Data modeling and feature extraction for image databases

Uri Shaft and Raghu Ramakrishnan

Computer Sciences Department, University of Wisconsin-Madison
1210 West Dayton Street, Madison, WI 53706
uri,raghu@cs.wisc.edu

ABSTRACT

Current image retrieval systems have many important limitations. Many are specialized for a particular domain of images, and are not applicable to other image domains. The more general systems treat all images uniformly. Consequently, the power of their query facility is limited to color, texture, shape, and other features that are common to all images, with no deeper understanding of the structure of a given image. Few systems (if any) have addressed the issue of *scalability* with respect to the size of the image collection and with respect to the underlying techniques.

There are two communities that can contribute to image databases: Computer Vision[7] and Database Systems. In this paper we focus on the database side of the issue. We consider how to design a database system that supports a rich class of content-based queries on image collections, scales with collection size, and can easily incorporate future advances in computer vision. This paper outlines one approach, in the form of the design, implementation and testing of an image database system called PIQ. The main contributions we make are:

1. The idea of a *data model* for describing image data. This is coupled with an Object Model Description Language (OMDL) for describing image domains. We introduce our first cut at such a data model and its description language.
2. The "Feature Extraction Manager". This is a general algorithm for extracting features from images that utilizes the data model and any computer vision algorithms given to it and manages the search for objects in each of the given images.
3. A demonstration of the power of a query language that is built on top of an image data model.
4. A system design with an extensibility feature that can incorporate new feature extraction algorithms.

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 OMDL 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.

Keywords: image database, data model, feature extraction, extraction management.

1. INTRODUCTION

Image Database Management Systems (IDBMS)[1−6,8,9,11] have two main purposes: storing large collections of images and efficient content-based retrieval of the images. When building an IDBMS one has to address the following issues:

Storage. Can the IDBMS handle the storage requirements of a large collection of images? This usually means using tertiary storage devices (e.g. tapes). Note that integrating the IDBMS with a DBMS that handles tertiary storage devices can solve the storage problem.

Expressibility of the query language. Can a user specify the contents of the images that he/she cares about? In other words: are the queries that the user wants to issue supported by the query language?

Efficient query execution. Can the queries be optimized and executed using reasonable amounts of resources such as time and I/O.

This paper focuses on efficient content-based retrieval of images. In particular we focus on the aspects of our IDBMS (called PIQ) that deal with the query language and query optimization.

1.1. Image summaries

The kind of queries we focus on are selection queries. In other words, we want to retrieve a few images from the database and select them based on their content. Imagine a query facility that loads all the images from the database and checks each image to see if it fits the selection criterion. In all but extreme cases this query facility would be inefficient. The only other option is to have more information about the images stored in the database. We call this extra information a "summary" (it is usually referred to as "features"). When an image is stored in the database, its summary is stored as well. Queries are executed using the information in the summaries and images are retrieved only after we check that the summary conforms to the selection criterion. This means that any content information we want to query needs to be in the summary. Therefore, the expressibility of any query language in this situation is limited by the type of information in the summary.

Note that every existing IDBMS uses the idea of a summary for query optimization and execution. Some existing systems create the summaries automatically when given an image to insert into the database. Other existing systems need the help of a user in constructing the summary. User supplied input on a per-image basis has dire consequences on the system's ability to scale. One can't expect a user to provide input for many thousands of images. Therefore, a system capable of handling large collections of images must have automated summary construction (also called "feature extraction").

1.2. Existing IDBMSs vs. PIQ

Existing IDBMSs focus on a single domain of images. The type of information in the summary is determined at the time the system is built and can't be changed.

Some systems consider only a very specialized image domain. Images in that domain have a much in common .This commonality is reflected in the type of information they put in summaries. For example, the VIMS[5] (or VIMSYS) system can handle only images of frontal views of single Human faces. The type of information in the summaries fits this image domain. A special feature extraction algorithm was constructed so that summaries can be created automatically. The query language is designed for the specific domain as well. These kinds of systems have a high development and maintenance costs. Every change we make, such as including new features or a new type of query, needs code development. If we want to support a different image domain we need to build another system altogether.

Other systems consider the domain of all images. Any image can be inserted into the database and all images are treated equally. Images inserted into the database may have nothing in common other than the fact that they are images. This means that the type of information that can be extracted automatically must be relevant to every possible image. This excludes any notion of semantic contents of the information in the summaries. QBIC[1] is an example for such a system. When an image is inserted with no user supplied input the system extracts two features of the image: a color histogram, and a condensed edge map that represents the layout of the image. The user may manually identify regions of the image (called "objects"). For each object, the system extracts its color histogram, its shape, and sometimes its texture. The expressibility of such query languages are poor because there is not much information is the summaries to work with.

PIQ differs from existing IDBMSs by being able to handle different specialized domains. We introduce the idea of "image data modeling" as a remedy to the problems listed above for existing IDBMSs. Data modeling is used for describing the contents of images of a specific image domain. We show how it can be used for both automated feature extraction (and hence scalability) and for a flexible query facility that has more power than any existing IDBMS.

1.3. Outline of the paper

The rest of the paper is as follows: Section 2 describes image data modeling in general. In Section 3 we introduce a specific data model that we developed for PIQ. In Section 4 we present the general feature extraction algorithm used in PIQ. In Section 5 we give examples of the expressibility of queries in PIQ. Section 6 contains the conclusions and future work.

2. DATA MODELING FOR IMAGES

2.1. What is a data model?

The usual situation we encounter is that images come in collections. In other words, images in the same collection have something in common. For example, take the collection of faces as described in the introduction for the VIMSYS system. Each image has a frontal view of a face where each face has eyes, a nose, a mouth etc. We would like the summary of images in collections like this to reflect the common features. The summaries should also reflect the features that we think are important for differentiating between the images. For example, for a collection of airplane photos we may want to know the color, shape and markings for each plane. We may also want to know the number of planes in an image, the orientation and relative position and size of planes etc. Finally, we do not want to build an entire database system every time we have a new collection of images. Instead, we want the same system to deal with all the collections.

The logical way to do this is to describe the features we are interested in for each collection. To do that we need an object model description language (OMDL). A specific description of a collection (using a OMDL) is called a "schema". The set of all legal schemas for a specific OMDL is called a "data model". For example, the relational data model allows us to describe relations (tables). Each schema in the relational data model tells us how many attributes (columns) are in the table and what is the name and type of each attribute. Similarly, we need a data model for images so we can create schemas for different image collections. These schemas describe what the content of an image from that collection consists of.

Current IDBMSs do not have a data model. Essentially, they support a fixed schema for a specific image domain. In such systems either image domains other than the one supported by the system are ignored or the issue of image domains is ignored.

2.2. How is an image data model used?

The OMDL is used to define schemas for image collections. A schema tells us what the summary of an image consists of. For example, in the airplanes collection, we may want to know the wing-span of each airplane or the markings that identify it. To support this, the schema may require that the summary have a number corresponding to the wingspan of the airplane. The schema may also require that each summary have one string per airplane in the image corresponding to identity markings of the airplane. Thus, the schema defines the type of data that should be in a summary. This information about summaries is used in two major parts of an image database system:

Feature extraction. We call the mechanism that computes a summary for an image "feature extraction". Obviously, an algorithm that computes a summary for an image needs the schema so it can return the correct information in the summary. Therefore, a feature extraction algorithm receives as input both an image and a schema and returns a summary.

Queries. We can ask queries using the information in the summaries. Therefore, we must know what type of information is in there. This means that a query over an image collection adheres to the schema. For example, the query "find images of airplanes which are identified as F-16" is possible only if the schema of the airplanes collection says that the identity of an airplane is present in the summary.

This stresses an important point about data models: when a data model is suggested for an image database system it should be able to describe the image collections we are interested in. The same data model should also have a feature extraction algorithm that can create summaries according to schemas of the data model. Without the feature extraction algorithm, the data model has very limited use. A data model for image collections differs from the usual data models (such as the "relational data model" or the "object-oriented data model") in that it is used for preprocessing the data using feature extraction. When any data model is developed, its expressibility is of major concern. We want to be able to create schemas that describe the things that are important to us in the data. An image data model also needs to take into account what feature extraction algorithms can do. A schema may need to contain information that is relevant only to the feature extraction algorithm. For example, a schema for a collection of faces may contain the fact that a nose is underneath the eyes. This may help the feature extraction algorithm to locate the nose but may not be needed as a part of the summary.

2.3. What we do in PIQ

PIQ demonstrates the data model idea. To do that we created an image data model. Along with the data model we supply a feature extraction algorithm and a query facility that can use schemas in that data model. The data model of PIQ is only an example for a possible data model. It is not the "correct" data model in the same way that the relational model is not the "correct" data model for conventional databases. The PIQ data model works well in many situations (for many collections) but does not fit other situations. We demonstrate that having such a data model is still strictly better than having none at all.

3. THE PIQ IMAGE DATA MODEL

The PIQ image data model is used for demonstrating that a system with a data model is much more powerful relative to existing IDBMSs. We do not suggest that it is the best model for all circumstances.

3.1. General form of a PIQ schema and summary

As mentioned before, a schema is a description of a collection of images (we call it an "image class"). In PIQ, a schema is a tree structure. Each node in the tree represents an object (or region) of the image. Figure 1 shows a simplified schema for a class of images of airplanes. Note that there may be any number of planes in each image. The node marked "plane" corresponds to one airplane in the image. The root of the schema (marked as "airplanes") corresponds to all the airplanes in the image. Each airplane is an object and their union is an object as well. Note that any region in an image, of any complexity whatsoever, can be considered an object. The type of the root node is "set". This means that the object of the root node is the union of the set of objects that belong to its children. The "plane" node has type "union". It means that the object of node "plane" is the union of an object of node "wings", an object of node "fuselage" and an object of node "tail". The types of the leaf nodes is not specified in Figure 1. We will discuss the types of leaf nodes in Section 4.1.

Internal nodes correspond to operations done on their children. The operations tell us how to construct the object of an internal node from the objects of its children. The types of internal nodes are:

Union. As seen with the airplanes example, the object corresponding to a node of this type is the union of the objects of its children. Since objects are regions of an image and a region is a set of pixels, the union is just a union of sets of pixels resulting in a single set of pixels.

Intersection. The same as union but using set intersection as the operation.

Difference. The same as union and intersection but using set difference as the operation. Nodes of this type must have exactly two children. The resultant object is the difference of the right child from the left child.

Set. Nodes of this type have one child. The object of such a node is the union of all objects that can correspond to the child.

Or. An object of a node of this type is the object of one of its children. Figure 2 is an example for the use of such a node. Assume a collection of images in which each image contains one aircraft of any type. Suppose the types of aircraft are "plane", "helicopter" and "missile". Figure 2 shows the top levels of the schema tree. In the summary of an image of a missile the object for the "aircraft" node will be the same as the object of the "missile" node. That image's summary will not have any information about a plane or a helicopter.

A summary of an image is a tree that corresponds to the schema of the image and the objects in the image. Each node in the summary tree corresponds to a node in the schema tree. However, there may be a few nodes in the summary tree that correspond to a single node in the schema tree. Similarly, there may be nodes in the schema tree with no corresponding node in the summary tree. Figure 3 is an example of a summary tree of an image with two airplanes in it according to the schema of Figure 1. The root of a summary tree always corresponds to the root of a schema tree. Since the image contains two airplanes, the summary tree has two subtrees, each headed with a node that corresponds to the "plane" node of the schema tree. The node marked as "1" in the summary tree corresponds to one region of the image that has an airplane in it. The node marked as "2" corresponds to the other region in the image with a plane in it. Figure 4 is an example of a summary tree of an image that contains a helicopter with the schema of Figure 2. Note that there are no nodes that correspond to airplanes or to missiles.

3.2. Attributes

In Section 3.1 we have seen what the general form of a schema looks like. Had a summary been only the form of the tree, there wouldn't have been much that could be described with it. We need additional information about the image. For example, we may want to know what the wingspan of an airplane is. We get this information by using "attributes". For each node in the schema tree we may define several attributes of several predetermined types. For example, we may define the attribute "wingspan" (of numeric type) as an attribute of node "plane". Each node in the summary tree corresponds to a specific node in the schema and has the same attributes as the schema node. The difference is that summary tree attributes have the actual values for them taken from the image. Suppose we have the attributes "color" and "location" for each schema node. We will have values for those attributes for each summary node. This is enough information for calculating the query: "find all images that have two airplanes in them where the top airplane has a white fuselage". If we include an attribute "identity-mark" for each "plane" node as well, we can ask the query: "find all images with an F-15 over a MIG-29". Section 4 deals with the problem of obtaining values for the attributes. The types of attributes we allow in PIQ are:

Basic attributes. These are numbers, strings and booleans.

Region. A set of pixels from the image.

Array. A multidimensional array of any of the basic types (and region type).

3.3. Restrictions

Consider the case of a collection of faces. Each image in the collection contains a single face in frontal viewpoint. Suppose we are interested only in the eyes. Figure 5 shows a simple schema for this situation. Suppose an image actually has one face with two eyes in it. There are four different legal summaries that correspond to the schema. Each eye in the image can be assigned to either the "left-eye" node or the "right-eye" node or to both or to none. We need exactly two such assignments, so we have four possibilities. Obviously, we want the left eye in the image to be assigned to the "left-eye" node and the right eye in the image to be assigned to the "right-eye" node. This means that we want to restrict the set of legal summaries. In general, feature extraction should be restricted to return only one legal summary as a result. If feature extraction can't perform this crucial task then the user must decide what is the correct summary; although this implies a non-automated creation of the summaries.

We include boolean expressions with the schema and call them "restrictions". The variables for the expressions are the attributes in the schema nodes. A summary is legal only if all the restrictions are evaluated to "true" given the values for the attributes. For example, we can have a restriction for the "eyes" schema that says that "left-eye" is located to the left of "right-eye". We can do that using the location attribute of each of the leaf nodes. (The location attribute can be an array of two numbers corresponding to the X, Y coordinates of the object.)

4. EXTRACTION MANAGEMENT

As we have seen, automated feature extraction is essential. Unfortunately, algorithms that perform specific tasks of feature extraction are very hard to develop. In fact, the entire Computer Vision community works on the problem of feature extraction. An extraction algorithm for a specific type of feature may take years to develop.

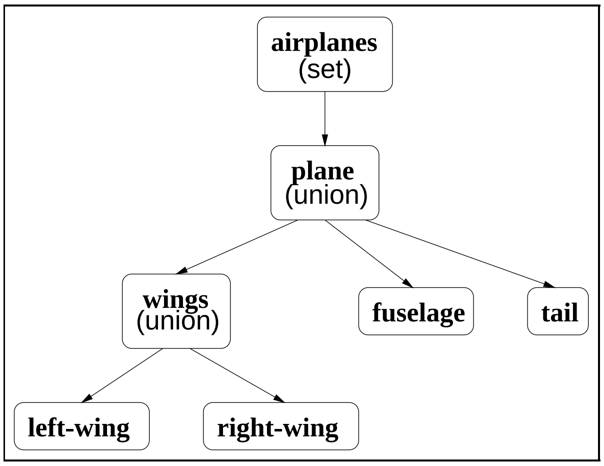

Figure 1: Example of schema for "airplanes"

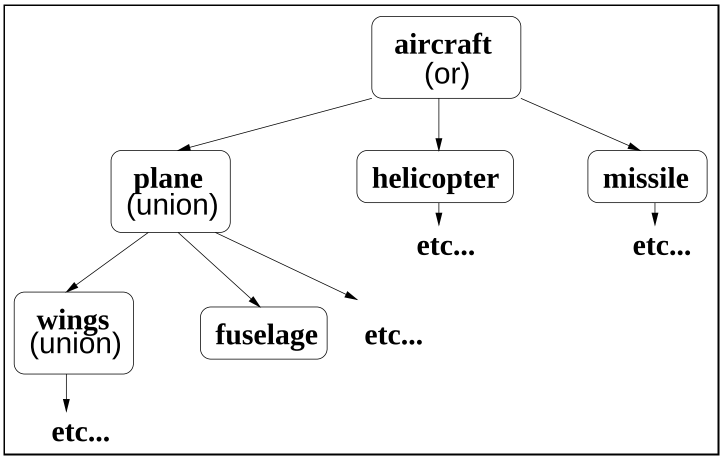

Figure 2: Example of schema for "aircraft"

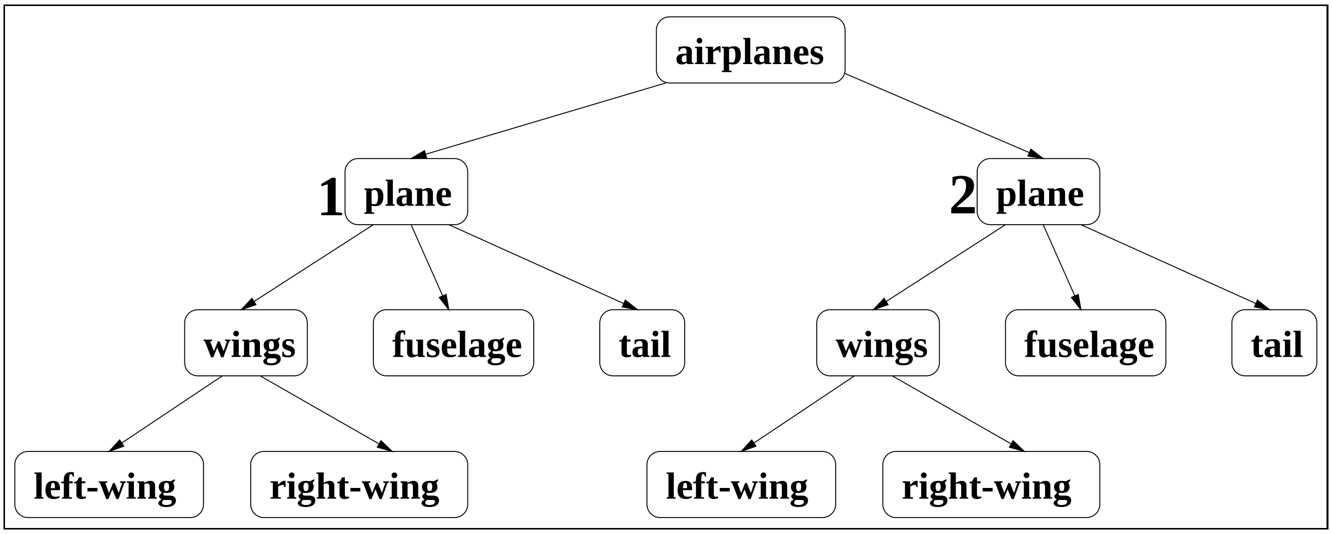

Figure 3: Example of summary for "airplanes"

Therefore, the people that develop it usually do not integrate it with other algorithms. We see the role of the IDBMS feature extraction algorithm as the glue that holds all these different algorithms together. Our purpose is to provide a facility that coordinates different feature extraction algorithms (taken from the Computer Vision community). Thus, fully automated feature extraction can be supported as long as the Computer Vision community can supply the different algorithms for each type of feature in the schema. We call the coordination facility "extraction management".

4.1. The types of schema leaf nodes

Each leaf of a schema has an extraction algorithm associated with it. This extraction algorithm is the one used to obtain values for the attributes of that leaf when a summary is constructed. The type of feature extraction algorithm associated with a leaf is essentially the leaf's node type. PIQ maintains a list of feature extraction algorithms developed by the Computer Vision community. Each such algorithm is a leaf node type. Note that different leaves of the same schema may have the same type and consequently use the same feature extraction algorithm.

The list of algorithms in PIQ is dynamic. When an algorithm is developed, it can be given to PIQ by defining a new leaf node type. The extraction management algorithm uses those feature extraction algorithms as subroutines.

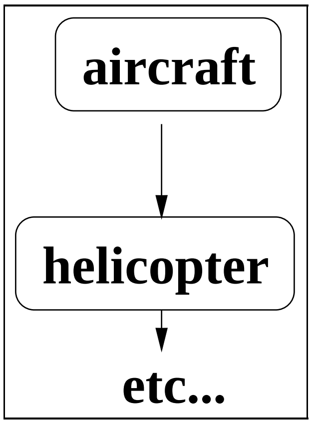

Figure 4: Example of summary for "aircraft"

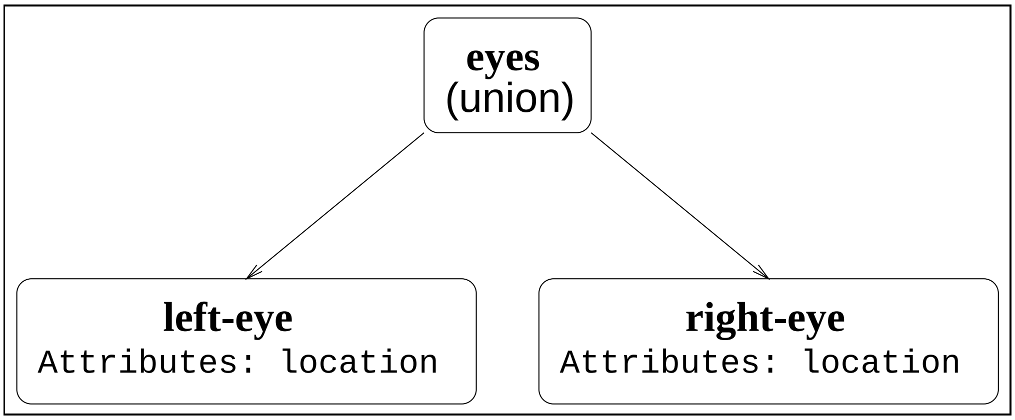

Figure 5: Example of schema for "eyes"

Thus, PIQ maintains a full separation of the database technology and the Computer Vision technology.

4.2. Computable attributes

There are two ways attributes are defined. First, the return value of a feature extraction algorithm is a set of values for a set of attributes. These attributes are automatically defined for a leaf node according to the feature extraction algorithm the node is associated with. Second, an attribute may be defined for a schema node by giving its name, and the expression that is used to compute it. The expression may use as variables all attributes of the node and the descendents that are already defined.

An attribute called "primary-region" is automatically defined for each node. The value of that attribute is the object the node is associated with in an image. The value of the primary region of a leaf node is given by the feature extraction algorithm when used on a specific image. The value of the primary region of an internal node is computed from the primary regions of its children according to the type of the internal node (see Section 3.1).

4.3. Feature extraction for simple schemas

A schema is considered to be simple if it does not contain internal nodes of types *set* and *or*. A schema that has a *set* or an *or* node is refered to as a "complex schema". A summary of an image with such a schema is a tree that has the exact shape of the schema tree. In other words, every node in the summary tree corresponds to exactly one node of the schema tree and vice versa. The extraction manager creates a summary for an image by building the summary tree in a bottom-up fashion.

A partially evaluated summary is a tree shaped like the summary tree, except that some, or even all, of the attributes may not have values assigned to them. The extraction manager maintains a list of partially evaluated summaries. It works as follows:

1. Create a list of partially evaluated summaries. Initialize the list with one totally non-evaluated summary (a summary with no values assigned to any attribute).
2. While the list is not empty do
   (a) Remove a summary S from the list.
   (b) Identify a leaf of S that is not evaluated and call its feature extraction algorithm.
   (c) Suppose the feature extraction algorithm returned k results. Create k copies of S (called $S_1, .., S_k$).
   (d) Assign values to the leaf of each S_i according to result i of the feature extraction algorithm.
   (e) For each S_i: evaluate all possible computable attributes. Then, evaluate all restrictions (if possible). If any of the restrictions evaluated to "false" discard the tree.

(f) For each remaining S_i: if it is fully evaluated (i.e., a legal summary) put it in a list of results. Otherwise, insert it into the list of partially evaluated summaries

3. Return all the collected results.

Figure 6 shows an example of a simple schema and Figure 7 shows an example of an image. Figure 8 illustrates all the partially evaluated summaries the extraction manager uses while creating the summary for the image in Figure 7 and schema of Figure 6. The steps the extraction manager follows are:

Step 1. A list is created with tree **A** in it.

Step 2. Tree **A** is taken from the list and *leaf-1* is evaluated. This involves calling an extraction algorithm that searches for circles.

Step 3. The extraction algorithm found two circles. Trees **B** and **C** are created as copies of **A** with the values for the circles in them. Note that both the "primary-region" and "color" attributes get values.

Step 4. Since the restriction for both trees can't be evaluated they are inserted to the list. Now the list contains trees **B** and **C**.

Step 5. Tree **B** is taken from the list and *leaf-2* is evaluated.

Step 6. There are two possible results returned by the extraction algorithm, so trees **D** and **E** are created as copies of **B** with values assigned in *leaf-2*.

Step 7. For both trees it is evident that the color of *leaf-1* is not darker than the color of *leaf-2*. The restriction evaluates to "false" so both trees are discarded. The list contains only tree **C**.

Step 8. Tree **C** is taken from the list and *leaf-2* is evaluated (again).

Step 9. Trees **F** and **G** are created as done in step 6 for tree **B**.

Step 10. The restriction evaluates to "false" for tree **F** so it is discarded.

Step 11. The restriction evaluates to "true" for tree **G**. Tree **G** is returned as the summary of the image for the schema *foo*.

Note that for trees **D,E,F,G** the value of "primary-region" of the root node is the union of the values of "primary-region" of its childrens. For example, the "primary-region" of the root of tree **F** is the area of the black circle while the the "primary-region" of the root of tree **G** is the union of areas of both black and white circles.

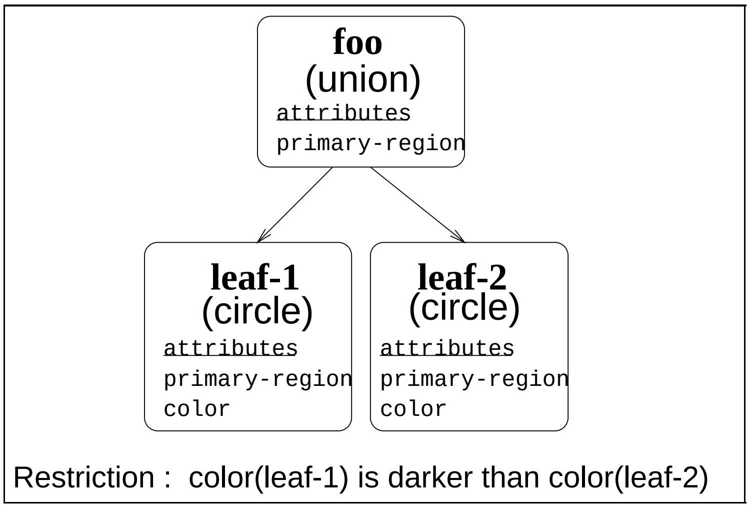

Figure 6: Example of a simple schema

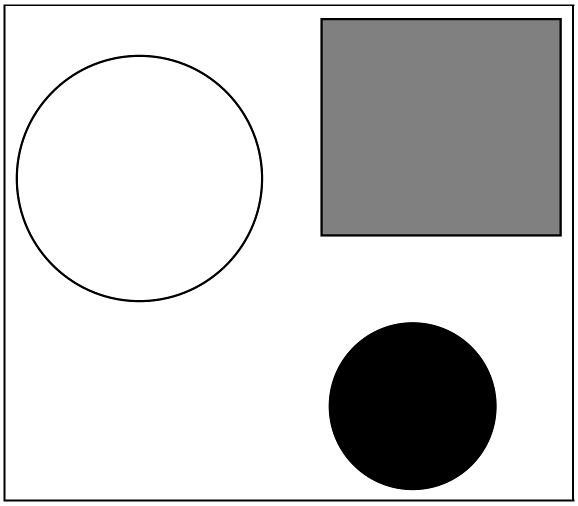

Figure 7: Example of an image

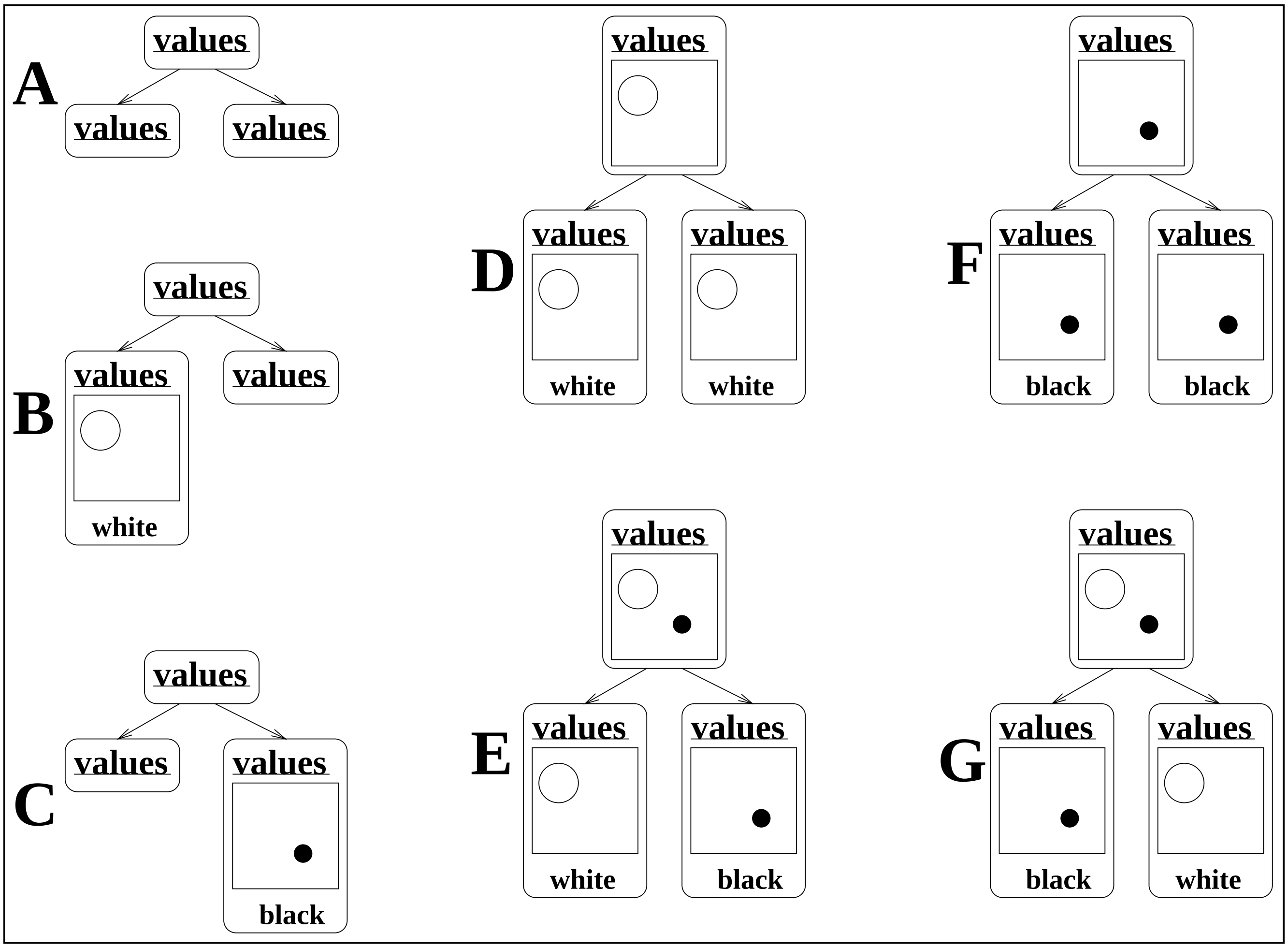

Figure 8: Partially evaluated summaries encountered by extraction manager

4.4. Extraction management for complex schemas

The extraction management for a complex schema is an enhancement of the algorithm for simple schema that allows for the existence of *set* and *or* nodes. The first observation we can make is that any schema sub-tree is also a schema. Another observation is that the extraction manager of simple schemas can work for complex schemas if any sub-tree whose root's type is *set* or *or* can be treated as a single leaf. The way we do this is by providing a feature extraction algorithm associated with such sub-trees. The full extraction manager works as follows:

1. Identify schema sub-tree whose root's type is *set* or *or* and whose every ancestor is not of type *set* or *or*.
2. Replace each sub-tree identifies in Step 1 with a leaf that has the appropriate feature extraction algorithm.
3. Run the simple extraction manager on the resultant schema.

Note that after Step 2, the resultant schema is a simple schema. When a leaf that represents a sub-tree needs to be evaluated, there are two possible algorithms to call:

***Set* algorithm.** Used when the sub-tree has a root with type *set*. The root's child is the root of a sub-tree. This smaller sub-tree is also a schema. We run the full extraction manager (recursively!) on that smaller schema, collect all the legal summaries, and create a single summary with a *set* root. Each child of the root is one of the legal summaries returned by the recursive call.

***Or* algorithm.** Used when the sub-tree has a root with type *or*. Each root's child is a root for a sub-tree schema. For each of those sub-tree schemas we call the full extraction manager (recursively!) and collect all the legal summaries from all the calls. Each of those legal summaries will become one of the the legal summaries returned by the *Or* algorithm.

4.5. Using caching and restrictions for optimizations

There are two major inefficiencies in the simple extraction manager. The example in Section 4.3 illustrates both. The first, and obvious inefficiency is that occasionally we may call the same feature extraction algorithm with the same parameters several times, and get the same result each time. In the example, every time a leaf is evaluated (steps 2,5,8) the answers returned are the same. The second inefficiency is that we did not use the restrictions for eliminating a possible answer before the restriction was fully evaluated. Step 4 of the example illustrates this well. Tree **B** is inserted into the list. We see that *leaf-1* of **B** has the value of a white circle. In addition, the restriction requires that *leaf-1* be darker than *leaf-2*. Since no value of *leaf-2* can be darker than white, it is obvious that tree **B** can't lead to a legal summary. Therefore, there is no need to insert it into the list.

We solve the first problem by using caching. Calls to feature extraction algorithms are registered in a cache and whenever we need the results of such a call we look at the cache to see if the appropriate result is already there.

The second problem can't be fully solved because it involves the optimization of expressions, which is an undecidable problem. We use a heuristic that puts limits on the set of values that attributes can receive, and still creates a legal summary. This means that instead of just evaluating the set of restrictions, we try to solve them as a system of equations. We may manage to prove that the system of equations has no solution. In that case we discard the partially evaluated tree. We may manage to put limits to the set of values that some attributes can have. This can enable us to optimize some feature extraction algorithms. For example if we have a position attribute for some leaf that has not been evaluated yet, and we can prove that the position is limited to a small portion of the image, then the feature extraction algorithm of that leaf can focus on that part of the image and ignore the rest. In our experience, the bottleneck of feature extraction is the algorithms that actually analyze the image. Thus, anything we can do to focus their effort is a very useful optimization.

5. QUERIES

The power of an IDBMS query facility is restricted mainly by the kind of data that is in the summaries. This section demonstrates the power of the PIQ query facility by using examples. Using an object-relational DBMS that can be enhanced with a few data types (such as arrays and regions), we can represent summaries in relations. Each schema can be converted to a few relations (one relation per node), while each summary can be converted into a set of tuples (one tuple per summary node). After doing that we can pose the queries is SQL. Although we did not use SQL for the examples in this section, each example query can be posed in SQL.

Sections 5.1 and 5.2 contain some examples of queries performed in PIQ on a real dataset. A more extensive study of queries on artificial datasets is avaiable.[11]

5.1. Content-based queries

The queries described in this Section were performed on a small collection of airplane photos taken from the NASA Dryden Research Aircraft Photo Archive (available at `http://www.dfrf.nasa.gov/PhotoServer`). We used a feature extraction algorithm that separates foreground from background in images to extract the airplanes. The schema has two nodes: the root is a *set* node, and its child is a leaf that is associated with the feature extraction algorithm mentioned. The reason we used a *set* node is that each photo had between 1 and 3 planes in it. The attributes we got for the leaf node are *color, shape, size, position* and *orientation*.

Figure 9 shows the result of the query: "Find images with exactly two planes. Order the result by how dark the top plane is". Figure 10 shows the result of the query: "Find images with at least two planes. Order the result by the size of the bottom plane". The results in those figures are ordered from left to right. Both queries illustrate the ability to use geometrical relationships among objects of the same image. In these cases the use of *top* and *bottom* are geometrical relationships. This is legal since the summaries contain the data for position of objects. To the best of our knowledge there is no other IDBMS that can compute such queries.

Figure 9: Result of query: "Images with two planes ordered by darkness of top plane."

Figure 10: Result of query: "Images with at least two planes ordered by size of bottom plane."

5.2. Similarity queries

A very popular kind of query for IDBMSs is the similarity query. An image is given to the system as an example and the query is: "Find the images in the database that are most similar to the example". Usually, these queries are not interpreted further by the user. This means that the system must have a predetermined notion of what "similar" means in order to perform the query. We think that the user should be allowed to present a criterion for similarity with the query. One can always put a default criterion in the system in case the user doesn't really know what he/she, wants or in case the user doesn't care. Figure 11 shows the result of a similarity query using the same airplanes collection mentioned in Section 5.1. The image in the top left corner of the figure is the example image. The rest of the images are ordered from left to right and then from top to bottom. The notion of similarity here is: "Find images with the same number of planes as in the example. The most important ordering factor is color, then shape and size to a lesser degree." Figure 12 shows the worst 5 results for the same similarity query. Note that existing IDBMSs give the user very little control of the notion of similarity. In PIQ one can use the features of a specific image as a constant for a query and pose any kind of SQL query using these constants.

Note also that in PIQ there is no difference between content based queries and similarity queries. Both kind of queries are stated in the same language (e.g. SQL). Therefore, the system uses the same query facility for both kinds of queries.

6. CONCLUSION

We have described the query by image content facility being developed in the PIQ project. The experiment presented in Section 5 and other experiments highlight the richness of the query facility one can support once a data model is used in an IDBMS. Another important contribution is a design for an IDBMS that separates Computer Vision technology from DBMS technology. The great advantage we gain is that the PIQ design can only become outdated with developments in DBMS technology while any other IDBMS we are aware of can become outdated by developments in either DBMS or Computer Vision technology.

The system design allows for several types of enhancements. If we find some sophisticated algorithm for

Figure 11: Similarity query. Best 15 results.

Figure 12: Same similarity query as the previous figure. Worst 5 results.

identifying objects of some image class, we can easily incorporate it into the system as another leaf node type. Our main extraction algorithm would invoke the sophisticated extraction algorithm when an instance of that type is to be evaluated. The querying facility doesn't need to know about the sophisticated algorithm, but uses the attributes that it extracts.

We are aware that there may be situations where the PIQ data model is not adequate. However, PIQ can always do as well as a system like QBIC, since we do not rule out feature extraction algorithms that use a user interface. There may be image domains for which specialized extraction algorithms need to be developed but we can always include those in PIQ and reduce some of the cost of developing a full DBMS.

Future work will be focused on refining the OMDL framework and developing more powerful mechanisms for defining similarity, querying and indexing. We are currently working on integrating the PIQ system in the $\mathcal{SEQ}$ object-relational database system.[10]

7. ACKNOWLEDGEMENTS

This work was supported by a David and Lucile Packard Foundation Fellowship in Science and Engineering, a

Presidential Young Investigator Award and NASA Research Grant NAGW-3921.

8. REFERENCES

1. C. Faloutsos, R. Barber, M. Flickner, J. Hafner, W. Niblack, D. Petkovic and W. Equitz, "Efficient and effective querying by image content", Journal of Intelligent Information Systems, vol. 3, no. 3, pp. 231-262, July 1994.

2. W. I. Grosky, "Toward a Data Model for Integrated Pictorial Databases," Computer Vision, Graphics, and Image Processing, vol. 25, no. 3, pp. 371–382, 1984.

3. W. I. Grosky, and R. Mehrotra, "Index-based object recognition in pictorial data management", Computer Vision, Graphics, and Image Processing, vol. 52, pp. 416–436, 1990.

4. W. I. Grosky, "Multimedia information systems," IEEE MultiMedia Magazine, vol. 1, no. 1, Spring 1994.

5. 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.

6. R. Jain, "NSF workshop on visual information management systems", SIGMOD Record, vol. 22, no. 3, pp.57-75, 1993.

7. R. Jain, R. Kasturi, and B. G. Schunck, *Machine Vision*, McGraw Hill, 1995.

8. M. Kurokawa, "An approach to retrieving images by using their pictorial features", Proc. of the ICIP, Singapore, September 1989.

9. 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.

10. P. Seshadri, M. Livny, and R. Ramakrishnan, "SEQ: a model for sequence databases," Proc. of the Int. Conf. on Data Engineering, Taiwan, 1995.

11. U. Shaft and R. Ramakrishnan, "Content-based queries in image databases", Technical Report 1309, Computer Sciences Dept., Univ. of Wisconsin-Madison, March 1996.