A Database Management System For Vision Applications

John Oakley, Richard Shann, Darryl Davis, and Laurent Hugueville.

Multi–Media Research Group, Electrical Engineering,
Manchester University, Oxford Road, Manchester, M13 9PL, U.K.

Abstract

The Manchester Content Addressable Image Database is a generic tool which has been designed for image informatics and computer vision problems. The system stores pre–computed feature tokens, which are obtained by conventional processing of the input images, into a database which is accessed by a specialised query language (MVQL [1]). The MVQL is based on the creation and refinement of groups of features by computing attributes.

We illustrate the system with an application concerned with the detection and classification of microfossil images. We use the MVQL to express a projective circular Hough Transform for microfossil detection. We also address the problem of classifying detected structures into six broad morphological groups. This is achieved using MVQL to define "structure" measures from the distribution of curve tokens in a circular region around each microfossil.

1. Introduction.

The aim of this paper is to present a simple database management system (dbms), which includes a data entry system, query interface and some graphical capability, which has been specifically designed for vision research. The work is particularly motivated by the need for content addressing of large volumes of image data. Over the last decade vision research has moved away from analysis of single images in constrained environments towards processing of bulk data in less constrained environments, for example image sequences from surveillance cameras. However the most common infrastructure is still a C (or C++) compiler together with (sometimes) a library of vision algorithms. A good example of the usefulness of indexing tools in machine vision are the various tree structures [3,4]. While 2–D arrays for data (reflecting 2–D spatial organisation) are common the more general techniques are less widespread because of the need to re–code for each application. The problem of storage and retrieval of image data, both image intensities and structures derived from images, is now of central importance [5,6,7].

BMVC 1994 doi:10.5244/C.8.62

Forty years of research into databases and associated algorithms has yielded many powerful tools. Of particular interest to vision workers are

- indexing, particularly spatial indexing
- storage management
- user interfaces
- query languages
- graphical databases.

In our view there are three main reasons for the poor take–up of database technology in vision systems:

1. Restricted Accessibility

Commercial database packages require a significant investment in terms of hardware/software cost and training requirements. The user must be something of an expert to arrange the data, the relations and the indexing in a way that would give rapid answers to the intended queries.

2. Flexibility

Using a commercial database package is often seen as using a hammer to crack a nut. The benefits are only seen in near–market research.

3. Useability

Although the SQL (and its new cousin Object–SQL) provides a standardised user interface, it is overly general, with a large syntax not well–matched to vision tasks.

We have addressed these issues by designing a low cost dbms with a simple query language, the Manchester Visual Query Language (MVQL), and some graphics capabilities. The motivation for this development was a requirement for content addressing of image data. The idea is to use image processing to obtain sets of primitive features (for example edges, curves, textons) from a large set of images and to store these as a kind of summary of the image content. Queries are then directed at the feature data in order to retrieve images of interest. For this reason we refer to the dbms as a Content Addressable Image Database (CAID) although is could be used in applications other than image retrieval.

In section 2 we address three important aspects of the Manchester CAID: the query language, spatial indexing method, and the implementation. In section 3 we give an

example of how the CAID may be applied to the problem of analysis of geological samples through microfossil detection and classification. In this way we show that the dbms, and in particular the query language is sufficiently expressive for real vision applications.

2. The Manchester CAID

In content addressing problems the number of primitive features can be quite large and have spatial, temporal and other aspects which will need to exploited to achieve a reasonable access to the image content To form these groupings of primitives we have developed a visual query language (the MVQL) based around a small set of set creation and refinement operations. Starting from sets of primitives the main operations defined are two pair operations (includes one, kD–PAIR which can exploit range restrictions), a partitioning operation that divides a set into non–overlapping subsets (called bins); and a selection operation.

2.1 The Basic MVQL Operators

The MVQL PAIR operator creates a set of objects which have properties computed from their constituents (FIRST and SECOND). Likewise the partioning operator BIN forms groups with properties (eg MEAN, MAX, SUM, COUNT) computed over the elements in the bin. To give some examples, we suppose our database to contain as primitive features arcs detected in the images. Each arc is assigned position coordinates *(X,Y)* (on the arc itself, rather than at the centre of curvature), orientation *(Angle)* and radius of curvature *(Radius)*.

A simple MVQL query for this database can be formed from the **SELECT** operation. For example the query

*largeArcs := (Arc) SELECT PI*SQR(Radius)>50*

defines largeArcs as the set of Arcs forming part of circles with area greater than 50 square pixels.

To illustrate forming a set of pairs, consider the query

arcPairs := PAIR((Arc,largeArcs),{FIRST[Angle]=SECOND[Angle]}) SELECT distance<10

where we use the macro

distance=SQRT(SQR(FIRST[X]−SECOND[X])+SQR(FIRST[Y]−SECOND[Y]))

as an aid to legibility.

The expression in curly braces is the pair–partition, which in this case has the effect that the processing to take place will first form blocks of Arcs of the same orientation and compute the distance only between elements in the blocks. The resultant set is a set of pairs of closely spaced parallel arcs, and the number of such pairs is reported on the console. An alternative version of the PAIR operator, the metric pair, exists to exploit situations in which the data can be spatially indexed. This is described in section 4 of this paper.

To illustrate the BIN operator consider the task of grouping Arcs that belong to the same Image and fall in the same quadrant of that image. If our image was a 256x256 pixels image then the expression X DIV 128 take on the values 0 and 1 in the left and right halves of the image, and Y DIV 128 would take on the values 0 and 1 in the top and bottom halves. So a partition of the data by image and by these expressions would yield the grouping required. As an example, the set

arcClusters:= BIN((Arc),{image (X DIV 25) (Y DIV 25)}) SELECT COUNT>10

groups Arcs by image and by position in a 10 by 10 grid of squares superimposed on the image (again assuming a 256x256 image). The resultant set is a set of groups of arcs, each group being associated with one square in one image, and each group having at least ten members. Set reduction is accomplished by SELECT and RANKing operations. The query language allows the operations to be nested and in this way quite complex groupings can be specified fairly simply. The running time for a query can be calculated given the operations and the number of entities involved; it is linear in the number of partitions and, for pairing, quadratic in the number in the largest partition.

2.2. Dynamic Spatial Indexing

The MVQL pair operator specifies a Cartesian product of two sets and hence the number of objects formed increase as N^2 if N is the number of elements in each set. For example if we have a db of 1000 images with 4000 features in each image then the number of possible pairs of features is 1.6×10^{13}. However we are often interested only in pairs of features belonging to the same image. For example, if we were to form corner hypotheses by grouping line segments then it would make little sense to group a line from one image

with another line from a different image. In this case we only have $4x10^6$ useful pairs. The MVQL allows partitioning of data in arbitrary blocks of compatible features in order to avoid a quadratic increase in cost with increasing N. Further gains in efficiency are obtained by exploiting range constraints. For example, it may not be necessary to form pairs of features which are well separated within an image. This can be achieved by spatial indexing of the features within a coordinate space using, for example, a k–D tree [3,4]. We have implemented a dynamic version of this strategy, where the query defines the coordinate space required, and gives the range constraints to be applied to the pairs. If N denotes the number of features in the partition then (to take the case of pairing a set with itself) searching a kD–tree of N items for matches with another set of N items take O(NlogN) time. Since the creation of the k–D tree also takes O(NlogN) time the asymptotic performance is not affected making the indexing dynamic.

The MVQL syntax uses lists of expressions for the coordinates of the FIRST and SECOND objects and of the ranges, for example

arcPairs := kD−PAIR((Arc,largeArcs),{FIRST[Angle]=SECOND[Angle]} ,{FIRST(X Y), SECOND(X Y), RANGES(distx disty)})

This constrains the X coordinates of the paired arcs to differ by less than *distx*, and the Y coordinates by *disty*.

As an indication of the speed up this gives consider pairing 8000 arcs with themselves. In 2–D without spatial indexing the timing for finding 500 matches is 17.1 times slower than with the indexing. In 4–D the benefit for a similar search is 22.5 times speed–up. Using uniformly distributed random values we found that the time to retrieve 500 matches from the 8000 pairs was 4.5+0.4 k log k seconds for dimensions k=1 ... 10 on an SUN–4 workstation. The dependence on k log k arises from the need to traverse an increasingly sparse tree and copy k values from it.

2.3 Comparison with SQL

The MVQL PAIR BIN and SELECT operations correspond approximately to the join, project and select operations in a relational database. In SQL terms the equivalencies are broadly

SQL	***MVQL***
SELECT FROM A,B... WHERE C AND D	*PAIR((A,B),{C}) SELECT D*
SELECT FROM A,B... WHERE C AND D AND E	*kD−PAIR((A,B),{C} {D})*

SELECT E
SELECT FROM A, GROUP BY B HAVING C *BIN(A,{B}) SELECT C*

The SQL is very much more general, a statement of what should be done than of 'how'. Nevertheless these three simple operators provide a powerful formalism for forward productions. The simplicity of the MVQL query structure has three advantages

1) enables efficient processing of large amounts of data and

2) queries can be easily costed. That is, it is easy to predict for a given MVQL specification the number of operations implied and hence the computational cost.

3) MVQL can be interpreted by a relatively simple query processor.

2.4 Implementation of the dbms

Subject	Relation	Object	Tagfields (32 bits)
Image	HasWidth	512	
Arc	HasX	42	

Figure 1. Triple store schema

Our implementation is based on a triple store: each data item is entered as a the target of some relation to an entity in the form (subject,relation,object), see figure 1. A general scheme that inserted the entities created by the user's queries (eg sets of features) into the triple store would require a fairly complex management system. (Typically the user needs to be kept informed of the space he is using, and needs to be able to remove unwanted triples). Instead we have a fixed number of tag fields in the triple store which can mark triples. User defined sets are then represented by tags at points in the triple store relevant to the entities grouped, rather than new triples, along with a representation of the processing needed to re–instantiate the sets. The time and space requirements for re–instantiation of a set are reduced compared with the initial creation time because only tagged triples are processed, and because only those attributes actually needed are instantiated.

To provide some modularity in the system we have split the implementation into three broad parts. Fig 2 shows a block diagram of the system. At the lowest level we have C–routines to issue identifiers for nodes and relations, to convert lexicals to identifiers and vice versa, and to insert triples in the triple store. These routines are based on the Sierra 1 software from Essex University [8]. At the next level, linked by a procedural interface is a

C–program that takes a language (the Table Processing Language TPL) input. This language is specified as a Yacc grammar. The TPL creates temporary tables holding data extracted from the triple store. Various operations on the tables can be specified, including sorting on columns and triples tagged as specified. At this level the inheritance of attributes is implemented (single inheritance), and simple display routines provided. At the highest level is a Lisp program that reads the user's MVQL (again a Yacc specified grammar) and generates a series of TPL commands.

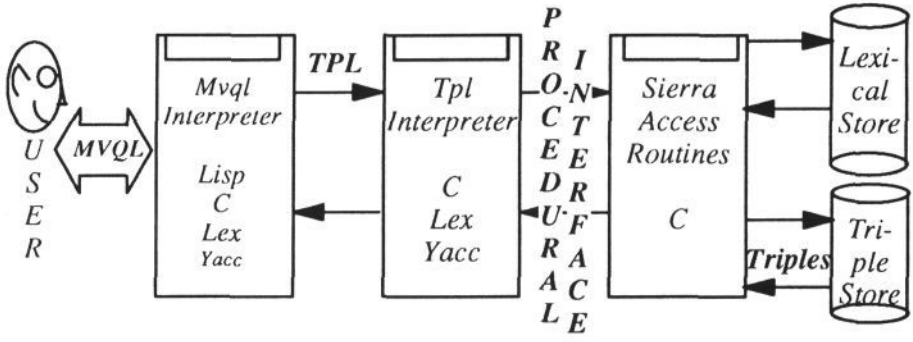

Figure 2. Modules in the CAID

For example the fragment $x + R\ cos(theta)$ in an MVQL statement would cause a table with columns for the x,R, and *theta* data to be created and a sequence of TPL commands to take the cosine of the data in *theta* column putting the result in a scratch–pad column, multiply this column by the R column etc. The result would be a column of values for the resultant attribute. It is the responsibility of the Lisp program to keep track of the meaning of the tags and tables.

3. Example Application

Microfossil images are used in the dating of geological samples. We can use the CAID system to to detect and classify microfossils, and so go some way to providing an automated geological sample dating system.

3.1. Microfossil Detection

The primitive features chosen were arcs, characterised by a position orientation and curvature; for each of 100 images 400 arcs were detected and stored. The images were annotated with areas of interest (AOI) by a geologist, each useful microfossil being assigned an AOI with associated position and radius.

Figure 4. Image showing microfossils.

As the fossils tend to be associated with the arcs detected a very simple detector would group the arcs by image and by position on some coarse grid within the image. The arcs are thus made to "vote" for grid squares. We accept grid squares with many votes as possible detections. By pairing the possible detections with AOIs we can find the

numbers of those that lie on the fossils and those that lie off the fossils. By varying the threshold on the number of arcs voting for a detection we can form performance curves for the false positive rate against the true positive rate. However the performance of this scheme proved to be poor. This was partly because many of the microfossils have roughly circular walls with strong contrast but rather little high contrast texture in the centre. The circular Hough Transform is a better detector for most microfossils

We can implement a projected circular Hough transform by making arcs vote for grid squares cut by a line segment normal to the arc covering the expected range of fossil radii. For a line segment of length *minrad* to *maxrad* we can use

```
votes:= PAIR( ((Array),(Arc)),{TRUE}) AT SECOND[Image]
        SELECT ABS((gridX-arcX)*SIN(arcAngle)-(gridY-arcY)*COS(arcAngle)) <
binsize
                                AND
ABS(ANGDIFF(ATAN2(gridY-arcY,gridX-arcX),arcAngle)) < PI/3
                                AND SQR(gridX-arcX)+SQR(gridY-arcY) < SQR(maxrad)
                                AND SQR(gridX-arcX)+SQR(gridY-arcY) > SQR(minrad)
```

Here *Array* is a fixed set of Hough cells inserted in the database. The grid and arc position references used here are the macros

```
gridX=FIRST[X]
gridY=FIRST[Y]
arcX=SECOND[TRUNC((X DIV 51)*51.2)+25]
arcY=SECOND[TRUNC((Y DIV 51)*51.2)+25]
```

The set *votes* associates each grid square in the array with the strong arcs in that square, regardless of image. We can group the *votes* by image using the bin operator, discarding low occupancy grid squares:

```
occupancyThresh= 15
hough:= BIN(votes,{ Image FIRST[Base]}) AT Image SELECT COUNT > occupancyThresh
```

To find the grid squares that are close to AOIs we pair them together. Again we use macros for clarity:

```
hx= SECOND[COMMON[gridX]]
hy= SECOND[COMMON[gridY]]
aoiX= FIRST[X]
aoiY= FIRST[Y]
aoiRad= FIRST[Radius]
```

in terms of these we can write the set of all detections as:

```
detects:= PAIR( ((AOI),hough),{FIRST[Image]=SECOND[Image]}) AT FIRST[Image]
                                                    SELECT
SQR(hx-aoiX)+SQR(hy-aoiY)<SQR(aoiRad)
```

The cardinality of this set is the total number of grid squares that lie close to fossils, so we can find the number of fossils detected from the cardinality of the following set:–

```
truePositives:= (AOI) SUPPORTING detects
```

Fig 5 shows the detection of two fossils. Note that the graphics were generated by the graphical output facility of MVQL. The performance of this type of detector is discussed in [2].

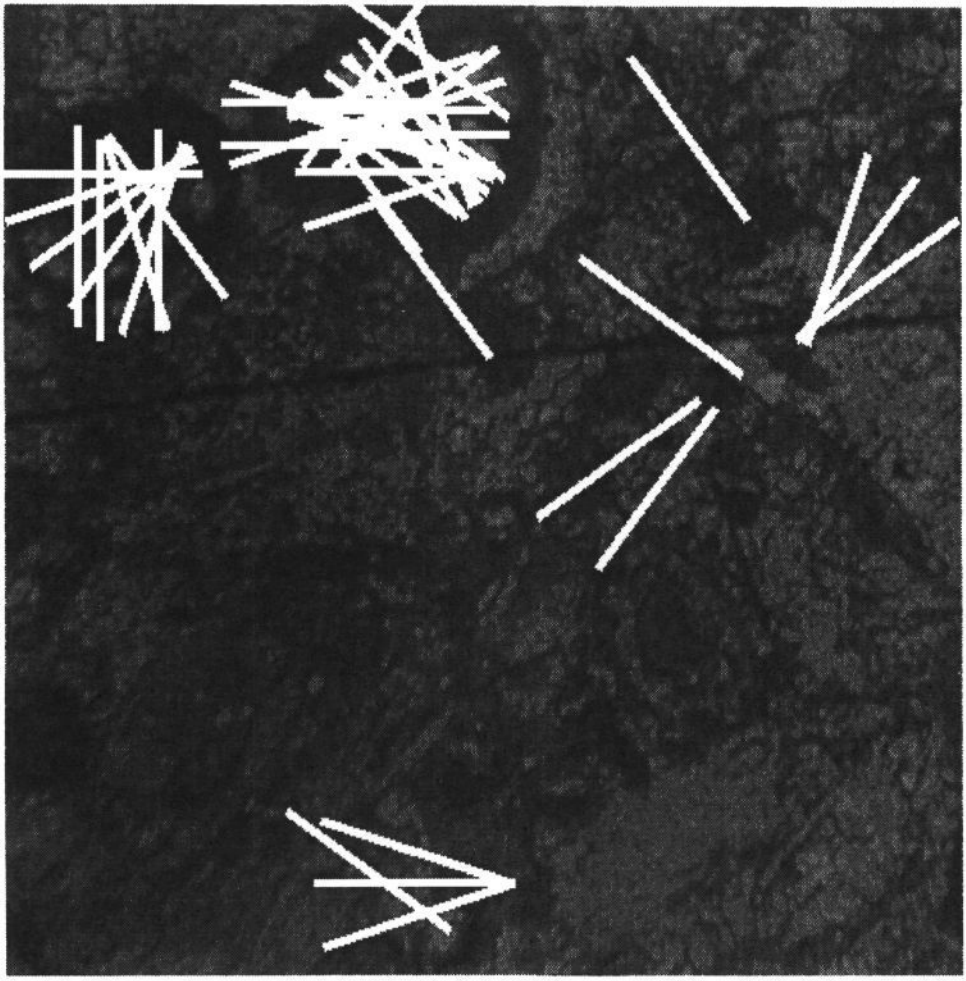

Figure 5. Image showing detection of 2 microfossils using the projective circular hough transform.

4. Conclusion

We have developed an application–independent content addressable image database, that addresses image informatics and computer vision problems. The query language in this dbms allows the user to handle large data volumes with predictable performance. Application studies have shown that it facilitates machine vision research by providing a uniform interface to image data with tools to search and group the data efficiently.

5. Acknowledgements

This work was done under a grant from the UK SERC. We acknowledge the IFS group at the Department of Computer Science of Essex University for the Sierra Software. Geological expertise and microfossil image annotation provided by Dr F.M. White of The Department of Geology, with funding provided by the University of Manchester.

6. References

1. J.P. Oakley, D.N. Davis & R.T. Shann, "The Manchester Visual Query Language", *SPIE Proceedings, Vol. 1908, Storage and Retrieval for Image & Video Databases, pp. 104–114, 1993.*

2. R.T. Shann, J.P. Oakley, D.N. Davis and F.M. White. Detection of circular arcs for content–based retrieval from an image database. *I.E.E. Vision, Image & Signal Processing. Vol. 141, pp. 49–55, 1994.*

3. Bentley, J.L. [1975], "Multidimensional binary search trees used for associative searching," Commun. Ass. Comput. Mach., vol. 19, pp.509–517, Sept. 1975.

4. Bentley, J.L. [1979], "Multidimensional binary search trees in Database applications," IEEE Trans. Soft. Eng., vol. SE–5, NO. 4, July 1979.

5. J.N.D. Hibler et C.H. Leung, K.L. Mannock and M.K. Mwara, A system for content based storage and retrieval in an image database , *SPIE, Vol. 1662 Image storage and retrieval systems, pp 80–92, 1992.*

6. W. Niblack R. Barber W. Equitz M. Flickner E Glasman D Petkovic P Yanker C Faloutsos G Taubin, The QBIC Project: Querying Images by Content Using Color, Texture and Shape *SPIE 1908 p173–188 (1993)*

7. M. H. O'Docherty C.N.Daskalakis, P.J.Crowther, C.A.Goble. M.A.Ireton, J.P.Oakley and C.S.Xydeas, The design and implementation of a multimedia information system with automatic content retrieval , *Information Services & Use*, vol 11, pp 345–385 Elsevier, 1991

8. S.H. Lavington and J. Wang, "The external procedural interface (EPI) for the IFS/2," Dept. Computer Science, University Of Essex, Report CSM–164, June 1991.