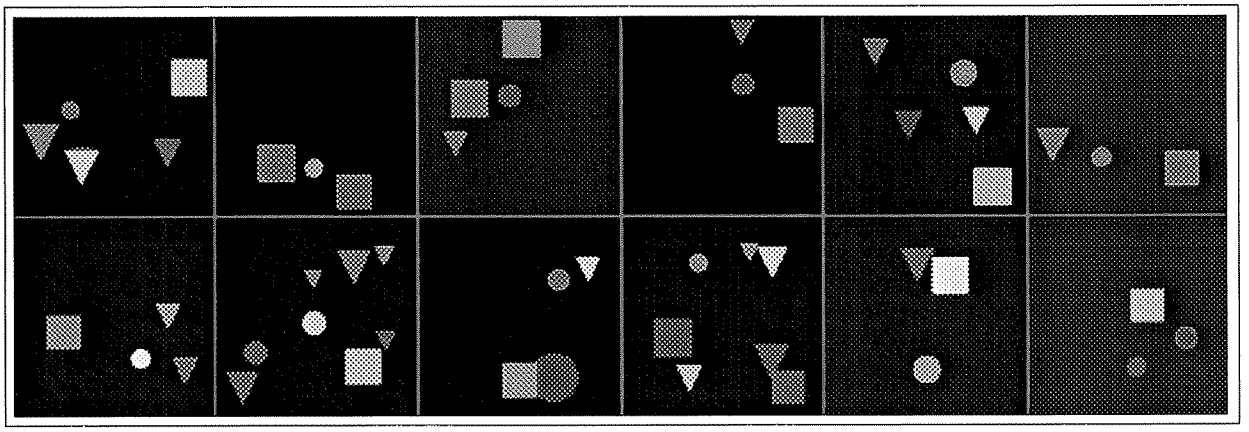

Figure 16: Result of query: "Images in which there are circles and squares, and all circles are smaller than all squares". Results are ordered by the difference in size between the smallest square and the largest circle.

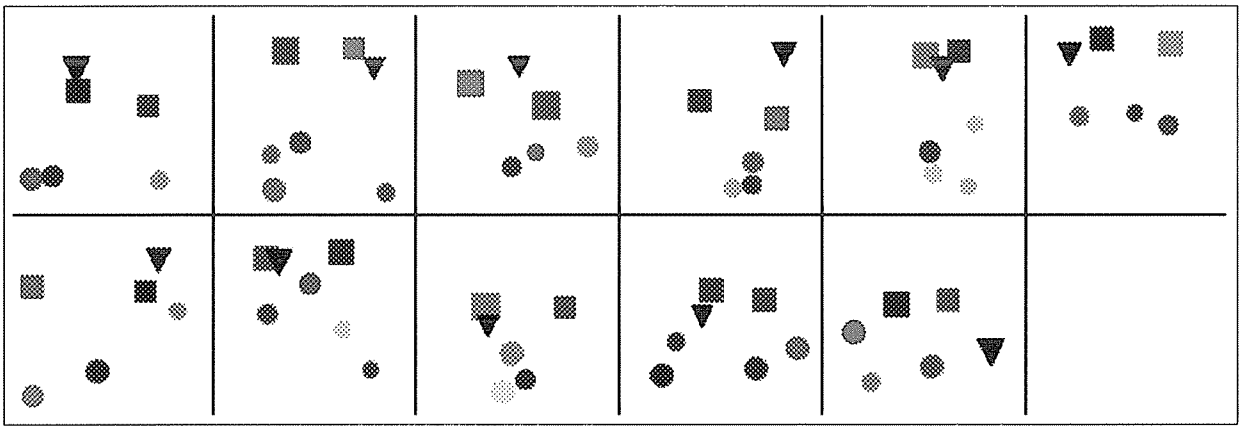

Figure 17: Result of query: "Images in which there are at least three circles, all of them below the squares and the blue triangle is closer to a square than it is to any of the circles". Results are ordered by the difference between the distance of the triangle to the nearest circle and the nearest square.

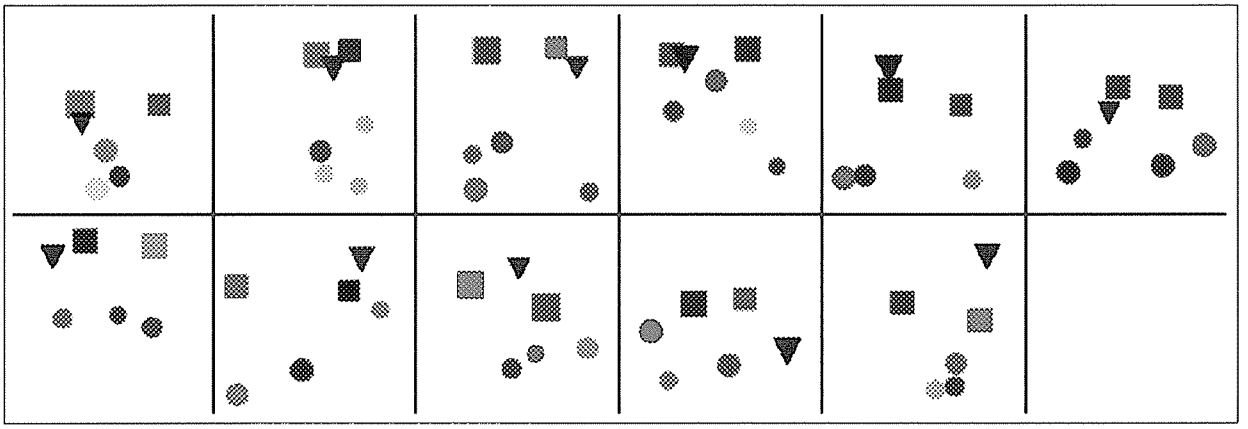

Figure 18: Result of the same query as in Figure 17. Results are in reverse order of the distance of the triangle to the nearest square.

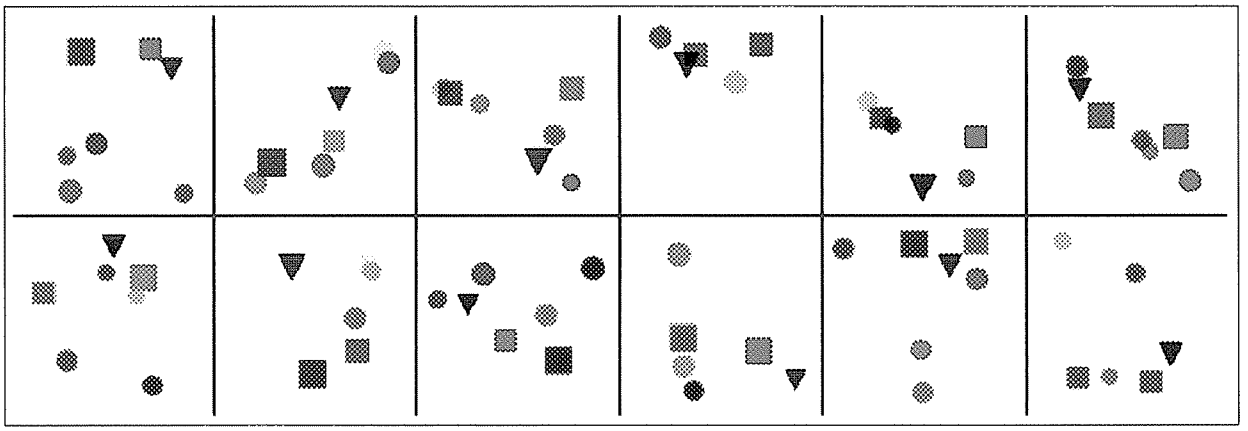

Figure 19: Result of query: "Images in which the square closest to the triangle is green, the other square is not green". Results are ordered by the amount of red and orange circles in the image.

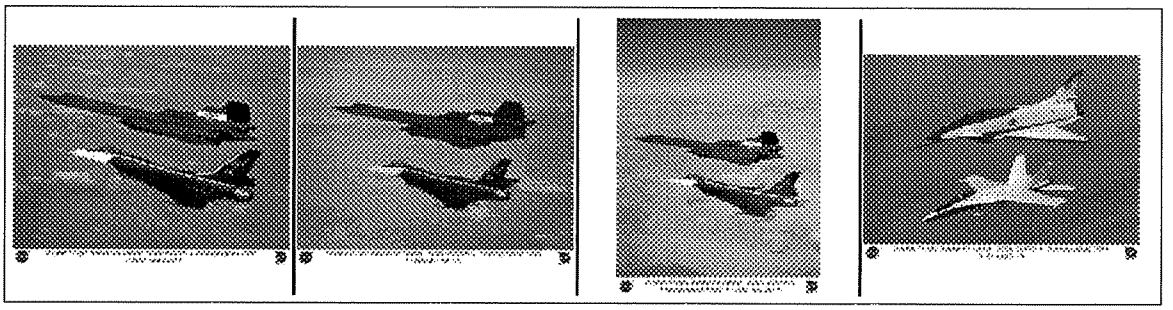

Figure 20: Result of query: "Images with two planes ordered by blackness of top plane."

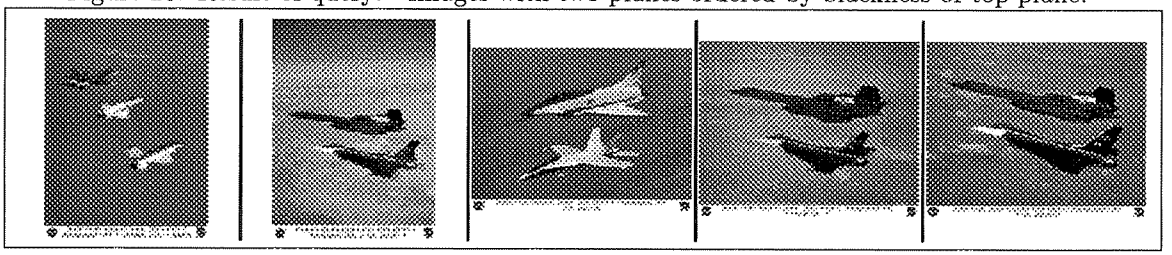

Figure 21: Result of query: "Images with at least two planes ordered by size of bottom plane."

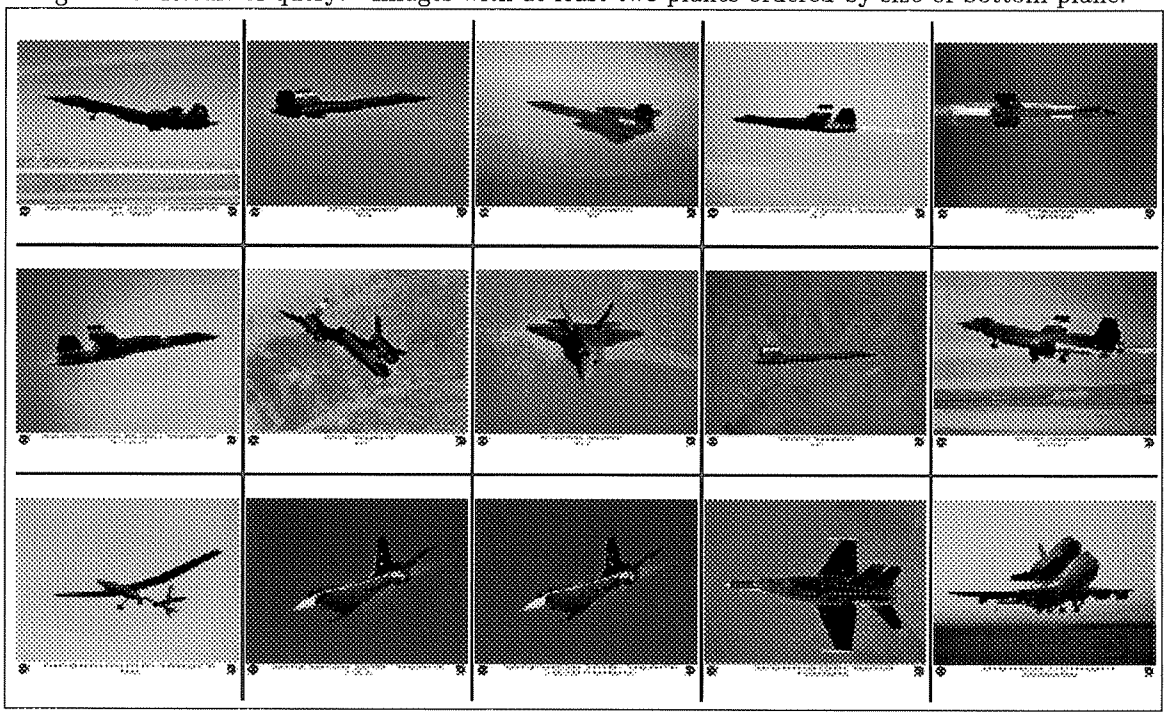

Figure 22: Similarity query. Best 15 results.

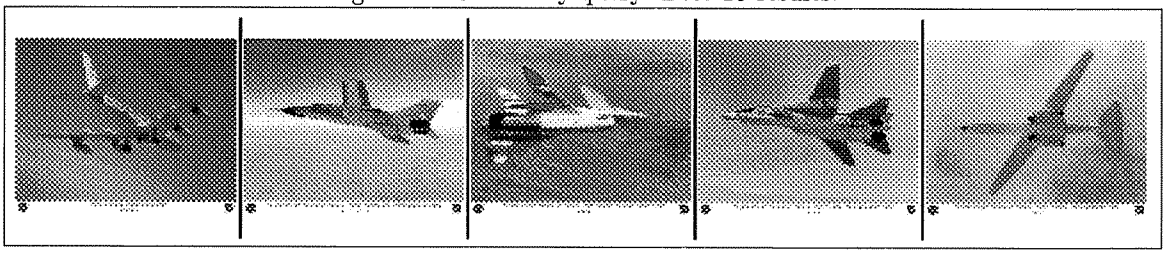

Figure 23: Same similarity query as the previous figure. Worst 5 results.