Data Modeling and Querying in the PIQ Image DBMS *

Uri Shaft *Raghu Ramakrishnan*

Computer Sciences Department, University of Wisconsin-Madison
1210 W. Dayton St., Madison, WI 53706
email: (uri, raghu)@cs.wisc.edu

Abstract

Image Database Management Systems (IDBMS) aim to store large collections of images, and to support efficient content-based retrieval. In this paper we explore the idea of **image data modeling** *as a tool for describing image domains, with the twofold objectives of guiding feature extraction and incorporating semantics into the extracted summaries of images. We discuss the implementation of these ideas in the* **PIQ** *image DBMS, and demonstrate that significat gains in the expressiveness of content-based queries can be achieved.*

Image Database Management Systems (IDBMS) aim to store large collections of images, and to support efficient content-based retrieval of the images. Like any other DBMS, an IDBMS has a *storage facility*, the data is arranged according to some *data model*, and it has a *query facility*.

The main IDBMS issue this paper explores is the data model and its impact on the *expressiveness* of the query language. The kind of queries we focus on are selection queries. In other words, we want to select a few images from the database, 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 so it can be used for evaluating the selection criterion. We call this extra information a *summary*. When an image is stored in the database, its summary is stored as well. Queries are executed using only the information in the summaries. 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 must be in the summary. Therefore, the expressiveness of any IDBMS query language 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 systems create the summaries automatically when an image is inserted into the database. Other 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 (which includes *feature extraction*).

*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.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

1.1 The Case for Data Modeling

Existing IDBMSs typically 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.

Images in a given specialized domain have much in common. This commonality is reflected in the type of information put in summaries. For example, the VIMS (or VIMSYS) [1, 3] 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 this specific image domain as well. These kinds of systems have high development and maintenance costs. Every change to the system, such as including a new feature or a new type of query, needs expensive code development. Supporting a different image domain can only be achieved by building 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 content of the information in the summaries. QBIC [4, 2] 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 assist in identifying regions of the image (called *objects*). For each object, the system extracts its color histogram, its shape, and sometimes its texture. Any query language used in such a system has limited expressiveness because there is not much information in 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. Data modeling is used for describing the contents of images of a specific image domain. In [7, 6] we show how the PIQ data model is used for automated feature extraction. In this paper we demonstrate the expressiveness of a query facility built for the PIQ data model, and the scalability of PIQ.

2 The PIQ Data Model

In order to describe image collections to the IDBMS we need a *data description language* (DDL). This language has constructs that enable us to describe the common features of images in the same collection. Each valid description in the language is referred to as a *schema*. The collection of all valid schemas for the language is the *data model* of that language.

We describe the PIQ data model by demonstrating how schemas are defined using the PIQ DDL. For every image in a collection (with a specific schema) the system creates a summary. That summary's data type is defined by the schema. For example, suppose the schema for a collection of faces tells the system that the information about eye-color is stored in the summary. Then, for every image, the summary of the image will contain a value for eye-color (e.g. *blue*). The values of summaries are generated by an automated feature extraction facility (see [7, 6]).

It is important to note a fundamental aspect of PIQ: the DBMS does feature extraction and organizes the resulting summary information, in order to store, index and query this information efficiently. This results in a central tradeoff in the design of the data model: the richer the data model, the easier it is to describe a collection of images, but the harder it is to do appropriate feature extraction and summary organization.

2.1 What Is a PIQ Schema?

A schema is structured as a tree. Each node in the tree has a name and specific type. Furthermore, each node represents an *object* in the image. An object is a region of an image (i.e., a set of pixels).

The system maintains a set of *leaf node types*. Each leaf node type is associated with a feature extraction algorithm. For example, if we have an algorithm for extracting circular objects from an image, we can have a leaf node type for it (called *Circle*). A schema node with type *Circle* refers to a circular object in the image. The set of leaf node types is dynamic. Whenever we have some feature extraction algorithm we want to use, we can define a new leaf node type for it. Thus, the system's data model can be extended. For more details on DDL guided feature extraction see [7, 6].

Internal nodes of a schema tree represent operations done on the objects associated with the children of that node. Those types are:

Union. The object referred to by a node whose type is *Union* is the union of the objects referred to by the node's children. Note that objects are sets of pixels, so a union of objects is a union of sets of pixels. For example, a schema for a face may have a *Union* node whose children represent *eyes, nose, mouth, hair* etc. Each child node represents a region in the image that corresponds to a part of the face. The *Union* node represents the region of the image that represents the entire face.

Intersection and *Difference.* Defined the same as *Union*, differing only in the kind of set operation done on the objects the node's children refer to.

Set. A *Set* schema node has exactly one child. The child node may have many valid objects to refer to in a single image. The set node indicates that all those objects need to be considered. The object associated with the *Set* node is the union of all the objects that can be associated with its child.

Or. An *Or* schema has more than one child. Images in a collection may have valid objects in them only for a few of the node's children. Any object associated with a child of an *Or* node can be associated with the *Or* node itself. For example, the *Or* node of a schema may represent some *vehicle*, whereas its children may represent a *car*, a *bike*, a *bus* etc. If an image in the collection features a car, the object associated with the *Or* node is that car. Similarly, if an image in the collection features a bike or a bus, the object associated with the *Or* node is that bike or bus.

Each schema node contains attributes. Those attributes, like attributes in a relational data model, have names and types. PIQ maintains a set of types that can be used for defining attributes. This set may be extended by inserting new types into the system. However, there are some types that are always present in PIQ. These are: **Number**, **String**, **Boolean**, and **Region**. A **Region** type is a set of pixels. Each node has at least one attribute called *primary-region* (of type **Region**). This attribute's value is the object that node refers to. We also have an **Array** type for each of the above types (and for all other types inserted into the system). The array may have any number of dimensions and any size for each dimension.

The DDL statements defining schemas are:

- "**schema** *name* **has type** *type*." This declares a schema with a single node. The node has the name specified in *name* and the leaf node type specified in *type*.
- "**schema** *name* **is** *operator* **of** *list-of-names*." This declares a schema whose root node's name is *name*, and its type is *operator* (either *Union, Intersection, Difference* or *Or*). Its children are the schemas whose names are listed in *list-of-names*. The list of names actually lists pairs of names in the format: "prefix-1 : schema-name-1, ... , prefix-k : schema-name-k"

  We want the names of nodes in a schema to be unique. Since some of the names in the *list-of-names* can be duplicates, we make them unique by adding prefixes. The prefix is optional if there

is no duplication of names. For example, suppose we have a schemas "A", "B". Suppose we want schema "C" to represent two objects of "A" and one of "B". The statement will be:
"**schema** C **is** *Union* **of** first : A, second : A , B."

- "**schema** *name* **is** *Set* **of** *name1*". This declares a schema whose root node name's is *name*, and its type is *Set*. The child of that root node is the schema named *name1*.

Note that we identify a schema by the name of its root. The DDL statement for attribute definition is: "**attribute** *attr-name* **of node** *node-name* **of schema** *schema-name* **is** *expression*." This means that the node named *node-name* of the schema named *schema-name* has an attribute called *attr-name*. *expression* defines both the type of the attribute, and the method for computing it. More details about expressions are presented in Section 2.3.

2.2 What Is a PIQ Summary?

For each image we create a summary according to the schema of the image's collection. The summary is a tree structure that is very similar to the schema tree. Note that every sub-tree of a schema tree is also a schema. We build a summary tree for a specific image and schema using the following rules:

- Suppose the schema tree has only one node. The summary tree has only one node as well. For each attribute in the schema tree there is a value in the summary tree. That value is taken from the image. In particular, the value of the attribute *primary-region* has a **Region** type and refers to a set of pixels in the image.
- Suppose the schema tree has a root node of type *Union*. Each child of that root node is a root of a sub-tree (which is a schema). For each such sub-tree we construct a summary tree. The summary tree for the entire schema will have a root whose children are the roots of the summary trees constructed for the sub-tree schemas. Again, for each attribute in the schema root node, the summary root node will have a value. In particular, the value of *primary-region* will be the union of the values of *primary-region* of the children of the summary root node.
- Suppose the schema tree has a root node of type *Intersection* or *Difference*. We treat it the same as the *Union* case except that the rule for computing the value of *primary-region* changes to reflect the different set operation.
- Suppose the schema has a root node of type *Set*. The child of the root node is the root of a sub-tree (also a schema). We construct as many valid summary trees as possible for that sub-tree. The summary tree for the entire schema will have a root whose children are the roots of all the summary trees constructed earlier. Again, we have values for all attributes of the root node and the value of *primary-region* of the root node is the union of all values of *primary-region*'s in its children.
- Suppose a schema S has a root node of type *Or*. Child i of S is the root of a schema S_i. For each S_i we construct summary trees. Any summary tree that is valid for S_i is also a valid summary tree for S.

2.3 Attributes and Expressions

Some attributes are defined by the user using the attribute definition statement described in Section 2.1. Other attributes are defined automatically. For each node in the schema tree the attribute *primary-region* is automatically defined. For each leaf node there may be other attributes that are automatically defined. This depends on the type of that leaf node. For example: a leaf node of type *Circle* may have attributes for the *position* of the center of the circle and for its *radius*.

When we construct a summary for an image with a specific schema, the automatically defined attributes get values from the feature extraction algorithm. For example, the position and radius of an object of a *Circle* node are determined by the algorithm that searches for the circle in the image. For leaf nodes, the extraction algorithm finds the value of *primary-region*. For internal nodes, we have rules that determine how to compute the value of *primary-region* from the values in its children (see Section 2.1 and 2.2).

The rules for computing the other attributes are given with their definition by an *expression*. An atomic expression is either a constant or a reference to a previously defined attribute. We construct more complex expressions from atomic expressions using function. When we construct a summary, the expression given for an attribute is a function whose variables are other attributes. When we evaluate the expression, we substitute the attribute references with their actual value to get the value for another attribute.

Expressions may be used to enforce some restrictions on a summary. The DDL statement is:
"**schema** *name* **is restricted by** *bool-expression*." This states that *bool-expression* is a boolean expression over the attributes of the schema *name*. This expression is called a *restriction*. A summary is valid only if it was constructed according to the rules in Section 2.2, and all the schema's restrictions evaluate to **true**.

An example for the use of a restriction is enforcing spatial relationships. Suppose we have a schema describing a face. A part of the schema can be stated as
"**schema** eyes **is** *Union* **of** left-eye : eye, right-eye : eye."
We assume that the schema for "eye" has position attributes (*x-position, y-position*). If we do not use restrictions and an image has two eyes in it then we have a problem. Each eye object in the image is a valid object for both "left-eye" and "right-eye". Therefore, we have four valid summaries for that schema. Obviously, we consider only one of the options as desirable. We can use the restriction
"**schema** eyes **is restricted by** x-position(left-eye) < x-position(right-eye)."
The restriction claims that the left eye is positioned to the left of the right eye. Now, only the proper summary tree is valid.

2.4 An Example of a Schema and a Summary

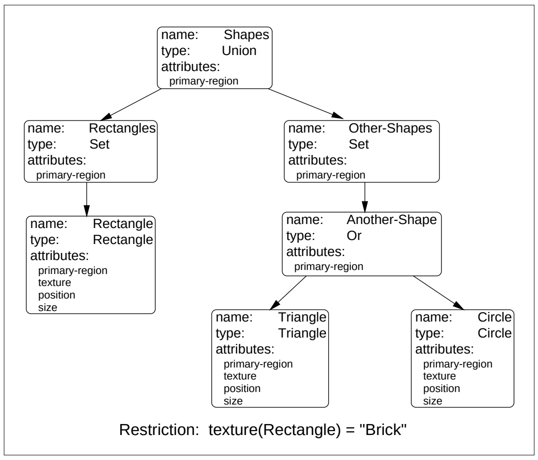

Figure 1: Example of schema.

Figure 1 is an example for a schema tree. The DDL definition of the schema is:
"**schema** Rectangle **has type** *Rectangle*."
"**schema** Rectangle **is restricted by** texture(rectangle) = 'BRICK'."
"**schema** Circle **has type** *Circle*."
"**schema** Triangle **has type** *Triangle*."
"**schema** Rectangles **is** *Set* **of** Rectangle."
"**schema** Another-Shape **is** *Or* **of** Circle, Triangle."
"**schema** Other-Shapes **is** *Set* **of** Another-Shape."
"**schema** Shapes **is** *Union* **of** Rectangles, Other-Shapes."
All the attributes are automatically specified in this case. This includes the *size, texture* and *position* of the leaf nodes. Figure 2 shows an example of an image. Figure 3 shows a valid summary for that image with the schema in Figure 1. The sub-images shown in each node of the summary tree in Figure 3 are illustrations representing the value of *primary-region* of that node. The values of other attributes are not shown. Note that there are only two rectangles in the summary instead of the four in the image. This happens because the schema has a restriction stating that the texture of a rectangle should be 'BRICK'. Only the two rectangles with a brick texture are presented in the summary.

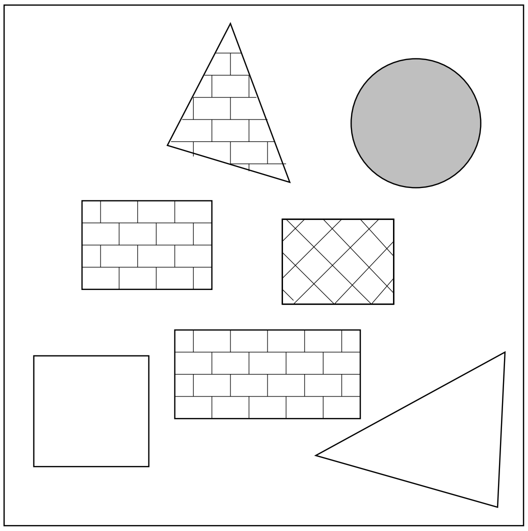

Figure 2: Example of image.

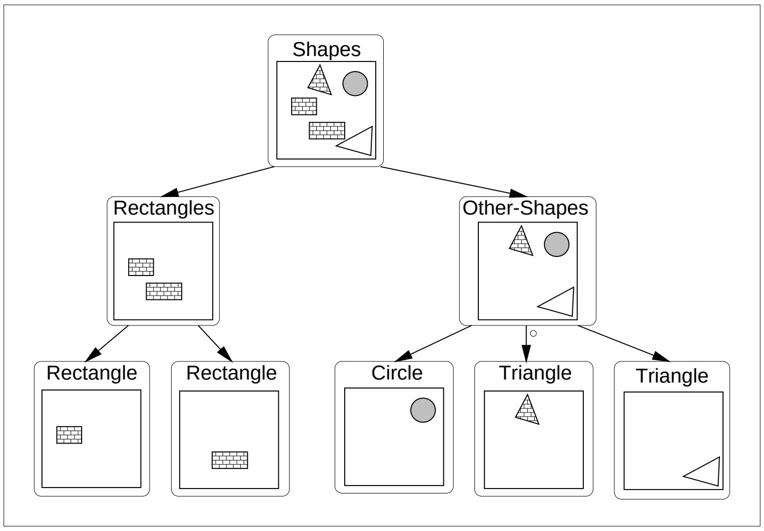

Figure 3: Example of summary.

3 Expressiveness of Queries

One can only ask queries about information that exists in the database, and is captured by the DBMS's data model. This point seems trivial, but it is the cause of major limitations to the leading IDBMSs such as QBIC [4, 2] and Photobook [5].

In the case of PIQ, one can create schemas, thereby determining what information in the images is important. In this case the true limit to the expressiveness of queries is the query language and query processing facility. Two main issues are still under research in the PIQ project. First, the issue of developing a good query language for the PIQ data model. Second, how to optimize and process queries efficiently when the database consists of summary trees.

3.1 Representing Summary Trees in an Object-Relational Model

We can represent a schema as a set of relations, and a summary as a set of tuples in those relations, using the object-relational data model. Once we do that, queries can be posed in any object-relational query language (e.g. SQL variants). The main advantage is that the PIQ system can use a conventional DBMS for storing summaries and processing queries.

The representation of a schema as a set of relations is based on the idea that a tuple can represent a summary node, and the association between a parent node and its child can be captured by using foreign keys. For each node in the schema tree we create a relation with a unique name (which is the node's name). The attributes for each relation are the attributes of the schema node with the following additions:

- Image ID. An object ID for the image.
- Tuple ID. An object ID for the specific tuple.
- Parent ID (for every node except the root). For identifying which tuple represents the parent of this tuple in the summary tree.

Since we use object IDs and various data types that are not in the standard relational data model, we need an object-relational data model (for object IDs) with the ability to store large uninterpreted blobs (for arrays, regions and images).

Each node in a summary tree is converted into a tuple. The relation used for that tuple is the relation created for the schema node associated with the summary tree node. The summary tree node contains the values for the attributes, so those values are used for constructing the tuple. Furthermore, we get a unique object ID for the tuple and fill in the object IDs for the image and for the parent of this tuple.

We may not want all of the attributes to be used by the query facility. Some attributes may be computed only to be used for restrictions, but once the restrictions are evaluated, the values are not important for us. We can declare some attributes as *temporaries*. When creating the relations and tuples (or when storing the summary trees in any other way) we ignore those attributes. This can save a lot a storage space if the unwanted attributes are represented by large blobs such as arrays and regions.

3.2 Example of an Experiment with Airplane Pictures

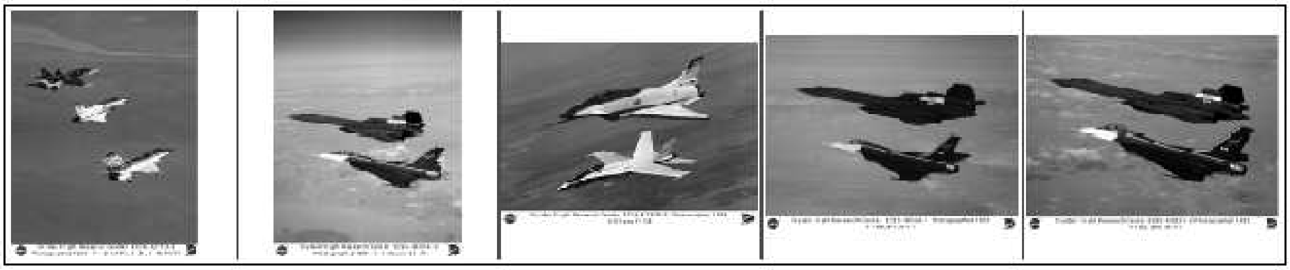

Figure 4: Result of query: “Images with at least two planes ordered by size of bottom plane.”

Figure 5: Similarity query. Best 5 results.

Figure 6: Same similarity query as in Figure 5. Worst 5 results.

The data set used in the experiment contains pictures of airplanes from the NASA Dryden Research Aircraft Photo Archive (available at `http://www.dfrf.nasa.gov/PhotoServer`). A very simple extraction algorithm that distinguishes between background and foreground was used for finding individual airplanes in the images. We created a schema leaf node type called *airplane* and associated it

with that feature extraction algorithm. The attributes we get from the algorithm are *size, color, shape, orientation* and *position*. The schema is:
"**schema** airplane **has type** *airplane*."
"**schema** airplanes **is** *Set* **of** airplane."
We discarded the *primary-region* attributes. *shape* was represented by some algebraic moments of the *primary-region*. *color* is a color histogram of the airplane. *position* was broken down into the X and Y coordinates of the center of mass of *primary-region*. *orientation* is the angle that the main axis of *primary-region* has with the X axis. *size* is the number of pixels in *primary-region*.

All example queries in this section are expressible in SQL over summary relations. The queries are not easy to write in SQL. This is a major incentive to create a better query language.

Figure 4 shows the results of the query: "Find images with at least two planes ordered by size of bottom plane." The order of results in the figure is from left to right. This query clearly demonstrates our ability to utilize spatial relationships (i.e., bottom) in conjunction with some other properties of objects (in this case *size*).

Figures 5 and 6 show the result of a similarity query. An example of an image is given (the image on the upper left in Figure 5). We want to find images that are similar to it according to a specified similarity metric. The metric in this case is: images with the same number of airplanes in them as the example image. The ordering of the images is: first order by how close the color of the airplane is, then order by the size and shape and orientation. The results in Figure 5 are ordered from left to right.

Note that most systems have built-in notions of similarity. It may be useful to reduce the amount of work needed for specifying a query by using a default similarity metric. However, it limits the user's ability to specify his/her requirements for the query.

4 Conclusions and Future Work

We described the idea of image data modeling and its application in the PIQ IDBMS. We showed that a very powerful query facility is feasible when using a non-trivial data model.

The PIQ system is extensible and scalable. The system's data model can be extended by defining new leaf node types for schemas. This enables the user to have automated feature extraction for many image domains by including specialized software developed by the Computer Vision community. Automated feature extraction makes the system scalable: the number of images inserted into the database can be very large. The reason is that we do not need to have user's input on a per-image basis, and summaries can be indexed and queried using scalable DBMS techniques.

Our future work focuses on developing a good query facility. This includes a query language that is both powerful and easy to use. It also includes solving the problem of query optimization and processing for summary trees.

References

[1] Jefferey R. Bach, Santanu Paul and Ramesh Jain. A visual information management system for the interactive retrieval of faces. *IEEE Transactions on Knowledge and Data Engineering,* 5(4):619–628, August 1993.

[2] C. Faloutsos et al. Efficient and Effective Querying by Image Content. *Journal of intelligent information systems,* 3(3):231–262, 1994.

[3] Amarnath Gupta, Terry E. Weymouth and Ramesh Jain. Semantic Queries with Pictures : The VIMSYS Model. *17th International Conference on Very Large Data Bases,* Barcelona, Spain, September 1991.

[4] W. Niblack et al. The QBIC project: Querying images by content using color, texture, and shape. In Wayne Niblack, editor, *SPIE vol. 1908: Storage and Retrieval for Image and Video Databases,* pages 173–187, San Jose, CA, February 1993.

[5] A. Pentland, R. W. Pickard and S. Scaroff. Photobook: Tools for content-based manipulation of image databases. In Wayne Niblack and Ramesh C. Jain, editors, *SPIE vol. 2185: Storage and Retrieval for Image and Video Databases II,* pages 34–47, San Jose, CA, February 1994.

[6] Uri Shaft and Raghu Ramakrishnan. Content-Based Queries in Image Databases. Technical report 1309, Computer Sciences Department, University of Wisconsin-Madison, March 1996.

[7] Uri Shaft and Raghu Ramakrishnan. Data modeling and feature extraction management in image databases. In C.-C. J. Kuo, editor, *SPIE volume 2916: Multimedia Storage and Archiving Systems,* Boston, MA, November 1996.