Image database management systems design considerations algorithms and architecture
Nes, N.J.

Citation for published version (APA):
Nes, N. J. (2000). Image database management systems design considerations algorithms and architecture

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 2

Image Databases

In this chapter we explain what kind of database management system appropriate is to construct and manage image collections. In particular, we introduce a requirement list for assessing a DBMS for deployment in this area. Subsequently we introduce our experimentation platform, the Monet DBMS[8], in more detail. We conclude with a short evaluation of commercial database and image retrieval systems, including Monet, against our requirement list.

2.1 Multi-Media Database Systems

The research field of image database systems can be characterized as a subset of the field of multi-media database systems. Multi-media database management systems are software systems dealing with data both with well-and ill-defined semantics. Data with well-defined semantics is usually called structured data, examples are numbers and formatted records. Data with ill-defined semantics is usually called unstructured data, such as text, audio, image and video. Example application areas for Multi-Media database management systems are Digital Libraries, Video on demand systems and news archives.

The combination of multiple media imposes additional requirements on the DBMS, because they typically can not be dealt with in isolation. The combination of multiple media gives additional valuable information and may therefore be easier to understand. Another important issue is quality of service, i.e. the clients of the multi-media database may not be able to handle a fast multi-media stream.

In this thesis, we restrict our research to a single media type, namely still images from the context of database research. Although this greatly reduces the domain, it makes the project manageable within the limited resources given.

We believe that progress obtained in the design of image database man-
agement systems carries over to the subsequent extensions to deal with other media types. The trajectory followed results in a requirements list for image databases. Many of these requirements also hold for multi-media databases at large.

The basic functionality to support Image Database Management Systems centers around the following issues:

- Image Storage
- Image Operations
- Derived Image Data
- Image Semantics
- Image Queries

These issues are elaborated upon in the next sections.

2.1.1 Image Storage

The first requirement for an Image Database Management Systems (IDBMS) is, of course, the ability to store images. The image processing community uses many image file formats. Well known examples include: TIFF, GIF, PNG, JPEG, PPM, BMP, PICT and XPM. These formats have been designed for specific applications or as an attempt to set a standard for image data exchange, however, none is general enough to support all image types. To illustrate, many of these file formats, such as GIF, PNG, JPEG, PICT and XPM were designed for viewable images, i.e. mono-chrome, gray scale and color images. This restricted domain makes them not suited for storing images in an IDBMS. For example, satellite images and stereo images can not be stored using these formats. Moreover, the image processing community still lacks a standard image file format, i.e. there does not exist an (extensible) data model to reason about images. As a result, we are challenged to come up with an appropriate image representation to cater for all intended use. For such an image representation scheme, the DBMS should provide an abstract data type (ADT) facility which provides for:

1. Extensible representation and algebra
2. Multiple views on components to support segmentation.
3. Cheap (de)compress functions.
4. Easy Input/Output Facilities
Extensible Representation and Algebra

In our search for a flexible image representation scheme, we came across the image algebra [90]. Before we continue with the discussion of our image representation of choice we give a synopsis of its concepts.

The image algebra [90] is a mathematical theory focused on the analysis and transformation of digital images. The main goal was to define a comprehensive and unified theory of image transformations, image analysis and image understanding.

The image algebra is defined as a heterogeneous algebra. Such an algebra is defined as follows:

**Definition.** An algebra \( A \) is a pair \( A = (\mathcal{F}, \mathcal{O}) \), where

1. \( \mathcal{F} = \{ F_\lambda \}_{\lambda \in \Lambda} \) is a family of non-empty sets of different types of elements and the subscripts \( \lambda \) are members of some indexing set \( \Lambda \), and
2. \( \mathcal{O} = \{ O_k \}_{k \in K} \) is a set of finitary operations (for some indexing set \( K \)), where each \( O_k \in \mathcal{O} \) is a mapping of the Cartesian product of some of the \( F_\lambda \)'s to another.

The elements \( F_\lambda \) of \( \mathcal{F} \) are called the sets of operands of \( A \), and the elements \( O_k \in \mathcal{O} \) are called the operators of (or operations on) \( A \).

An Algebra \( A \) is called a *homogeneous* or single valued algebra, if \( \mathcal{F} \) contains only one element i.e., \( \mathcal{F} = \{ F \} \), otherwise \( A \) is called a *heterogeneous* or *many valued* algebra.

The Image algebra [90] defines an image as follows.

**Definition.** Let \( F \) be a homogeneous algebra and \( X \) a topological space. An \( F \)-valued image on \( X \) is any element of \( F^X \).

Given an \( F \)-Valued image \( a \in F^X \), then \( \mathcal{F} \) is called the set of *possible range values of \( a \)* and \( X \) the *spatial domain of \( a \)*.

It is often convenient to let the "graph" of an image \( a \in F^X \) represent \( a \). The graph of an image is also referred to as the *data structure* representation of the image. Given the data structure representation \( a = \{ (x, a(x)) : x \in X \} \), then an element \( (x, a(x)) \) of the data structure is called a *picture element* or *pixel*. The first coordinate \( x \) of a pixel is called the *pixel location* or *image point*, and the second coordinate \( a(x) \) is called the *pixel value* of \( a \) at location \( x \).

Many digital images require the topological space \( X \) to be a subspace of \( \mathbb{Z}^2 \). A sequence of images can be modeled using \( X = \mathbb{Z}^3 \), with \( x \in X \) of the form \( x = (x, y, t) \), where the first coordinates \( (x, y) \) denote spatial location and where \( t \) denotes a time variable.

The value set \( F \) can be replaced with \( \mathbb{Z}_{2k} \) or with \( (\mathbb{Z}_{2k}, \mathbb{Z}_{2l}, \mathbb{Z}_{2m}) \). The first provides us with digital integer-valued images of \( k \)-bits. The second provides us with digital vector-valued images.
The image algebra defines a logical image representation, we will follow this definition of an image. To use a logical representation calls for an implementation, i.e. a physical representation. We elaborate more on our physical image representation including performance analysis in Chapter 3.

**Color Models**

Mono-chrome and grayscale images can easily be represented using \( F = 0,1 \) and \( F = \mathbb{Z} \). More problems arise when we try to model color images. Physically color is a composition of light signals of different wavelengths. These signals are discretized and represented using so called color models. Many such models exist and each has its strong and weak points, see [45]. Examples are RGB, CMY, HSV, HSI, \( L^*a^*b^* \), XYZ, UVW and xyz. The RGB and CMY models are used for output devices, such as displays and printers. XYZ is a color model which is device independent. The \( L^*a^*b^* \) and \( L^*u^*v^* \) are perceptually uniform, i.e. distances in these spaces correspond to human perceptual differences. The HSV and HSI are intuitive to the user, Hue is the bare color, Saturation is the infection of this hue with other colors, and I is the intensity of the color.

The different models have different applications, RGB and CMY color models are used for output. For many image operations other color models are more appropriate. Therefore, conversion primitives from one color model into another is a pre-requisite for any image analysis system.

An additional requirement for color images is to store the color model information. This can be done directly, using a separate table, or implicitly, using a special pixel value type.

From a semantic point of view there is no reason to differentiate among monochrome, grayscale or color images. They are all images. They can all be stored using (the more general) color images. We therefore like to treat all images equal. Unfortunately the theory for color images is less evolved then for gray and monochrome images. This makes explicit type coercion necessary.

**Image Segments**

Segments represent interesting parts of an image. Many image processing applications first reduce their problem by looking only at the segment or region of interest within an image. This pre-processing can reduce the resource requirements substantially. This leads to the image database requirement of segment representation and segment construction.

As a consequence images can also be seen as a collection of disjoint segments. Each segment would contain different, but interesting information. This alternative view on images and segments indicates the concepts are closely related. To fully exploit this resemblance the concepts should be
represented using the same logical representation. Image operations should therefore also work on segments without additional work.

Our image representation, a mapping from a spatial domain $X$ to a domain of range values $F$, fits both concepts of images and segments. Operations to segment an image would therefore easily return new images.

**Image Compression**

An image database will store lots of images, taking up lots of disk space. Compression techniques to reduce storage can be used. Two alternatives exist, we can compress each image in isolation or we can compress a set of images together. The second will result in better overall reduction, but the first makes access to a single image still reasonably fast. This choice between performance and storage will depend on the applications using the database. Therefore, the database administrator should be able to make this choice. We therefore support compression on the image and table level.

We aim with image set compression on sets of images that contain very similar images. Therefore, we envision of set image compression using a base image and image differences. Each image in the set is stored based on a base image. The differences are computed and stored. Each resulting image can then again be compressed using the single image compression techniques. A 3D wavelet-based approach over the set might be effective.

**Input/Output Facilities**

Another important requirement for an image ADT are easy input and output facilities. We support input from and output to various image file formats. This makes it possible to access the large number of images currently stored in these formats. Support for output into these types makes it possible to reuse the huge amount of existing software. In our image input/output module we support conversion of our image ADT to JPEG, GIF, TIFF and PPM.

**2.1.2 Image Operations**

From a database perspective having a single image data type in the query language is ideal. It greatly simplifies query construction and optimization. This single image data type should come with a complete set of operations, such that by combining operations all relevant logical image operations can be performed.

The Image Algebra [90] defines a complete set of operations for all image types. It consists of operation on images and templates. All operations use basic mathematical functions on the spatial domain or range value domain. Therefore, the semantics are properly determined by the semantics of these basic operations.
For some image types considered in practice, however, most of the mathematical operations have no clear semantics. For example mathematical morphological operations on multi channel images, such as color images, have no proper theoretical background yet. Also there is no consensus on the linear filter theory for these images. In these directions progress is made as can be read in [21], [107] and [37].

Although a complete set of operations is known for the individual image types, their results are usually very different. A histogram of a color image is different from a histogram of a gray scale image. This complicates any query language greatly.

Dynamic resolution of the result types would make query optimization impossible. A query optimizer needs to know the result types of all operations to be able to make a complete query graph. With a single image type this is impossible. Therefore, we need to express an image as a complex data type which is uniquely described by the type of its spatial domain and its range domain. This requires a polymorphic image data type.

Operations

Following the image algebra [90] we can classify the operations on images in the following categories.

- Restriction and Extension
- Induced pixel operations
- Reduction Operations
- Spatial Operations
- Template Image Operations
- Template Operations

The image algebra defines two operations, \( \text{domain}(a) \) and \( \text{range}(a) \), to extract point sets and value sets from a particular image, \( a \). The domain of an image is the set of points expressing the spatial extent of the image. The range of an image are the range values of an image, for example the gray values or colors of the image.

Image Restriction removes pixels from an image. This can be realized through both the spatial domain and range value domain. Given a set of points, an image can be reduced to include only these positions. Or given a set of range values, an image can be restricted to only include these values. The inverse of image restriction is image extension. This operation adds pixels to an image, which it not yet contains. A combination of image restriction and extension could be used for replacing parts of an image, for example the blue screen replacement often seen in video editing.
2.1. MULTI-MEDIA DATABASE SYSTEMS

Induced pixel operations are unary and binary operations on individual image pixels. If the operation $\gamma$ is a binary operation on $F$, then $\gamma$ induces a binary operation $\gamma$ on $F^X$ defined as follows:

Let $a, b \in F^X$, then

$$a \gamma b = \{(x, c(x)) : c(x) = a(x)\gamma b(x), x \in X\}$$

If the operation $\theta$ is a unary operation on $F$, then $\theta$ induces a unary operation, also called $\theta$ on $F^X$ defined as follows:

Let $a \in F^X$, then

$$\theta(a) = \{(x, c(x)) : c(x) = \theta a(x), x \in X\}$$

See table 2.1 for a list of unary and binary image operations.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$-a$</td>
<td>image negation</td>
</tr>
<tr>
<td>$\neg a$</td>
<td>logical image negation</td>
</tr>
<tr>
<td>$\sin(a)$</td>
<td>sinus image</td>
</tr>
<tr>
<td>$a + b$</td>
<td>image addition</td>
</tr>
<tr>
<td>$a - b$</td>
<td>image subtraction</td>
</tr>
<tr>
<td>$a \times b$</td>
<td>image multiplication</td>
</tr>
<tr>
<td>$a/b$</td>
<td>image division</td>
</tr>
<tr>
<td>$a \lor b$</td>
<td>image minimum</td>
</tr>
<tr>
<td>$a \land b$</td>
<td>image maximum</td>
</tr>
<tr>
<td>$a &lt; b$</td>
<td>image smaller than</td>
</tr>
<tr>
<td>$a \geq b$</td>
<td>image larger equal than</td>
</tr>
</tbody>
</table>

Table 2.1: Example Induced Pixel Operations

When for one of the operands of the binary operation a constant value is used we get a scalar operation. If the operation $\gamma$ is a binary operation on $F$, then $\gamma$ induces a binary scalar operation $\gamma$ on $F^X$ defined as follows:

Let $k \in F$ and $a \in F^X$, then

$$a \gamma k = \{(x, c(x)) : c(x) = a(x)\gamma k, x \in X\}$$

$$k \gamma a = \{(x, c(x)) : c(x) = k\gamma a(x), x \in X\}$$

The global reduction operations reduces an image into a single complex value. Let operation $\gamma$ be a binary operation on $F$, then $\gamma$ induces a unary operation

$$\Gamma : F^X \rightarrow F$$

called the global reduce operation, which is defined as

$$\Gamma a = \Gamma_x a(x) = \Gamma_{k=1}^n a(x_k) = a(x_1)\gamma a(x_2)\gamma \ldots \gamma a(x_n).$$
Simple examples of reduce functions are addition, multiplication, minimum and maximum of pixels.

Spatial operations transform images based on the point set, which represents the topology of the image. Examples of spatial transforms are image translation, rotation and reflection. Also the family of affine transforms are spatial operations. Let \( f : Y \rightarrow X \) and \( a \in F^X \), then we define the induced image \( a \circ f \in F^Y \) by:

\[
a \circ f = \{(y, a(f(y))): y \in Y\}
\]

Template image operations transform images based on templates. A template is an image whose pixel values are images (functions). Formally defined as follows:

**Definition.** A template \( t \) is an \( F \)-valued template from \( Y \) to \( X \) is a function \( t : Y \rightarrow F^X \). Thus, \( t \in (F^X)^Y \) and \( t \) is an \( F^X \)-valued image on \( Y \).

For notational convenience we define \( t_y \equiv t(y) \ \forall \ y \in Y \). The pixel values \( t_y(x) \) of this image are called the *weights* of the template at point \( y \).

We can divide templates into two categories, the translation variant and invariant templates. A template is called translation invariant when for each triple \( x, y, z \in X \) we have \( t_y(x) = t_{y+z}(x + z) \). Many of the translation invariant templates can be defined pictorially. See Figure 2.1.2 for an example pictorial definition of a template.

![Figure 2.1: Example Template](image)

A template image operation performs an induced pixel operation for each image, \( t_y \), in the template. Each resulting image is reduced using a global image reduce operation. The resulting value will be the pixel value at the
pixel position $y$ in the resulting image. Formally, let template $t \in (G^X)^Y$, image $a \in E^X$, and $a \bigcirc t_y \in F^X$ and $\Gamma(a \bigcirc t_y) \in F$. It follows that the binary operations $\bigcirc$ and $\gamma$ induce a binary operation

$$\gamma : E^X \ast (G^X)^Y \rightarrow F^Y,$$

where

$$b = a \bigcirc t \in F^Y$$

is defined by

$$b(y) = \Gamma(a \bigcirc t_y) = \Gamma_{x \in X}(a(x) \bigcirc t_y(x))$$

$$= (a(x_1) \bigcirc t_y(x_1))\gamma(a(x_2) \bigcirc t_y(x_2))\gamma \ldots \gamma(a(x_n) \bigcirc t_y(x_n)).$$

This is the right product of image $a$ with template $t$ also the left product of $a$ with template $t$ exists.

Example template operations are image convolution and the basic morphological operations, dilation and erosion. In case of the convolution, the original image is multiplied with each template image. Each resulting image is summed to a single value.

A histogram for an image $a(x)$ can be calculated uses template operations. A template, $t \rightarrow (N^Y)^X$, used together with a function,

$$t(a)_x(j) = \begin{cases} 1 & \text{if } a(x) = j \\ 0 & \text{otherwise} \end{cases}$$

transform an image into a set of images. Using an image reduce operation which sums each image the histogram is calculated.

Templates are just a special kind of image. This assures all image operations are also defined on templates. We can restrict and extend templates. The induced operations on templates map each operation on each pixel, i.e. an image. So template addition maps to image addition for each image in the template and the image addition will map to pixel addition for each value in these images. Those template operations make the image algebra such a powerful framework.

**Requirements Summary**

The requirements involving image primitives are summarized as follows.

1. Requires support for a polymorphic image type.

2. Requires support for the complete set of image operations as specified by the image algebra on this image type.
CHAPTER 2. IMAGE DATABASES

2.1.3 Features

Another basic requirement for image databases is storing and managing derived data. In the image processing domain derived data types are called features. These features are used to describe, interpret or understand the image data. The features derived from the complete image are usually called global features as opposed to the local features calculated at a region, segment or single point within the image.

Over the years, a large collection of image features has been proposed, which can be grouped into a few categories: color, texture, frequency analysis, and shape features. Some examples in each category are given below.

**Color Features** Average color, dominant color, color histogram, color distribution and color variance.

**Texture Features** Dominate angle, object granularity.

**Frequency Features** wavelet and discrete Fourier transforms

**Shape Features** Circularity, eccentricity, bounding energy, boundary, moment features.

Example single pixel value features include intensity, color and reflectivity. Over a region or segment a histogram of the pixel values can be calculated, this is called a complex feature. From this histogram many features can be derived, like dispersion, mean, variance, mean square value and average energy. A second order histogram, a histogram of all pairs of pixel values, is also used often [28].

Texture is observed in the structural patterns of surfaces of objects such as wood, grain, sand, grass and cloth. Textures are usually described using a repetition of basic texture elements. Natural textures have usually random repetitions and changing texture elements. Artificial textures are often deterministic and periodic. Often the textures are described using measures for the coarseness of the basic textures, periodic and orientation. Many other texture features exists. Examples can be found in [49, 60] and [61].

Examples of frequency features are based on the discrete Fourier and wavelet transforms. The Discrete Fourier Transform (DFT) results in a decomposition of the image in the frequencies of cosine functions. The frequencies tell us something about the content of the image. A high number of high frequency cosine functions indicates many small changes in the image. Low frequencies would indicate a rather smooth image.

The wavelet transform [31, 104, 114] analyses an image at multiple scales. It recursively decomposes an image using a low and a high band filters. This gives a similar description of the image as a Fourier transform, but with an additional component of locality. The Fourier transform only globally
decomposes a signal in its frequencies, i.e. no locality is preserved. The wavelet transform uses small filters with a limited size so it preserves locality.

Shape features can be divided into two categories, geometrical and moment features. Examples of geometrical features are perimeter, area, max-min radii and eccentricity, corners, roundness, bending energy, holes, Euler number and symmetry. Moment features are e.g. center of mass, orientation, bounding rectangle, best-fit ellipse and eccentricity. Object boundary and skeleton are also interesting features. Shape can also be described using Fourier and wavelet coefficients.

Another important category are the spatial relations within the image. However, they are not often used in prototype image retrieval systems. Example spatial relations are overlap, touch and disjoint.

This list of features is by no means complete, but it gives a good indication of the various features studied in the field. New features are likely to be found to solve specific problems. This requires that the database management system needs an extension mechanism for both data structures and operations on (complex) features.

Although many features exist, a limited number of data structures would suffice to store them. Many features are single values, i.e. no need for extra data structures. Some examples are area, pixel sum, and mean orientation. These can be stored using the database management systems built-in types, such as integer and floating point number.

Multi value features, like histograms of pixel values, vectors of eigenvalues, moment description vectors and segment descriptions, such as polygons, need additional data structures. However, a vector of complex data values would suffice to represent many of them. For polygons and histograms special data structures are needed. In the area of geographic information systems proper representations and index structures for geographic data like polygons exist[13].

The local features are calculated over parts of an image. For instance a single pixel or a segment can be the basis for this calculation. Since an image may have several pixels or segments, these features usually result in feature value sets for the whole image. This complicates calculation but the additional information may also produce better retrieval results[100].

Invariant features

One aspect of features has received great interest from the image processing community, namely their in variances to certain aspects, such as scale, rotation, view point and light source. To illustrate, assume we are searching for a certain scene in our image database. We do not care if the scene is recorded under a white or under a colored light source. In that case we should use features invariant under light sources. This means we have to look at the hue color component only or extract the color shift. But when
interested in sunsets or images taken at indoor dance parties, we definitely want variant features. In case of the sunsets we would like natural light sources, in the later case we look for artificial light sources. A similar story holds for the other variances, for example scale invariance could be useful unless your searching for objects you know the size of.

The negative aspect of invariance in feature space is that it reduces the feature selectivity. Invariance to some aspects makes a feature less specific and, therefore, less selective. Therefore, from a retrieval point of view invariant features are certainly not more important then variant features. Using the appropriate one at the correct time is far more crucial. This means that invariance is a predicate to be expressed at the query time only.

Retrieval requires that the permissible variances can be modeled explicitly. Modeling (in)variances requires the knowledge about the aspects that a feature is variant to. So if a feature is variant to a light source we should record that.

Requirements Summary

The requirements involving image features are summarized here.

1. The IDBMS should support for feature data types
2. The IDBMS should have support for modeling feature (in)variances.
3. Invariance can be expressed in the Query Language.

2.1.4 Image Semantics

Image recognition is research concerned with object recognition, i.e. tries to recognize the objects in an image and attache a description to these objects. Unfortunately, the image recognition problem has not been (and cannot be) solved in general.

However, in certain sub-domains interesting results have been obtained. One striking example is face recognition[86]. When it is possible to recognize the objects, we can construct a semantic description. Such semantic descriptions should be stored in an image database as well, which introduce new interesting problems. We will mention two: the multiple interpretation problem and the accumulated error problem.

The former stems from the fact that an image has many interpretations. Everybody can have its own interpretation of an image depending on the knowledge and cultural setting of the person. This results in many possibly large semantical descriptions. The consequence is that at query time, such an image database should be able to reason with multiple interpretations.

The second problem, the accumulated errors, results from the fact that images are always derived from inaccurate devices. When recoding an image
using some sensor error signals are added to the original scene. Also because of the digitization errors are introduced. Building semantic descriptions for these images will yield an accumulated error. The database management system should therefore be able to handle errors and error propagation.

Requirements Summary

The image recognition problem is still an open research area. Therefore, the problems related to representing the semantic information will not be considered in this thesis. In the future when semantics are attached to images these problems will come back and will than result in requirements for the image database system. Mostly at level of data modeling and semantic driven querying.

2.1.5 Image Queries

In database systems all relational operations are based on logical predicates being either true or false. This rigid logic perspective works, because the semantics of the data entities in business database systems are known and fixed.

For image database systems this is no longer sufficient. The world of images is a lot more fuzzy. Images can be very similar, but are hardly ever exactly the same. This makes it hard to write boolean predicates, such as image equality. A fuzzy set approach is in order here as an alternative.

Image queries are often navigational and steered by a user. Take for example image retrieval systems, which let a user navigate through the image space. In such systems the user constantly refines his query to navigate to the desired image. The reason for this navigational query approach are two fold. First, the mentioned fuzzy data makes predicates hard to use. The second reason is there is still little knowledge of the applications that uses an image database. This makes it hard to predict what sort of queries are needed, because it is unknown what the interesting data is.

Since the predicate logic expressions are hard to use, they should be replaced by a new technique for comparison. A solution found in many image retrieval systems is based on similarity measures.

**Definition.** A Similarity Measure, $S(a,b) \rightarrow \mathcal{R}_{[0,1]}$, expresses how similar two objects are.

Many similarity measures have been defined. For vectors different similarity measures exist as for value sets. One such fixed form similarity measure is based on the Minkowski metric.

$$S(a, b) = \frac{\sum_{i=0}^{n} ||a_i - b_i||}{\sum_{i=0}^{n} \max(a_i, b_i)} \quad (2.1)$$
This measure assumes the features in the vector are all unrelated, which is usually not the case. For example is the color of an object preserved by a human dependent on the colors in the area around the object.

The best known similarity measure is histogram intersection, which is formally defined as:

$$S(a, b) = \frac{\sum_{i=0}^{n} \min(a_i, b_i)}{\sum_{i=0}^{n} a_i}$$

(2.2)

This is a non symmetrical definition, i.e. the similarity for a,b and b,a are not equal. Therefore, the following symmetrical definition is also used frequently.

$$S(a, b) = \frac{\sum_{i=0}^{n} \min(a_i, b_i)}{\sum_{i=0}^{n} \max(a_i, b_i)}$$

(2.3)

These measures are both used for color histogram similarity calculations. The reason for its popularity is its robustness against cluttered images, and its invariance to scale and rotation. Also the measure is less variant to different view points. The measure also assumes the features in the vector are unrelated, for color histograms this is clearly not the case.

Therefore, another well known measure, i.e. the weighted Euclidean similarity measure is used. This measure comes from the family of squared similarity measures, which is formally defined by:

$$S(a, b) = 1 - \frac{1}{\sqrt{\sum_{i=0}^{n} (a_i - b_i)^2}}$$

(2.4)

And the weighted Euclidean similarity measure is defined by:

$$S(a, b) = 1 - aWb^T$$

(2.5)

Where $W$ is an $(n \times n)$ matrix which represents the weighting factor for each $i, j$ pair. Using this weighting factors the relations among feature values in the vector can be modeled. This $W$ should be derived from global database properties.

Measuring the similarity between feature sets is a less touched research direction. One known measure is the set intersection measure, as defined by:

$$S(a, b) = \frac{\|a \cap b\|}{\|a \cup b\|}$$

(2.6)

The problem of feature set comparison becomes even more difficult when the values in the sets are fuzzy. In that case the set elements need to be compared also using some similarity metric. Then the semantics of the $\cap$ and $\cup$ operators have to be changed too.

A database query will usually involve many different features, which should be compared using different similarity measures. Queries over multiple similarity spaces could be handled in various ways. A simple solution
could be to used traditional boolean predicates. For example selecting image on color and texture requires the color feature similarity value should excide a threshold $t_c$ and the texture feature similarity value should excide a threshold $t_t$.

A more advanced method could be a combination of the feature values involved\[58]. This method has the drawback that the search space explodes, since it combines two feature spaces in one, which makes query optimization difficult.

Another method is based on fuzzy logic operators. Fuzzy logic theory maps the and and or logical operators to minimum and maximum operations. For example the query showing in Figure 2.1.5 which selects images similar to an example image $ex$, will calculate the minimum of the color and texture features and the maximum of this with the combined color and texture features.

```
select
from images i
where i.color = ex.color
and i.texture = ex.texture
or i.color_texture = ex.color_texture
```

Figure 2.2: Example Query

The introduction of this new query model with similarity measures and with fuzzy logic operators requires new index structures. Such structures are more generally applicable when they are independent of the logic operator or similarity measure used.

**Requirements Summary**

The main requirement coming from image queries is a new query model. The current binary logic model is not suited for image queries. One suggested query model is the fuzzy logic model. Currently, similarity measures are used mainly.

### 2.2 DBMS

In this section we introduce our extensible main memory database management system, which is the appropriate system for image based applications.

#### 2.2.1 Extensible

Important requirements we found for an image database management system are
1. There is a strong need for Image and feature data types, and their operations.

2. There is a need for Index structures for efficient queries on images and features.

Therefore, we need an extensible DBMS.

An extensible database management system can be extended with new abstract data types, new commands and new index structures. This makes it possible to add an image data type to the system. An image would be treated the same way as ordinary types, like integers and strings. So the basic algebraic functionality, like the set operations: union, intersection, minus, and symmetrical difference and the relational operators: select, join and anti-join, would work without extra coding.

In addition new image processing commands can be added to such a system. Also new feature data types can be added. The new data structures can be large and expensive to query, therefore new index structures may be added to speed up the retrieval of these structures.

### 2.2.2 Main Memory

A design issue so far ignored is performance. Image applications are CPU demanding and often time critical. Example applications, like surveillance, authentication and error detection all demand high performance. But also interactive access of an IDBMS calls for a performance wise approach to avoid losing interest of end users.

Therefore, a database management system for image applications needs to deliver high performance. At the hardware level this can be achieved with better CPUs, memory and, disks. Such a system is no longer io-bound but CPU-bound, giving the system the appreciated performance. The ideal system for this is a main memory DBMS.

To further improve the performance of a database system shared memory multi-processor systems are needed. On these systems parallelism could be exploited to improve the throughput.

### 2.2.3 Objects versus Sets

As image processing software moves more and more towards object oriented program languages [68, 112, 105] it seems wise to choose an OODBMS for image database applications. The same programming language can then be used for both application and database specific code. The problem with this seemingly ideal case is the mismatch between the imperative programming paradigm and the declarative paradigm of the database. The advantage of only declaring what is needed will be lost in such a situation, because the
imperative programming paradigm of the object oriented requires you to specify how to obtain it.

A possible solution is to use a proper object oriented query language, like OQL, to interact with the database. Although such a combination of an object oriented language, such as C++, and an object oriented query language solves the mismatch, it requires proper programming practice from the application programmers to make full use of the database functionality. It's too easy to fall back to object at a time processing.

A better approach is to identify a minimal, but complete set of primitives for image applications, including both image operations and image query primitives. These combined with an extendable SQL like query language, such as SQL-99, would leave space for optimization and put no extra burden on the programmers programming skill. It clearly separates database use from application programming tasks.

2.3 Architecture of Monet

Monet is a novel database kernel under development at the CWI and UvA since 1994. Its development is based on experience gained in building PRISMA, a full-fledged parallel main-memory RDBMS running on a 100-node multi-processor, and on market trends in database server technology.

Developments in personal workstation hardware are at a high and continuing pace. Main memories of $>>1$ GB are now affordable and mass-market CPUs currently can perform over 1000 MIPS. They rely more and more on efficient use of registers and cache, to tackle the ever-increasing disparity\(^1\) between processor power and main memory bus speed. These hardware trends pose new rules to computer software – and to database systems – as to what algorithms are efficient. Another trend has been the evolution of operating system functionality towards micro-kernels, i.e. those that make part of the Operating System functionality accessible to customized applications. Prominent research prototypes are Mach, Chorus and Amoeba, but also commercial systems like Silicon Graphics' Irix and Sun's Solaris increasingly provide hooks for better memory and process management.

Given this background, we applied the following ideas in the design of Monet:

- *binary relation model.* Monet vertically partitions all multi-attribute relationships into Binary Association Tables (BATs), consisting of \([\text{DID, attribute}]\) pairs.

This Decomposed Storage Model (DSM) [27] facilitates table evolution, since the attributes of a relation are not stored in one fixed-width relation. Figure 1 shows this model in detail.

\(^{1}\text{In recent years this disparity has been growing with 40\% each year}\)
The price paid for the DSM is small: the slightly bigger storage requirements are compensated by Monet’s flexible memory management using heaps. The extra cost for re-assembling multi-attribute tuples before they are returned to an application, is negligible in a main-memory setting, and is clearly outweighed by saving on I/O for queries that do not use all the relations attributes.

Finally, maintaining all attributes in different tables enables Monet to cluster each attribute differently, and to precisely advice the operating system on resource management issues, for each attribute according to its access path characteristics.

- **perform all operations in main memory.** Monet makes aggressive use of main memory by assuming that the database hot-set fits into main memory. All its primitive database operations work on this assumption, no hybrid algorithms are used. For large databases, Monet relies on virtual memory by mapping large files into it. In this way, Monet
avoids introducing code to 'improve' or 'replace' the operating system facilities for memory/buffer management. Instead, it gives advice to the lower level OS-primitives on the intended behavior\(^2\) and lets the MMU do the job in hardware.

Unlike other recent systems that use virtual memory, Monet stores its tables in the same form on disk as in memory (no pointer swizzling), making the memory-mapping technique completely transparent to its main-memory algorithms.

- **extensible algebra.** As has been shown in the Gral system [51], many-sorted algebras have many advantages in database extensibility. Their open nature allows for easy addition of new atomic types, functions on (sets of) those types. Also, an SQL query calculus-to-algebra transformation provides a systematic framework where query optimization and parallelization of even user-extended primitives becomes manageable. Monet’s Interface Language (MIL) interpreted language with a C-like syntax, where sets are manipulated using a \textit{BAT-algebra}.

- **coarse grained shared-memory parallelism.** Parallelism is incorporated using parallel blocks and parallel cursors (called "iterators") in the MIL. Unlike mainstream parallel database servers, e.g. PRISMA [2] and Volcano [50], Monet does not use tuple- or segment-pipelining. Instead, the algebraic operators are the units for parallel execution, which simplifies query optimization. Their result is completely materialized before being used in the next phase of the query plan. This approach benefits throughput at a slight expense of response time and memory resources.

### 2.3.1 Monet Architecture

The architecture of Monet is structured as a frontend/backend system. The current implementation has frontends for the Monet interface language, the object database management groups[17] object definition language (ODL) and for the structured query language (SQL) (see Figure 2.3.1).

The Monet database system is designed to be a extensible in all directions. Meaning, new data types, commands and accelerators can be added. The MIL has a sister language called the Monet Extension Language (MEL), which should be used to specify extension modules. These modules can contain specifications for new atomic types, new instance- or set-primitives and new search accelerators. Implementations have to be supplied in C/C++ compliant object code.

\(^2\)This functionality is achieved with the \texttt{mmap()}, \texttt{madvise()}, and \texttt{mlock()} Unix system calls.
2.3.2 Monet Interface Language

The Monet Interface Language is a low level BAT-manipulation database query language, extensively described in [9]. For self containment of this thesis, we introduce some part of Monet's instructions set and programming concepts.

Table 2.2 lists the relational operations with there functionality. Monet uses binary tables (BAT) all relational operations are defined on those. The first column of a BAT is called its head column and the second its tail column.

The select operation selects all binary units (BUNs) where the tail value of the input BAT is between the lower and upper bounds given. The semi-join operation selects all BUNs of the $\alpha$ BAT whose head value also occurs in the $\beta$ BAT. The join operation implements a natural equi-join based on the tail of the $\alpha$ BAT and the head of the $\beta$ BAT.

The difference operation (diff) selects all BUNs which occur in $\alpha$ but not in $\beta$ BAT. The union and intersect are the well known set operations. The unique operation selects all unique BUNs from a BAT. All these functions work over both the value in the head and tail. i.e. an intersection between two BATs is the intersection between the pairs of head and tail values. They
all have an equivalent version based only on the head value, these functions have the same name prefixed with a k, i.e. kdiff, kunion and kintersect.

<table>
<thead>
<tr>
<th>RELATIONAL OPERATION</th>
<th>FUNCTIONALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>select(BAT α, T low, T high)</td>
<td>{ab : ab ∈ α ∧ low ≤ b ≤ high}</td>
</tr>
<tr>
<td>semijoin(BAT α, BAT β)</td>
<td>{ab : ab ∈ α ∧ cd ∈ β ∧ a = c}</td>
</tr>
<tr>
<td>join(BAT α, BAT β)</td>
<td>{ad : ab ∈ α ∧ cd ∈ β ∧ b = c}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SET OPERATION</th>
<th>FUNCTIONALITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>diff(BAT α, BAT β)</td>
<td>α/β</td>
</tr>
<tr>
<td>union(BAT α, BAT β)</td>
<td>α ∪ β</td>
</tr>
<tr>
<td>intersect(BAT alpha, BAT β)</td>
<td>α ∩ β</td>
</tr>
<tr>
<td>unique(BAT α)</td>
<td>{ab : ∃ab ∈ α}</td>
</tr>
</tbody>
</table>

Table 2.2: Monet's Relational and Set Operations

Table 2.3 lists the construction and update operations with their functionality. Bats are created using the new(ht, tt, capacity) constructor, where ht and tt represent the head and tail column type, respectively. The capacity is an optional parameter to identify the initial table size. A proper guess of the tables size reduces memory fragmentation.

Bats can be updated using the insert(BAT, h, t), replace(BAT, h, t), and delete(BAT, h, t) primitives. Insert appends a single pair of atom values to the end of the table. Replace updates all tail values for binary records (BUN) with the given head value. Delete removes all matching pairs. Monet assures BUN's are in consecutive memory, i.e. no holes are allowed between BUN's. Therefore, the BAT scan operations can be kept simple and fully optimized.

Performing a single operation on all tail values of a table can be done using the [] multiplex operator. For example adding the tails of two tables can be done using the statement [+](a, b), this will find pairs with equal heads and add the tail values together. Aggregating groups can be done using the {} operator. This operator performs an operation for each group in the given BAT. Groups are identified by unique values in the head column of the given table. For example to sum all groups we can use {sum}(a), where sum is a BAT operation to add all tail values in the BAT. The sum operation gets one parameter, a BAT. In the same way an average, min, and max operators over groups can be defined. Such operator gets the group BAT so it can do any initialization steps itself, i.e. for the sum start at zero, and also any post processing, i.e. for the average dividing the sum by the group count.

MIL is a procedural language. It has well-known flow of control constructs such as, while and if/then/else statements. Also BAT related iterators such as batloop, and hashloop(a), iterates over all BUN's executing a MIL block. New MIL procedures can be introduced using the proc
### CHAPTER 2. IMAGE DATABASES

1 proc sum (BAT[any,int] b) : int := {  
  var res := 0;  
  b@batloop(){  
    res += $t;  
  }  
  return res;  
}

**Figure 2.5:** Example mil procedure

The keyword. See Figure 2.5 for a simple MIL procedure which sums a BAT. The first line defines the procedure. Sum gets a BAT of integers as parameter, and returns a single integer. In line 2 the res variable is initialized. Line 3 to 5 loop over the BAT and add the BUN’s tail value ($t$) to the res variable. Finally in line 6 the sum is returned to the caller.

<table>
<thead>
<tr>
<th>Operation</th>
<th>functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>new(ht, tt, capacity)</td>
<td>BAT[ht,tt]</td>
</tr>
<tr>
<td>insert(BAT α, T h, T t)</td>
<td>α ∪ ht</td>
</tr>
<tr>
<td>delete(BAT α, T h, T t)</td>
<td>{ab : ab ∈ α ∧ a ≠ h ∧ b ≠ t}</td>
</tr>
<tr>
<td>replace(BAT α, T h, T t)</td>
<td>{at : (at ∈ α ∧ a ≠ h) ∨ (ab ∈ α ∧ a = h)}</td>
</tr>
<tr>
<td>sum(BAT α)</td>
<td>a₀ + ... + aₙ ∧ a₀, ..., aₙ ∈ α</td>
</tr>
<tr>
<td>avg(BAT α)</td>
<td>(a₀ + ... + aₙ)/n ∧ a₀, ..., aₙ ∈ α</td>
</tr>
<tr>
<td>max(BAT α)</td>
<td>max(a₀, ..., aₙ) ∧ a₀, ..., aₙ ∈ α</td>
</tr>
<tr>
<td>min(BAT α)</td>
<td>min(a₀, ..., aₙ) ∧ a₀, ..., aₙ ∈ α</td>
</tr>
<tr>
<td>[op](BAT[h,t1] α, BAT[h,t2] β)</td>
<td>{at : ab ∈ α ∧ ac ∈ β ∧ t = op(b,c)}</td>
</tr>
<tr>
<td>{op}(BAT[h,tt] α)</td>
<td>{ab : β = {c : ac ∈ α} ∧ b = op(β)}</td>
</tr>
</tbody>
</table>

**Table 2.3:** Monet’s BAT Update Operations

#### 2.3.3 Monet Extension Language

Monet is an extensible database system, and hence MIL is an extensible language. Experts in some application domain can extend the Monet database system to store new kinds of data and define operations on them.

Extending Monet is achieved by writing an extension module in the Monet Extension Language (MEL). The MEL is a specification-only language, see Figures 2.6, 2.7, 2.8 and 2.9 for an example MEL specifications.

A MEL module describes the types (atoms), index structures (accelera-
2.3. ARCHITECTURE OF MONET

tors), and commands to include in the Monet kernel. The module definition also contains help texts on the new primitives, which will be inserted in Monet's online help system. Finally, the module contains references to C functions that implement the primitives.

Mel modules can contain any of the following Monet extensions:

**new atomic data types** Mil has built-in support for `{boolean, character, integer, oid, pointer, float, double, long, and string}` values. You can add new types, like `date` or `vector` readily (see Figure 2.6).

**new algebraic commands or operators** Algebraic commands get passed a set of values as parameters, then do some execution, and return one value. Mil allows for overloading of algebraic commands and operators. The extension programmer should specify a type-signature and MIL chooses dynamically which command to use on the basis of the actual parameters (see Figure 2.8).

**new search accelerators** Some operations on tables need additional – persistent – data structures for efficient execution. Famous examples are R-trees and hash-tables. The accelerator builder should provide the interface operations to create, destroy, and traverse such tables. The database needs to keep these tables consistent under updates; for this reason additional update interface operations, `insert` and `delete`, are required from the extension programmer (see Figure 2.9).

MEL Modules

Modules are the unit of extension in Monet: a module is loaded, or not. When it is loaded, all its new language elements are added to the Monet's interpreter language. When a module is `dropped`, they are removed.

The `.prelude` and `.epilogue` constructs allow for C routines to be called when a module is loaded and dropped **physically**. An example for its use is the initialization of (empty) C structures, or for instance the creation (/destruction) of shared locks. The C routines are parameterless and do not return any value.

The `.load` and `.drop` keywords allow for the specification of MIL code that is to be executed when a module is loaded. These features come especially handy for defining standard MIL procedures (procs) and constants. As opposed to the prelude/epilogue initializations, the load/drop scripts are executed at logical load and drop points: each time a user loads or drops a modules (they are executed in the context of that user). The prelude and epilogue are only execute once at physical module load and unload.
atomic types

An important design issue of Monet’s atomic types is that there are basically two classes:

fixed-size atoms Their memory management is simple, because all possible instances have the same size. They are stored directly in the BUN heap (tuple heap) of a BAT. These types are very efficiently implemented in Monet.

variable-size atoms From the builtin-types, string is the only variable sized type. Values are stored in a separate heap, the BUN heap contains integer byte-offsets into this heap. Monet ensures in this way that the BUN heap can be implemented as an array of fixed-size elements, even if it contains values of variable-size.

The correct functioning of all Monet’s standard operations (like join and select) is guaranteed by suppling the atom interface for any new ADT, see Figure 2.4 and 2.5.

For instance, a hash-function on a string is required to make Monet’s hash-join work on columns of strings. The heap provides a context for domain dependent optimization. For example, the implementation of the string atom uses a hash-based catalogue in this heap. The catalogue is used to have only a single copy of a string in the heap. MIL will recognize each new atom as a keyword.
2.3. ARCHITECTURE OF MONET

<table>
<thead>
<tr>
<th>Function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FromString</td>
<td>constructs the atom from a string</td>
</tr>
<tr>
<td>ToString</td>
<td>converts the atom to a string</td>
</tr>
<tr>
<td>hash</td>
<td>calculates a hash number for the atom</td>
</tr>
<tr>
<td>nequal</td>
<td>tests if two atoms are not equal</td>
</tr>
<tr>
<td>comp</td>
<td>compares two atoms, returns smaller, equal or larger</td>
</tr>
<tr>
<td>null</td>
<td>get the nil value for this atom type</td>
</tr>
</tbody>
</table>

Table 2.4: Fixed Atom Interface

<table>
<thead>
<tr>
<th>Function</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heap</td>
<td>creates and initializes the heap</td>
</tr>
<tr>
<td>put</td>
<td>insert an atom in the heap</td>
</tr>
<tr>
<td>get</td>
<td>get a copy of an atom from the heap</td>
</tr>
<tr>
<td>del</td>
<td>delete an atom from the heap</td>
</tr>
<tr>
<td>len</td>
<td>get the size of the atom</td>
</tr>
</tbody>
</table>

Table 2.5: Additional functions for Variable Sized Atom Interface

**Redefined Atoms** A feature borrowed from Object Orientation is atom overloading. One atom can be implemented using the existing interface functions implemented by a parent atom. For instance an rgb atom is implemented as a different interpretation of a vector3D atom, See Figure 2.7. The new type is different from its implementation type on the logical level in MIL, but at the physical level it uses the implementations of its parent type (or actually its root ancestor).

```
.module rgb;
  .atom rgb = vector3D;
  .end;

  .operator (rgb) "+" (rgb) : rgb = rgb_add; "rgb addition"
  .operator (rgb) "+" (rgb) : rgb = rgb_min; "rgb substraction"
  .operator (rgb) "*" (rgb) : rgb = rgb_mul; "rgb multiplication"
.end rgb;
```

Figure 2.7: example of the overloading of Monet ATOMs

2.3.4 new primitives

Apart from atomic types, the MEL extension mechanism also allows for introduction of new execution primitives into MIL.
Commands and operators are much alike. The exact way in which a command implementation in C will get passed its parameters and what it is expected to return will not be discussed here.

The functional part of the MIL language is a set of algebraic commands and binary and unary operators. Commands get passed a set of values as parameters, then do some execution, and return a (single) value. A collection of such commands and operators, where for each $f(D) \subseteq D$ holds, is called an algebra. ³

Figure 2.8 shows an example image module containing some new mil commands and operator.

```
.module image;
  .atom image = BAT;
  .tostr = imagetostr;
  .fromstr = imagefromstr;
.end;

.command width( image ) : int = imagewidth;
  "Get the width of this image"

.command height( image ) : int = imageheight;
  "Get the height of this image"

.operator (image) "=" (image) : bit = imageeq;
  "Test if the images are equal"

.operator (image) "!=" (image) : bit = imageneq; ""
  "Test if the images are not equal"

.command readFile( str filename ) : image = readFile;
  "read an image from the file, the extension expresses the format in which the image is stored"

.command writeFile( str filename ) : image = writeFile;
  "write an image to the file, the extension expresses the format in which the image should be stored"

.end image;
```

Figure 2.8: The image module

³It is actually not correct to call this a true algebra, since BAT-parameters are call-by-reference (as opposed to the simple values which are call-by-value), and can hence be modified. This is just a pragmatical choice.
2.4. STATE OF THE ART OF IMAGE DATABASE SYSTEMS

MEL also allows for overloading of the multiplex, [], and group aggregate operations, {}. At run time when MIL resolves the formal call [+{} to a physical function it will take the special optimized case for this operator. The default implementation of the [+{} operator uses a batloop and calls a + function for each BUN.

2.3.5 New Search Accelerators

A search accelerator is defined as 'a data structure associated with a database column kept up to date with changes'. The prime reason for maintaining such data structures is to achieve better speed on common database operations.

Well-known search accelerators for traditional data types are the B-tree and hash-tables. Examples from the GIS application domain are Grid-files and R-trees[52, 4, 83].

For this reason the Monet’s extension mechanism allows for addition of new (persistent) search accelerators. A BAT can hold two user-defined search accelerators: one for the head and one for the tail. Monet provides two standard search accelerators oriented towards main-memory relational processing: the index binary tree index and the hash chained bucket hash-tables.

The metric accelerator which will be introduce in Chapter 8 is shown in Figure 2.9. This example accelerator requires besides the construction (BUILD) and destruction (DESTROY) only insert and delete instructions to keep the accelerator inline with the underlying BAT. It includes the definitions of the vector module using the USE statement.

Monet will recognize each new accelerator as a MIL keyword.

2.4 State of the Art of Image Database Systems

In this section we give a short overview of image database management systems, image retrieval systems and Image indexing techniques described in literature. Each image database management and image retrieval system is first described individually. These sections conclude with a evaluation based on the requirement list. This means we evaluate each IDBMS on extensibility with new data types, commands and index structures. For image retrieval systems we evaluate based on the existance of image operations, the available image file formats, and on the available global and local features.

2.4.1 Commercial Image Databases

Despite the awareness in the database research community that general database technology would be a clear asset to multi-media application domains, limited progress has been made so far. This partly stems from a lack of
application domain knowledge within the database community to isolate the functionality needed, as well as the lack of experience in using database technology in the image research community to focus the effort on query formulation and evaluation instead of dedicated storage and index management.

Moreover, it has only recently become manageable to enhance the database kernels with application domain functionality. For example, research prototype database systems, such as postgres[103], Jasmine [26] and Starburrs showed the route towards low-level extensibility of a database kernel. This route is only recently followed by Oracle, DB2 and INFORMIX. It is expected that these facilities will become available in all commercial systems within a few years.

**Oracle Data Cartridges[84]**

The Oracle 8 universal server supports a form of abstract data type (ADT) extensibility. Oracle calls extension modules "data cartridges". A data cartridge defines new "Object data types" (ODT) with their behavior. The description specifies both the ODT attributes in terms of existing Oracle data types, member functions and procedures on these data types.

The procedures can be written both in PL/SQL, Oracle’s extended structured query language, and in C using external shared libraries. The PL/SQL is not well suited for object member function implementations, because this high level interpreted language caries to much weight to achieve the required
2.4. STATE OF THE ART OF IMAGE DATABASE SYSTEMS

high performance. The shared libraries runs in a separate process, which ensures Oracle’s server stability under bogus member function implementations. The downside of this separate process model is performance. A call to a separate process is orders of magnitude less efficient than an in process function call. Furthermore, the external libraries can only access the server via Oracles call interface (OCI), which again reduces performance. So both implementation paths are performance wise not very promising.

INFORMIX Data Blades[57]
The INFORMIX-Universal Server provides a very advanced extensibility interface, called Data Blades. A Data Blade module may include the following components; new data types, functions, access methods, tables and indexes, and client code.

User defined data types are treated as built in types. The database allows various new data type definitions, the opaque type definition offers maximum flexibility. It allows any data represented in C structures to be natively stored and processed by the server.

The function component is a collection of function definitions which operate on any data types, new or built in. These functions extend the processing and aggregation functionality of the database.

The access method component enables Data Blade developers to write special index structures. An index structure for the INFORMIX server is defined by a set of methods, open a scan, get next record, insert, delete, replace and close scan.

The interface component can be used to export functionality of a Data Blade. So, for instance an image retrieval Data Blade can use a text retrieval Data Blade for keyword search.

A Data Blade developer can store and index data needed for the Data Blade in tables and index structures directly. The client code component contains the code which exports a client user interface for the new data types.

INFORMIX supports an extensibility mechanism which is powerful enough to support an image data type and operations on it. There is no support for polymorphic data types.

DB/2 Universal Database[56]
The DB/2 Universal Database from IBM also supports extensibility, called extenders. These extenders can extent the DB/2 server with user defined types and user defined functions. These user defined types can only be stored in large objects. This way common relational operators on these newly created types are lost. The defined functions can be used from the DB/2 SQL interface.
The supported extensibility of DB/2 is limited. New data types are treated different than built-in types and it is not possible to extend the server with special index structures.

**Jasmine OO Database server**

Jasmine is a fully object oriented database management system. It supports addition of new classes, inheritance and method overloading. The index is hidden, i.e. no support for advanced index structures is available.

This database has been extended with multi-media classes. The main feature is data independence. Images can be stored in the database, so the physical location is no longer needed.

**Image Database Comparison**

The differences between the commercial systems and Monet's extensibility are summarized in table 2.6. A + indicates the extensibility is available ( - means not available). A ++ means available and has a superior performance.

<table>
<thead>
<tr>
<th>Database</th>
<th>ADT</th>
<th>Commands</th>
<th>accelerators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle 8</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Informix</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
<tr>
<td>DB/2</td>
<td>+</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Jasmine</td>
<td>++</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Monet</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 2.6: Image Database Comparison

As can be seen from this comparing table the Informix and Monet system clearly support the extensibility needed. The Monet system was chosen because of its superior performance which comes from its main memory oriented implementation.

**2.4.2 Commercial Image Retrieval Systems**

**Virage: Visual Information Retrieval Module**

The Virage Visual Information Retrieval (VIR) Module extends commercial database management systems, such as Oracle 8, INFORMIX Universal Server, Sybase and Object Store from Object Design, with image storage and management capabilities. In addition developers can use the VIR Image Engine to interact with their own DBMS. The VIR Image Engine capabilities include:

**storage** Reading and writing multiple image file formats.
thumbnail Automatic thumbnail creation.

content Analysis and comparison of images based on their visual content.

The storage capability provides users shared access to images centrally stored in a DBMS. The VIR Image Engine supports translations between popular image file formats during storage and retrieval, the following image file formats are supported: JPEG, BMP, SGI, PSD, Sintex CT, TIFF, PICT, TGA, MAC, RLE, EPS, PNG and PCX. The list of file formats again illustrates the lack of a standardized image data type.

The VIR Image Engine provides a simple interface for image thumbnail creation. A reference to the original full size image is maintained. The thumbnail creation is offered for performance reasons. The lack of a complete set of image operations makes this special feature necessary.

The visual comparison capability allows users to search for images based on their content. Virage uses four features, i.e. color, color composition, structure and texture to describe an image. These quantitative measures provide easy access based on a similarity metric.

The VIR Image Engine provides no query language enhancements, such as fuzzy or probabilistic reasoning. All queries should be defined as boolean predicates on the features extracted from the images. Querying for similar features involves full scans of the feature tables, because no special index structures is added.

Excalibur Visual Retrieval Ware

Excalibur has build retrieval software, which runs on Jasmine, INFORMIX and Oracle. It uses of the image contents shape, color and texture to index the database. Retrieval is supported by query by example or by sketch.

The Excalibur Visual Retrieval Ware is based on Excalibur’s Adaptive Pattern Recognition Processing (APRP) technology. APRP acts as a self-organizing system that automatically indexes binary patterns in the digital information, creating a pattern-based memory that is self-optimized for the native content of the data. The bases for this indexing are the shape, color and texture features extracted from the images.

Oracle Visual Image Retrieval Data Cartridge

Oracle also has its own image cartridge, the Oracle8 Visual Image Retrieval Cartridge. This cartridge supports image storage in various image formats. No support for image operations and image query extensions is provided.

2.4.3 Research Image Retrieval Systems

Several image database retrieval projects are underway, see survey [91]. A few snapshot descriptions are illustrative for the approaches taken.
Keyword based image retrieval is supported by the web search engines Yahoo and Alta Vista. They support search for images based on categories and keyword matching. Yahoo manually annotates the images. Alta Vista uses an annotated stock photo archive. No support for image retrieval on image content can be found here.

The QBIC project [41], which later became a commercial product, studies methods to query image databases based on the image content, it is based on IBM’s DB/2 Image extenders. The content features include color distribution, texture, and position and shape of edges. The color feature is described by the average RGB and Munsell[74] color coordinates and by a 64 bins color histogram. The texture is summarized by a triplet, i.e. coarseness, contrast and directionality. Shape is described as a combination of area, circularity, eccentricity, major axis orientation and a set of algebraic moments. The similarity measure used are limited too quadratic form distance functions, like the Euclidean distance.

To improve efficiency, the search space is reduced using a lower bound metric on the color histogram Euclidean distance. The average color turns out to be a lower bound for this distance [93]. Therefore, using the average color does not result in missing actual hits, though extra false hits will be introduced.

The VisualSEEK image retrieval system, as described in [101], automatically segments the image into objects with equal color-set content. A color set represents the colors in a segment. The spatial information about these objects is stored. Using both the spatial and color properties the user can query this database. A large database of 12,000 images is used in their web demo.

It has an interesting graphical interface, called SaFe where spatial relations between features can be modeled by sketch. The features used for this search type are color and spatial relations between similar regions.

Recently two other research projects appeared with a system using spatial relations, ExSight[120] and Blobworld[16]. Both systems start by automatically segmenting the images in the database. The user can then specify the queries by selecting segments from sample images and spatially arrange them to from a spatial image query. The system searches for all similar images based on these spatialrangements using the segments features. Query results show why an image is returned by displaying the segments used and optionally the features can also be visualized.

The Photobook [87] provides a large amount of image processing functionality useful for content-based image retrieval. An example is the semantics preserving image compression technique, which reduces images to a small set of perceptually-significant coefficients. Using a training set of images, the "eigenimage" vectors are computed. These vectors are used to compress the image content information. The similarity between two images is computed using the distance in this compressed "eigenimage" space. This
has been successful in face recognition.

The approach taken by the PictoSeek [46] is to build histograms of the hue, the dominant hue edges and hue corners. The hue color component is chosen since it is invariant to surface specularities, like shadow and highlights. The similarity measure is color histogram intersection [106], which is less variant to occlusion and less dependent on the view point. Histograms are invariant under a number of transformations. A web demo is available with various databases, the largest contains 10000 images.

To improve retrieval performance various experiments are done with signatures. A signature indates the presence or absence of a color in the image. Using binary operators such as and, or, and x-or quickly a set of images with similar colors can be retrieved.

Image Retrieval System Comparison

The difference between the various (non-)commercial image retrieval systems and Monet’s image retrieval system are given in tables, 2.7 and 2.8. Table 2.7 expresses how the image retrieval systems score on the availability of image operations and image input/output routines to various standard image file formats. Scoring is again done with ++ (very good), + (available) and - (not available). The non-commercial image retrieval systems have limited documentation about their image ADT.

<table>
<thead>
<tr>
<th>Retrieval</th>
<th>Operations</th>
<th>Input/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virage</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>Excalibur</td>
<td>-</td>
<td>+</td>
</tr>
<tr>
<td>Jasmine</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Oracle</td>
<td>-</td>
<td>++</td>
</tr>
<tr>
<td>Monet</td>
<td>++</td>
<td>+</td>
</tr>
</tbody>
</table>

Table 2.7: Image Retrieval Systems Comparison of the image ADT

The second table shows the level on which the retrieval takes place, globaly or localy (segments or pixels).

As can be seen from this comparing these tables none of the commercial retrieval systems include all required functionality. The VisualSEEk, Blob-world and ExSight score in the same range as Monet. We will explain more about the Monet’s image retrieval system in the chapters 4.2 and 4.

2.4.4 Image indexing techniques

Data structures for image feature indexing have received quite some research attention. The baseline is to replace the search key of an ordinary index structure by a feature vector and to include a proper comparison operator.
CHAPTER 2. IMAGE DATABASES

<table>
<thead>
<tr>
<th>Retrieval</th>
<th>global features</th>
<th>local features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virage</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Excalibur</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>Jasmine</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Oracle</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>QBIC</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>VisualSEEk</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>PhotoBook</td>
<td>++</td>
<td>-</td>
</tr>
<tr>
<td>PictoSeek</td>
<td>+</td>
<td>-</td>
</tr>
<tr>
<td>Blobworld</td>
<td>+</td>
<td>++</td>
</tr>
<tr>
<td>ExSight</td>
<td>++</td>
<td>+</td>
</tr>
<tr>
<td>Monet</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

Table 2.8: Image Retrieval Systems Comparison of feature levels

For example, quad-trees can be easily extended to encode multilevel color histograms, by Lu et.al. [53]. This enables fast similarity searches based on those color histograms.

Signature files, originally developed for textual information retrieval, have been extended by Faloutsos [3]. The trick is to use the important image features as signatures for the images. Fast retrieval can be achieved using bit comparison on the signature files.

Chang et.al.[98] proposed a “2D-string representation” to encode the objects and their spatial relationships. Similarity retrieval of images encoded in 2D strings is mapped to substring matching.

Nabil et.al. [75] use a graph-based encoding of the objects and their spatial relationships. Subsequently, retrieval is turned into a weighted graph-matching problem.

The Fourier transform of a signal yields a frequency decomposition which is rather unsuited to describe local transitions. The wavelet transform [31, 104, 114] is designed to describe signals at different scales. The wavelet coefficients yield a multiresolution decomposition of a signal.

Jacobs et al.[59] apply a fast Haar wavelet transform to each color band of the images. The feature vector is composed of the N maximal coefficients of the wavelet transform, only the sign and indices are use not the values. Also the average pixel values of each color channel are used. Using this feature vector they claim to be able to find an image based on an in accurate (low-resolution) version of the image.

Wang et.al.[115, 116] uses the Daubechies 4-layer 2-D fast wavelet transform. They use a hierarchical query method. First based on the standard deviation in the 8*8 low frequency bands of the wavelet transform is used to fast reduce the result set. This set is further reduced using the weighted
distance between the 8*8 low frequency bands. The last step uses the 16*16 low frequency bands to find the best matching images.

In [99] a method for segmentation based on the quad-tree index structure is introduced for texture based image queries. Each image is recursively split in four parts until the distances in the texture-feature space between the parts and its enclosing part exceed a certain threshold. Image parts are merged when their texture feature distance is less than this threshold. For each resulting segment the texture is calculated. So each image is represented by a set of segment, texture pairs. A user can query this by supplying an example texture. The texture features are based on the Quadrature Mirror Filter wavelet representation.

In [120, 101, 16] methods based on segmented images are described including the segmentation algorithm. The Blobworld segmentation is done based on color features using the HSV color model and texture features. Safe only uses color to segment and query the images. The segments in Blobworld are described using its centroid and scatter matrix expressing the variance, eccentricity and orientation. The queries in Blobworld use fuzzy operators to combine feature values. Use of the and operator in a query will take the minimum of the two feature values incase of the or operator the maximum is used.

2.4.5 Requirements

From the existing systems we can deduce a list of requirements. Aside from the basic requirements, i.e. storing and retrieving of images and derived features, these systems need to compare features using similarity measures, many of which exist. Unfortunately none perform perfectly in all cases. So we need to support a large set of these feature comparing measures. Because we use an extensible system later found measures can be easily added to the system.

Most current systems do not allow for partial image queries, i.e. give me all images which contain the following parts. Only the SaFe, ExSight and Blobworld systems, lets users specify a query using combinations of spatial relations and color and texture features.