A Spatial Database Manager for a Generic Image-Understanding System

D. C. MASON,*,† C. ODDY,‡ A. J. RYE,‡ S. B. M. BELL,† M. ILLINGWORTH,† K. PREEDY,‡ C. ANGELIKAKI,† and E. PEARSON†

GEC Marconi Research Centre (MRC), Space and Defence Research Laboratory, West Hanningfield Road, Great Baddow, Chelmsford, Essex, U.K., and Natural Environment Research Council Unit for Thematic Information Systems (NUTIS), Department of Geography, University of Reading, Whiteknights, Reading RG6 2AB, U.K.

Received January 14, 1991

The design of a spatial database management subsystem for an image-understanding system for multisource image processing is considered. The image-understanding system is a generic knowledge-based architecture providing a basic environment for fusing spatial information from a number of sources in order to produce an improved image description. The database manager copes with a variety of spatial data models, including raster data from a variety of sensors and vector data from image segmentations and maps, within a tiled image format. Facilities for relating raster and vector data are provided. An image set database handles relationships between objects in images which have been fused. The database system encompasses certain low-level image-processing utilities including a data flow control module for local windowing operations and a connected component finder.

INTRODUCTION

Since the 1960s there has been considerable interest in computer-based methods for simplifying and describing images from a variety of disciplines, including earth resource management, medicine, biology, physics, manufacturing, defense, and law enforcement. The science of image understanding has evolved which can employ a substantial battery of techniques to obtain a detailed description of an image by partitioning (segmenting) the image into its constituent spatial entities and assigning a class label (or labels) to each entity. Once an image has been interpreted in this fashion, the segmented image may be used for many other purposes, such as studying the spatial relationships between different classes of entity.

MuSIP is an image-understanding system for *Mu*lti*S*ource *I*mage *P*rocessing which is being developed within a European ESPRIT II project involving a multinational collaboration. A main aim of the MuSIP project is to develop a generic knowledge-based architecture providing a basic environment for combining image information from different sensors and other spatial information such as maps, in order to provide a unified description of an imaged scene. Demonstrators are being produced illustrating the application of image-understanding techniques to fused data sets in the two disparate fields of earth resource remote sensing and medical image processing. The earth resource application is the monitoring of change in forested regions using multitemporal satellite imagery. The medical application is the fusion of images of the human brain obtained from PET (positron emission tomography) and NMR (nuclear magnetic resonance) scanners. The generic nature of the system means that it could be applicable to spatial analysis problems in images pertaining to other disciplines, for example, in the study of the interactions between different molecules.

This paper discusses the design of the spatial database management subsystem for MuSIP. A large number of spatial processing and information systems have now been developed within a variety of disciplines. Each system contains some form of pictorial database management system (PDBMS). The term PDBMS is taken here to mean a system in which a large amount of spatial information is stored together with related nonspatial information in an integrated manner. A PDBMS must contain a DBMS as well as an image database, thus ruling out straightforward image archives or systems aimed at retrieval of secondary information about images rather than the images themselves. A substantial literature now exists on PDBMS (see reviews in refs 1-3). The picture understanding DBMS (PUDBMS) considered here is a special form of PDBMS for use in an image-understanding system. A PUDBMS tends to be biased towards the use of raster data, because one of the data sets being operated on is always an input image. However, it is often necessary to integrate other data types with the image, e.g., vector data from maps. The PUDBMS of a number of image understanding systems have been described previously (see, e.g., refs 4-9). McKeown[10] has pointed out that the databases of such systems have often been presented rather cursorily, partly as a result of the database problem not being regarded as a research issue in its own right.

THE MUSIP SYSTEM DESIGN

An overview of the MuSIP system is given in order to set the context for the database management subsystem. Details of the MuSIP architecture are given in ref 11.

MuSIP is designed to cope with analysis of the complex scenes typically associated with the earth resource management and medical imaging fields. In addition to problems of image complexity and noise, there is also the problem that objects in the image may not be modeled exactly, as their appearance and arrangement in the scene is often unpredictable. In common with earlier image-understanding systems designed for specific image scenarios, the approach employed within MuSIP involves the integration of numerical image-processing techniques with knowledge-based techniques. A number of different types of knowledge are exploited. One type of knowledge is a priori knowledge about the scene being imaged, for example, the knowledge contained in a digital map. A second type is procedural knowledge, enabling the dynamic selection of the optimum sequence of algorithms to apply for a given application and set of sensor types. This type of knowledge encompasses knowledge about algorithms and sensors, and might include, for example, knowledge of the best type of algorithm to use to detect edges in images of a particular sensor type. MuSIP employs a knowledge-based supervisor which manages subsystems operating at the various processing levels dynamically.

† University of Reading.
‡ Space and Defence Research Laboratory.

0095-2338/91/1631-0290$02.50/0

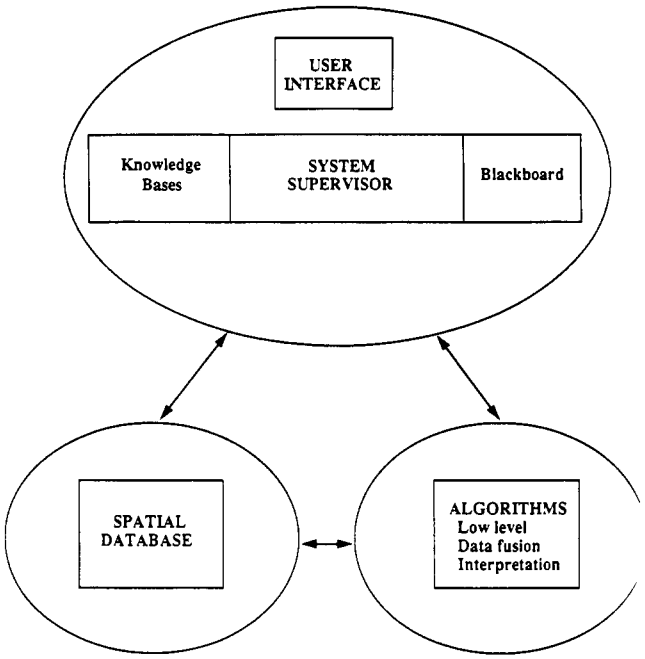

Figure 1. MuSIP system diagram.

Figure 1 illustrates the major structural components of the MuSIP system. These include a spatial database manager, a large collection of image analysis algorithms for low-level image processing, data fusion, and interpretation, a supervisor to control the system, and a user interface.

Central to the MuSIP design philosophy is the modular nature of the system. This allows concurrent processing of modules forming objects and easy expansion of the system. Each object (algorithm) in the system has the same generic interface to the supervisor. The interface is made up of an object description file which describes the object in sufficient detail to enable the supervisor to assess its suitability for the current task, and a command description table which contains a list of the commands that the object can understand. Data are exchanged between objects using asynchronous message passing.

The supervisor is responsible for breaking down the operator goal in a structured way and, by use of the object description files, producing a strategy to satisfy this goal in the form of a tree of algorithm objects. The supervisor controls the system by issuing commands, which include the invocation of the algorithms in the tree. Dynamic assessment of the situation after a particular object or group of objects have performed their processing allows the supervisor to follow an optimum path through the tree. Knowledge contained within knowledge bases is used for goal selection and situation assessment to determine if any backtracking in the processing tree is required. It is the responsibility of the backtracking module to identify not only when backtracking is required but also where to backtrack to in order to achieve an improved output. A blackboard manager controls access to a blackboard upon which global features of the scene being analyzed are written. Because it can be voluminous, all image data is read from and written to the database and is not duplicated anywhere within the system. Each knowledge base is associated with a distinct knowledge domain, so that knowledge about sensors is held in a different knowledge base from knowledge about algorithm behavior.

The functionality provided by algorithm objects falls into three categories, namely, low-level image processing, data fusion, and image interpretation. Low-level image processing includes such operations as image smoothing, filtering, edge detection, region growing, and region finding. Another central concept of MuSIP is that improved image interpretation may be made if data from different sources are fused together. The data to be fused may be images from different sensors, a sequence of images from the same sensor, or an image and other spatial data such as a map. Initial segmentations of the data sources produced by low-level algorithms are first subjected to prefusion interpretation to assign initial class labels to each region. The different segmentations are then combined in a fusion process, each cycle of which produces an image aggregate and an error set of unmatched regions. The fusion process is based on attributes of regions and on structural information regarding their neighbors. In the first instance, regions are fused on an intraimage basis, then on an interimage basis.

The final processing stage is interpretation, the role of which is to attach a class label to each matched region, together with a belief measure dependent on the quality of the supporting evidence. Labels are selected from an existing hierarchy of labels for each application. Access to application-specific knowledge enables the interpretation to generate scene descriptions for any sensor and application. Changes may also be detected by using the error set of unmatched regions or the changing attribute values.

DESIGN OF THE SPATIAL DATABASE MANAGEMENT SYSTEM

The design of the spatial database management subsystem for MuSIP is based upon a number of requirements.

(1) The system must support a variety of spatial data models. The two basic models involved are raster data from scanners and other remote sensors and vector data from segmented images and maps. Raster data consist of two-dimensional arrays of attributed pixels, where the pixel attributes are single- or multiband grey values. The spatial entities of vector data are points, sinuous arcs, and sets of closed arcs, all of which are constructed from (x,y) coordinate pairs. These spatial entities have associated nonspatial attributes (e.g., entity class). These spatial data models tend to be associated with the two ends of the processing chain, the raw raster data with the input, and the topologically structured vector data with the output. Other spatial data models also need to be handled, such as models for compressed raw raster and generalized raster (e.g., run-length-encoded or hierarchical data models) for use at intermediate stages. It must be possible to relate different spatial data types easily and efficiently. Interconversion between different types is also required.

(2) To cope with the remote sensing case, it must be able to handle very large images (complete satellite scenes of say 6000×6000 pixels or more containing multiple image bands) within the restrictions of limited computer memory and disc space and the requirement of minimum disc input/output. (In contrast, the images used in medical image processing tend to be much smaller, typically 512×512 pixels). Various methods for handling large images have been adopted previously, such as image tiling using spatial proximity in file organization to exploit the fact that many local operations are generally required.

(3) While it is required to cope with the majority of functions handled by a spatial information system, it must be particularly efficient at performing operations extensively used in image segmentation. Because the spatial operations involved tend to be local, local search operations need to be fast. Also, common segmentation operations involve assessing primitive regions to test whether they should be split or adjacent regions to test if they should be merged. Therefore, the database needs to be easy to update, as it undergoes constant modification during segmentation.

(4) The main access route is via low- or high-level image-processing modules rather than manually, so that facilities for

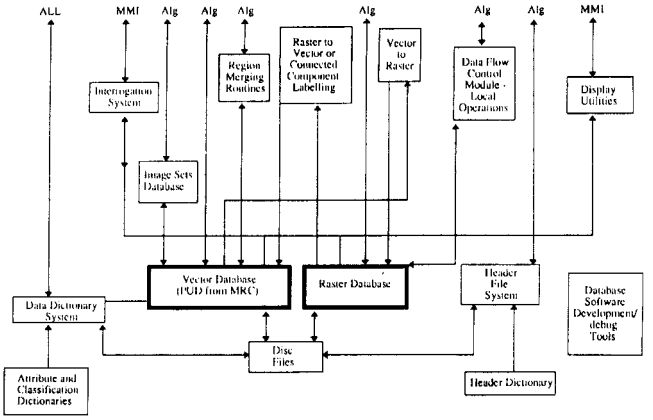

Figure 2. Database subsystem diagram.

user interaction (e.g., a spatial query language) are of less importance.

(5) Hierarchies of entities (objects) must be supported, so that objects may be contained within other objects, or themselves contain other objects.

(6) In order to store the results of the fusion process, it must handle relationships between entities across sets of related images, so that the correspondence between entities in one image with those in a second may be extracted.

(7) In the present system there is no requirement for data models to have more than two spatial dimensions.

(8) The subsystem should encompass commonly used low-level image-processing utilities making extensive use of the database.

There are difficulties in adapting conventional database models for use with image databases, due mainly to the characteristics of spatial data. A number of PDBMS have been developed on the basis of the popular relational model. However, there are a number of drawbacks to handling spatial data by using this model. Among these are the facts that access times for spatial information tend to be slow, that the model does not support the necessary spatial data types, and that object hierarchies are difficult to implement. The extended relational model has been developed to overcome these difficulties and has been incorporated into systems to manipulate geographic data (e.g., POSTGRES[12]).

There are also disadvantages in using the network model for spatial data. This model is difficult to set up and organize and is rather inflexible, as all relationships between entities must be stated explicitly. However, it is more suited to handling spatial data as the system of pointers permits fast access and retrieval. This advantage is exploited in the TIGRIS system developed by Intergraph for handling raster and vector data.[13] For MuSIP, it seemed appropriate to trade flexibility for fast access, and the database design is based on the network model.

The PUDBMS for MuSIP consists of a number of components (Figure 2). Its basis is Marconi Research Centre's PUD (Picture Understanding Database) DBMS, which copies with vector data. In order to satisfy the requirements of MuSIP, it has been necessary to extend PUD in a number of ways. The extensions include the ability to handle other data types such as raster data and sets of images, the provision of facilities for relating vector and raster data, user interface, and image display facilities, and a number of low-level image-processing utilities. The resulting DBMS for MuSIP is called MuSID (MultiSource Image Database). The various components are discussed in the following sections.

Vector DBMS. The PUD DBMS is designed to store topologically structured vector information derived from segmenting images and map information.[14,15] The information stored in PUD is essentially two-dimensional. Three basic

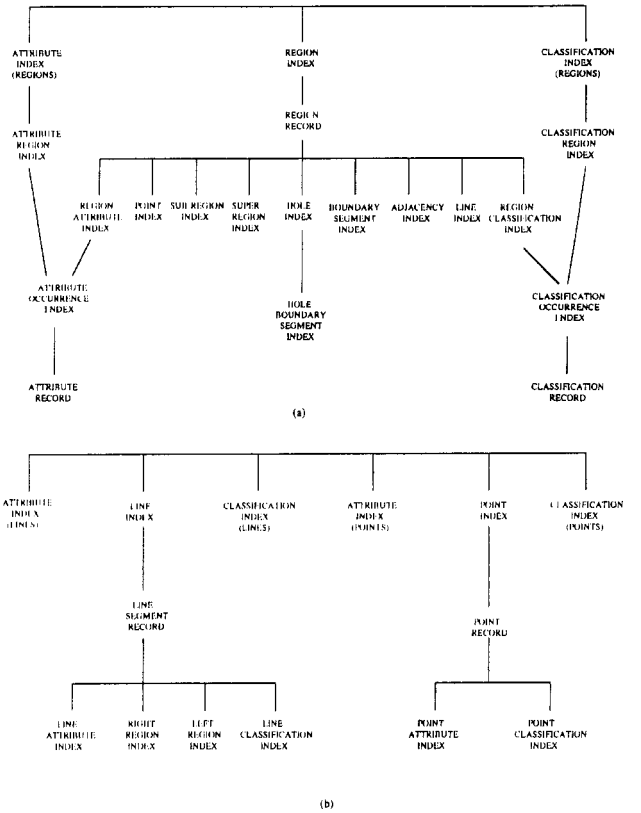

Figure 3. Vector database schema for regions (a) and lines and points (b).

spatial entities are handled, namely, regions, lines, and points. All features extracted from images and contained in maps will fall into one of these categories. High-level image analysis techniques require rapid access to entity information (such as region attributes, boundary code, etc.) and to relations between entities. PUD is designed to store this information in a flexible manner, in order to allow the data to be accessed in a number of different ways.

(a) Database Structure. The structure of PUD is shown in Figure 3. The database is basically a tree structure, with additional pointers between branches to make certain data access paths more efficient. Each of the three spatial entities (regions, lines, and points) is a top-level index in the hierarchy, and for each there is also a top-level attribute index and classification index. Each type of top-level index, and its descendants, is described below.

(*i*) *Region Index.* The region index contains identifiers for all the regions in the image. Each region is identified by a unique region identifier. For each region, information is stored on its bounding rectangle, boundary, adjacent regions, holes, and attributes (e.g., perimeter). In addition, the facility for storing subregions and superregions is available. This facility is useful when it is required to build up hierarchies of objects, e.g., a superregion representing a forest may be constructed from a group of forest regions divided by paths.

The region attribute index gives access to information about all attributes which have been calculated for a region. This index would be used, for example, to find the area of a given region. In contrast, the attribute region index gives the regions for which a particular attribute has been calculated. This index could be used to find those regions for which, e.g., a grey value standard deviation had been calculated. This allows access to attribute values by attribute identifier as well as by region. Attributes may be scalar or vector quantities, and if the latter may be of variable length. An occurrence index allows PUD to contain more than one computation of a given attribute (e.g., obtained by using a different algorithm).

A separate region classification index allows access to a list of possible classification categories for a region, such as wheat, field, urban, etc. These may be mutually exclusive alternatives (e.g., wheat or barley) or may be compatible classes (e.g., field, wheat) so that hierarchies of classes may be built up. A probability may be associated with each possible classification. As with attributes, there is a complementary classification region index. There is also a classification occurrence index. The meanings of attribute and classification identifiers are stored in a dictionary to allow controlled expansion.

The boundary segment index contains the identifiers of the line segments making up a region's boundary. Each segment has a unique line identifier. The region boundary may be traversed in an anticlockwise fashion by accessing successive entries in the index. Clockwise boundary segments are distinguished by storing the negative of the line identifier in the index.

The region line index lists those lines which pass through the region (but not region boundaries). Similarly the point index lists the identifiers of points contained within the region.

(*ii*) *Line Index.* The line index is used for storing region boundaries and also other lines which are not part of a region boundary (e.g., lines supplied in vector form from maps). Attributes may be associated with lines (e.g., edge strength) by using the line attribute index. There is also a separate line classification index. A line may be represented in one or more of a number of possible forms, including Freeman code or a polygonal approximation, by storing its description as a line attribute. As lines may pass through several regions, the left and right region indexes may be a list of regions to the left and right of the line rather than one of each.

(*iii*) *Point Index.* A point index stores the location, attributes, and classifications of point features. The identifier of the region containing the point may also be stored.

(b) Database Record Types. A PUD database consists of two types of record: index and data. Index records are further subdivided into hierarchical and cross-pointing index records.

Hierarchical indexes are those which point to items that are the immediate descendants in the tree structure. The management of this type of index is performed by PUD, and identifiers are inserted and deleted automatically when the relevant descendant record is created or deleted. The user is able to read the information in the index and any associated data values, but no further access is allowed or required.

Cross-pointing indexes are those which point to items which are not immediate descendants in the tree structure. These indexes are always leaves of the tree (i.e., have no immediate descendants). The identifiers of such indexes must be inserted and deleted by the user, and greater access facilities are available than for hierarchical indexes. Many cross-pointing indexes store relationships between pairs of features, and there are corresponding indexes or data records which store the reciprocal relationship. For example, the region boundary segment index stores the identifiers of line segments making up the region boundary, and the line index stores the left/right region index. For such reciprocal relations it is only necessary for the user to insert or delete one of the relations and the reciprocal relation will be handled automatically.

Data records contain data, pointers to immediate descendants, and cross-pointers. Data records are accessed internally via index records, though this is invisible to the user.

(c) Database Access. A suite of subroutines is used to interact with a PUD. These allow interacting software to create, read, delete, or update a record. These routines can be used to perform any operation on a PUD. Only one call is needed to access a selected record, and all the identifiers needed to traverse the database tree to this record must be supplied in the call. Access to the data is in a top-down fashion. For example, to read the boundary code for a particular region the user would make a call which first read the region record to find the pointer to the boundary segment index, and then read the boundary segment index to get the identifiers of the line segments making up the boundary. Further user calls would then be made to read the line segments themselves.

(d) Database Tiling. PUD is able to store and process very large images which are too large to fit into the available memory. A vector image is split into a number of tiles, and each of these contains all the information on the regions, points, and lines within the tile boundary. Only selected tiles are stored in memory at any one time, and tiles are swapped in as required and swapped out on a least recently used basis. Each tile has a unique identifier, and this forms part of the unique composite identifier of each region, line, and point in the image. The tiling is invisible to the user unless (s)he wishes to know about it.

Tiles are created at the time of image segmentation. Tile boundaries are defined by the user, either by entering a rectangle tile size, or a mean region size and a mean number of regions per tile. A region is placed in the tile which contains the center of its frame. A line is placed in the tile containing its start point. A point is assigned to the tile containing it. Note that by these rules a line segment which is part of a region boundary may be contained in a different tile than that containing the region.

A number of methods of accessing entity information within tiles exist, including access by identifier, by location, and by index scan. Access by identifier is straightforward as the identifier of the tile containing an entity is part of the composite identifier for the entity. Access by location involves a search of the tile boundary information stored in the database master file. For access to entities by index scan, tiles are processed sequentially in an ordered raster fashion so that if further operations to be performed on each item of the index being scanned involve accessing adjacent information, then nearby tiles are likely to be already stored in the buffer.

(e) Database Storage in Memory. A single large buffer contains all the database data held in memory at a given time. In many applications, data from several databases (e.g., an image and a map) may be required to be operated upon within the same program. To cope with this, the buffer is split into N parts, where N is the number of images and maps to be processed. Different parts may be of different lengths in order to reflect relative priorities among the various images for the particular processing strategy being followed. Each section of the buffer generally contains data from a number of tiles of the image associated with that section. Dynamic storage allocation is used to control the buffer. Each record within a tile contains data which is contiguous within the buffer. A number of chains of free slots of various size ranges are also contained in the buffer for each tile. In order to insert a new record, a free slot which is large enough is found, and part of this is used to store the new record. When a record is deleted, the empty slot released must be added to the free-chain store for the tile.

(f) File Structure and Management. The PUD database for a single vector image consists of one file for each tile in the database. In addition, a master file holds information on the tiling structure, including the rectangle covered by each tile. All these files are held in a common directory. There is also a separate related header file to hold ancillary data describing the database (e.g., image frame, image sample spacing, etc.) (Figure 4). The header file also holds information on the processing history of the image. This header file is in free format so that it can easily be read by a user but is also compilable to a form suitable for machine interpretation.[14] A

```
HEAD_POINTER = SHOTLEY.IMG;
/* SAR Image of Shotley */
TYPE DEFINITION:
        IMAGE TYPE = (RASTER);;
FRAME:
        BHLC = (5,5);
        TRHC = (260,260);
        SAMPLE RATES = (2,2);;
PIXEL DEFINITION:
        BITPIX = (16,16);
/* a two component image, each component 16 bits */
        TYPES = (I,I);
/* 2's complement integers */
        NAMES = (RED,BLUE);;
TILE DEFINITION:
        TX = (128);
        TY = (128);;
/* history */
        DERIVATION:
        USER_INPUTS = (PSUBPICT, LSHOT, 5,5,262,262, 1,3,2,2, SHOTLEY);
        INPUT FILE1:
        INPUT_FILENAME = LSHOT.CHD;
        USER INPUTS = (...);;;
```

Figure 4. Example header file for a raster image.

dictionary of known keywords is used to assemble the compiled header.

Raster DBMS. The ability to handle multiband raster images is an important extension to PUD. To match the tiling of the vector data, the raster data must also be tiled. To create the tiled vector structure, the raster image is scanned in a non-raster ("tiled-raster") scan order, by performing a raster scan within each image tile and examining the tiles themselves in a raster fashion. One important consequence of this scan order is that special routines for image processing are needed, e.g., for connected component labeling, it is necessary to transmit partially detected objects across tile boundaries (see Connected Component Finder).

The file structure of a raw raster image is such that it is suitable for processing in a tiled-raster fashion. The raw data is reformatted and held as a single file in a form which is tile sequential and band sequential within each tile, i.e., the order is tile 1 band 1, tile 1 band 2 ..., tile 1 band n, tile 2 band 1, tile 2 band 2 ..., tile 2 band n, and so on. This structure allows easy access to any single band of a tile in a multiband image without accessing any other bands. Alternatively, several bands in the tile can be accessed at one time without much physical disc head movement.[16] Images may be held in a variety of data types, including bit, byte, integer, and real. As with vector data, an ancillary header file is associated with the raster image (Figure 4).

Raster data may also be held in compressed form on disc. Raw raster imagery can be subjected to "error-free" compression, by coding the differences between adjacent pixels' grey values in each band of each tile line. Because the differences are less correlated than the original grey values, it is possible to achieve compressions of up to ×3.5 by such methods, which allow perfect reconstruction of the original image.[17] Satellite images in particular are often of enormous size—a complete Landsat Thematic Mapper image occupies over 300 Mbyte—and a large reduction in disc storage can be achieved even for modest compressions. Processed raster data which have been generalized in some way (e.g., classified, so that many adjacent pixels have the same class value) may be compressed by storing them in run-length-encoded form. For both these compressed raster types, image tiles are of variable length on disc, and each tile is held as a separate file.

For raster data, the logical interface between the PUDBMS and the user is an "interval" of data. For 2-D images an interval is a connected segment of a complete image line, typically a tile line. On image output or input, the user is either presented with or may provide an array of pointers, each of which points to the start of the unpacked grey values for the interval for each image band. On reading, the PUDBMS plants the relevant grey values at these locations, while on writing it picks up the input grey values from these locations.

Within the in-core tile buffer, raster tiles may be of variable length in the same way as vector tiles. This provides the ability to read parts of tiles adjacent to other tiles which need to be bordered, e.g., during processing to detect edges in an image (see Data Flow Control Module).

Relating Raster and Vector Data. It is necessary to relate raster and vector data easily and efficiently. A common need for this is when initial estimates of segmented regions have been produced in vector form and it is necessary to measure region parameters (such as mean intensity) by referring back to the grey values of the original raster image. These parameters would then be used in the modification of the initial segmentation. Another example is when an image and a map must be combined. A number of complementary facilities are provided.

(1) The usual method of relating raster and vector data is to use a region map, i.e., a raster image containing the covering region label at each pixel. Given a particular position, this allows the covering region identifier to be determined. The region map may be stored on disc in runcoded form, as it is a generalized image containing identifiers which may become very large (e.g., in the case of complete satellite images).

(2) An important requirement is to be able to perform a raster scan of the pixels associated with a region domain, for example, to calculate some feature of the region (e.g., mean intensity). While the shape of the region is encoded in the region map and the region boundary, neither of these data structures is particularly suitable for a raster scan of the region. For example, one method is to scan all the pixels in the region frame in the region map in order to select those pixels belonging to the region. However, this would be very wasteful for certain shapes (consider a long thin object at a 45° orientation to the horizontal). An alternative data structure which is suitable is an "intervals list", a form of 1-D run-encoding.[18,19] An interval in a line is defined by two x-positions xb_i and xe_i, marking the start and end positions of a connected segment of the region in that line. For each line the region may have a number of such intervals (xb_1,xe_1), (xb_2,xe_2), ... (Figure 5). Traversing the intervals of a region in order allows a raster traversal of the region. In order to allow random access into the intervals list, an auxiliary "interval pointers list" gives the position in the intervals list of the first x coordinate in the ith line of the region. Algorithms exist for performing most image-processing operations using intervals lists. The intervals list is maintained as an auxiliary data structure and stored as a region attribute. It is used in conjunction with the region frame. Regions are thus represented in both raster and vector forms, and the form chosen for a particular operation is that which is most efficient. A number of previous schemes[20-23] have employed relational models which treat the raster and vector approaches to modeling the topology as equivalent alternatives.

The intervals list may be used to identify the grey levels in the raster image with which it is associated, because it is possible to calculate the raster file offset given position (x,y).

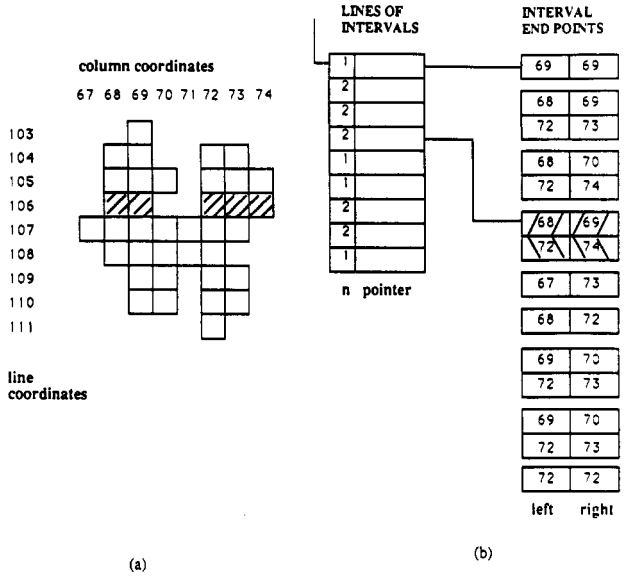

Figure 5. Region and its intervals list (after ref 19). (a) A region with two intervals shown shaded. (b) The corresponding intervals list domain structure. The shaded intervals correspond to those in (a). Not all pointers are shown.

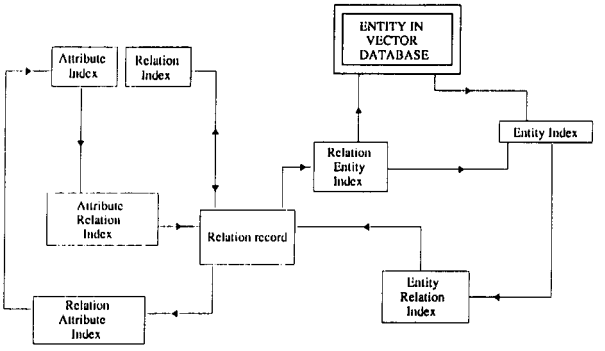

Figure 6. Image set database schema.

In this case, the description of a region domain based on its intervals list is linked dynamically with a description of its grey values within a single C structure. All the relevant information relating to this region can then be accessed via the single address of the structure.[19]

(3) It is also possible to store the grey values associated with a region directly with the region as an attribute.

Image Sets. It is also necessary to cope with sets of related images, e.g., images of the same scene taken with different sensors, or a sequence of images taken with the same sensor. In image fusion, it is often necessary to match a region or regions in one image with another in a different image and use the information to provide a fuller unified description of the scene. Another extension to PUD has been designed to hold the required information, namely, the image set database shown in Figure 6.

The image set DBMS is designed to use the PUD low-level routines but to allow maximum flexibility in relating different images. Three types of entity are used, set entities, attributes, and relations. A set entity is a point, line, region, or super-region from a particular image in the vector database. Two set entities from different images may have a relation between them, e.g., region A from image 1 may correspond to region B from image 2 in a one to one relation. No restriction is made on the number of set entities participating in a single relation, and n to m as well as one to one relations are possible. A relation may be qualified by attributes. Examples of relation attributes might be the following: match, no match, is different from, encloses, is enclosed by. A relation may also have numeric information held on it, for example, a confidence level.

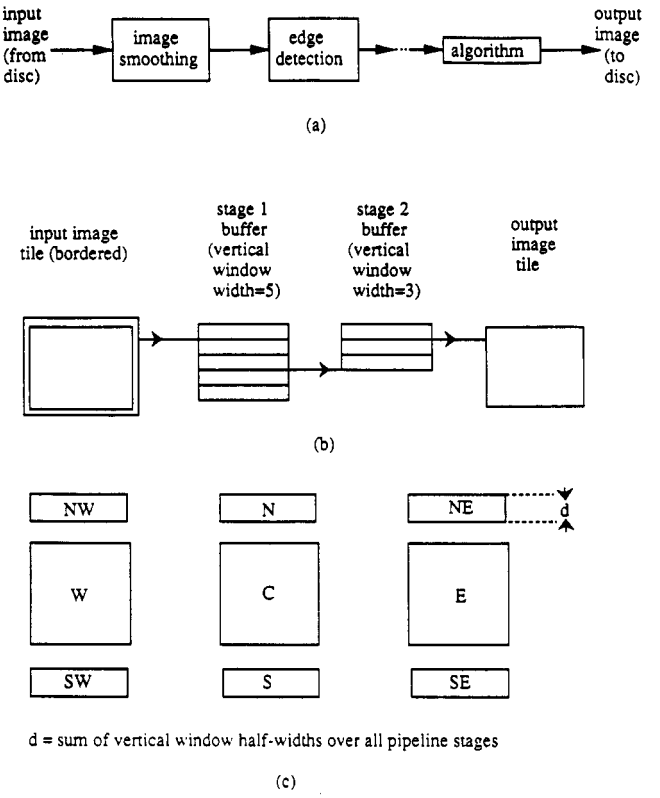

Figure 7. Data flow control module. (a) Algorithm pipelining. (b) Pipeline of temporary buffers (one band) for two pipelined algorithms. The first algorithm has a vertical window width of 5 pixels, and the second one of 3 pixels. (c) Neighboring tiles in core with current tile.

The image set database may be accessed on set entity, attribute, or relation. The key for a set entity is the same as the vector PUD database key for that entity, but qualified by the particular PUD from which it comes. From this key the relations in which the set entity participates can be found, as shown in Figure 6, and the attributes of these relations. The other set entities in the relation can then be found and traced to their original vector PUD database representations if desired.

Common Data-Handling Utilities. The database subsystem encompasses certain low-level image-processing utilities which make extensive use of the database. These cope with tiled images, while at the same time hiding the tiled nature of the data from the user. Of particular interest are the data flow control module and the connected component finder.

(a) Data Flow Control Module. Many low-level image-processing algorithms have a common form and are best supported by a data flow control module which provides data input/output facilities to a class of algorithms. The simplest class where this is required involves algorithms which perform local spatial operations using a rectangular window of pixels which is scanned across the image in raster fashion. At any window position, the grey value in the output image associated with the central pixel in the window is a function of the input pixel values in the window. These local operations include the following: spatial convolutions, smoothing, filtering, edge detection, edge thinning, edge thresholding, and others. Local operations are often chained together to form algorithm sequences of up to say 10 algorithms (Figure 7a). For example, an image may be smoothed iteratively five times with the same algorithm, edge detected, edge thresholded, edge thinned, and so on.

A data flow control module has the advantage that it allows algorithm pipelining, thereby saving disc I/O and disc storage requirements since intermediate results do not need to be saved

to disc. It also permits rapid algorithm prototyping and evaluation.[24] The alternative is to write each individual algorithm as a stand-alone program with its own input/output to disc.

The domain processed by the algorithms may be a complete tiled-raster image or an individual region in a PUD. The calculation of the central pixel value in the output image is performed by a user-supplied function called for each window position.

The local transform operations may be pipelined through a series of p stages, each involving a different local transform. Temporary buffers are set up for each stage of the pipeline (Figure 7b). The temporary buffer for the ith stage is $2ywini+1$ lines long, where $ywini$ is the vertical half-window width for this stage. A line is output from stage i to stage $i+1$ as soon as the first set of $2ywini+1$ lines of stage i have been processed. The initial domain is bordered by a distance equal to the sum of the x half-window widths of the pipeline stages in the x direction, and the sum of the y half-window widths in the y direction. This ensures that a border of processed pixels of sufficient width is associated with the domain at each stage of the pipeline.

If a complete tiled image is being processed, each input tile is subjected to all pipeline stages to produce an output tile before proceeding to the next input tile. Tiles are processed in a raster scan order. Each input tile must be bordered as described above. To speed up the processing, at the start of processing each tile, it and its W and E neighbors are brought into core (if not already there), and also "minitiles" containing the immediately adjacent tile lines to the NW, N, NE, SW, S, and SE (Figure 7c). The number of tile lines in each minitile is the sum of the y half-window widths. This ensures that each tile is read only slightly more than once in the processing of the image.

(b) Connected Component Finder. A basic operation performed by the subsystem involves partitioning the image into a set of disjoint regions. This is known as connected component finding. In the input image, pixels in the interior of regions must be marked with "0", and pixels having edges along their S, W, or S and W sides marked with 1, 2, 3, respectively (i.e., edges lie in the "cracks" between pixels). A region R is connected if any two pixels in R are connected by a chain of adjacent pixels each of which is in R.

The connected component finder operates over a domain which may either be a complete tiled-raster image or an individual region in a PUD. At the same time as finding connected components, the algorithm also finds certain features of each connected region in the domain, including its intervals and boundary lists, and its area, frame center, and perimeter. If the domain is a tiled-raster image, the input values are taken from this image and the connected components generated are output to a PUD. If the domain is a PUD region (i.e., its intervals list attribute), the input values are taken from the tiled-raster image containing the region and the new regions generated are added to the PUD containing the input region. Four-connectivity is used to demarcate regions.

If the domain is a tiled-raster image, the algorithm is similar to that given in ref 25, with the advantage that tiled vector output is allowed. Tiles are processed in raster scan order. In the first stage, for each line of the current tile, horizontal strips of "0" pixels in the interior of regions are run-coded into intervals. An interval begins either with a "2" or "3" pixel or the beginning of line, and ends with the last "0" before the next "2" or "3" pixel or the end of line. The component labeling algorithm given in ref 17 is then used to build up connected regions by searching for touching intervals in the current and previous lines.

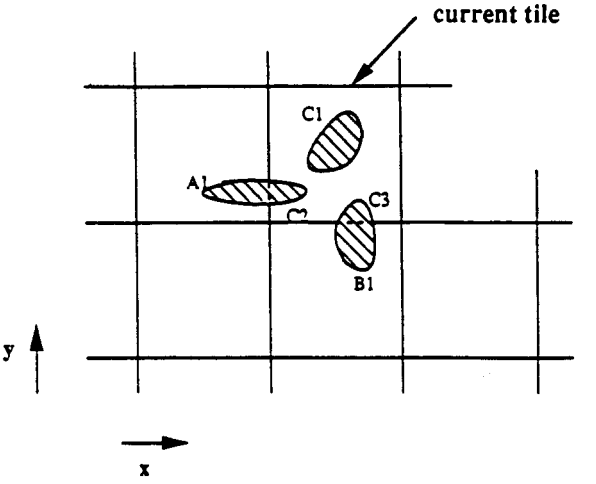

Figure 8. Connected component finder.

The next stage puts all the regions completely contained within the current raster tile into the corresponding PUD tile (e.g., region C1 in Figure 8). Any "unfinished" regions touching the tile boundaries are put onto a stack for further consideration (e.g., regions C2 and C3 in Figure 8).

The final stage "stitches" unfinished objects straddling the W and S tile boundaries together as far as possible for this tile. Each set of n unfinished touching regions in the current tile and previous tiles to the W and S is merged to a single region (e.g., C2 and A1, and C3 and B1, in Figure 8). If the resulting region is now complete (i.e., if it does not continue into unprocessed tiles to the N or E), it is put into the vector tile containing the center of its frame.

Database Interfaces. The facilities of MuSID may be summarized by considering its interfaces with the rest of the system. These consist mainly of C functions for manipulating vector, raster, and associated nonspatial data (Figure 2). A number of these routines may also be called by the MuSIP supervisor module. The functions include

File connection routines for image file I/O initialization and termination.

Raster DBMS functions for reading/writing intervals of raster data.

Vector PUD functions for read/write/edit/delete of region/line/point data. These also include higher level functions for merging adjacent regions and splitting regions.

Functions for operating on image sets for data fusion.

Image headerfile manipulation functions for read/write of header keywords.

Functions for user operations on images, including image and map input from tape, image output, image deletion, subimage extraction and printing, and vector database interrogation.

Display utilities for image display, zoom, and contrast stretch, including overlay of raster and vector data.

A number of utility modules, including a data flow control module, raster to vector conversion, and vector to raster conversion.

CONCLUSION

A spatial database satisfying the system requirements of the MuSIP project has been designed. A substantial subset of the database is currently being implemented on a SUN 4 workstation prior to integration with the rest of the system. This will then be evaluated against the diverse applications of change detection in forested areas using remotely sensed imagery and the identification of anatomical structures within

the brain by use of NMR and PET data.

ACKNOWLEDGMENT

We thank the other members of the Consortium for their assistance, namely, G. Sawyer and I. Jones-Parry [Marconi Space Systems (prime contractor)], D. Johnson (Marconi Research Centre), N. Hindley and J. Fullwood (Marconi Command and Control Systems), T. Pike and A. de Salabert (Messerschmitt-Bolkow-Blohm), J.-Y. Catros, T. Plassard, and I. Pottier (Thompson CSF), M. Barr and A. Wielogorski (Huntings Technical Services), and S. Dellapiane, S. Serpico, and G. Vernazza (University of Genoa). The NUTIS work was partly funded under NERC Contract F60/G6/12.

REFERENCES AND NOTES

(1) Tamura, H.; Yokoya, N. Image database systems: a survey. Pattern Recognit. **1984**, *17* (1), 29-43.
(2) Chang, S. K. Image information systems. *Proc. IEEE* **1985**, *73* (4), 754-64.
(3) Nagy, G. Image database. *Image Vision Comput.* **1985**, 13 (3), 111-6.
(4) Levine, M.; Shaheen, S. A. A modular computer vision system for picture segmentation and interpretation. *IEEE Trans. Pattern Anal. Mach. Intell.* **1981**, *3* (5).
(5) McKeown, D. M. The role of artificial intelligence to the integration of remotely sensed data with geographic information systems. *IEEE Trans. Geosci. Remote Sensing* **1987**, *25* (3), 330-48.
(6) Wu, I. K.; Cheng, D. S.; Wang, W. T. Model-based remotely-sensed image interpretation. *Int. J. Remote Sensing* **1988**, *9* (8), 1347-56.
(7) Nicolin, B.; Gabler, R. A knowledge-based system for the analysis of aerial images. *IEEE Trans. Geosci. Remote Sensing* **1987**, *25* (3), 317-29.
(8) Toriwaki, J.; Hasegawa, J.; Fudumura, T.; Takagi, Y. Pictorial information retrieval of chest X-ray image database using pattern recognition techniques. *Proc. Medinfo'80* **1980**; pp 1116-9.
(9) Corr, D. G.; Tailor, A. M.; Cross, A.; Hogg, D. C.; Lawrence, D. H.; Mason, D. C.; Petrou, M. Progress in automatic analysis of multitemporal remotely-sensed data. *Int. J. Remote Sensing*, **1989**, *10* (7), 1175-95.
(10) McKeown, D. M. MAPS: the organization of a spatial database system using imagery, terrain and map data. Carnegie-Mellon University: Pittsburgh, PA, 1983; Rep. CMU-CS-83-136.
(11) Rye, A. J.; Oddy, C. J.; Johnson, D. G.; Bishop, M.; Jones-Parry, I.; de Salabert, A.; Mason, D. C.; Bell, S. B. M.; Wielogorski, A.; Catros, J.-Y.; Plassard, T.; Serpico, S.; Hindley, N. MuSIP—Multisensor image processing. *Proceedings of the ESPRIT 1990 Conference*, 1990; in press.
(12) Stonebraker, M.; Rowe, L. A. The design of POSTGRES. *Proc. ACM SIGMOD* **1986**, 340-4.
(13) Herring, J. R. TIGRIS: Topologically Integrated Geographic Information System. *Auto Carto 8 Proceedings*, Baltimore, MD 1987; pp 282-91.
(14) Oddy, C. J. Picture Understanding Database—PUD system specification. Marconi Research Centre Report PALS/WN/108; 1988.
(15) Cruse, D.; Oddy, C. J.; Wright, A. A segmented image data base (SID) for image analysis. *Proc. IEEE 7th Int Conf. Pattern Recognit.* July 30-August 2, Montreal, 1984; pp 493-6.
(16) Haralick, R. M.; Minden, G. KANDIDATS: an interactive image processing system. *Comput. Graphics Image Process.* **1978**, *8*, 1-15.
(17) Rosenfeld, A.; Kak, A. C. *Digital picture processing.* Academic Press: New York, 1982.
(18) Merrill, R. D. Representations of contours and regions for efficient computer search. *Commun. ACM* **1973**, *16* (2), 69-82.
(19) Piper, J.; Rutovitz, D. Data structures for image processing in a C language and Unix environment. *Pattern Recognit. Lett.* **1985**, *3*, 119-29.
(20) Haralick, R. M. A spatial data structure for geographic information systems. In *Map Data Processings*; Eds.; Freeman, H., Pieroni, G. G., Academic Press: New York, 1980.
(21) Shapiro, I. G. Design of a spatial information system. *Ibid.*
(22) Shapiro, I. G.; Haralick, R. M. A spatial data structure. *Geoprocessing* **1980**, *1*, 313-97.
(23) Burrough, P. A. Principles of Geographical Information Systems for Land Resources Assessment. Clarendon Press: Oxford, 1986.
(24) Oddy, C. J.; Rye, A. J.; Tavendale, R. D. Software system design for general purpose image analysis. *GEC J. Res.* **1983**, *1* (1), 48-58.
(25) Petkovic, D.; Mohiuddin, K. Combining component features from multiple image frames. *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Database Management* **1985**, 169-74.

Technical Data Interchange Using Tabular Formats

PHILIP M. SARGENT
Engineering Department, Cambridge University, Cambridge CB2 1PZ, England

Received January 3, 1991

Formats and protocols used to exchange data between materials property databases are an enabling technology necessary for integration of materials information with computer-aided engineering. However, it is also necessary to ensure that the format is *at least as capable* of expressing "associativities" between data as are many of the participating databses. This paper shows that an extension to a "flat-file" or "tabular" data format can cope with arbitrary complexity in the associativity between the data items in a dataset (i.e., between the numeric data and the "metadata" describing the experimental conditions). The extension is simply to use several distinct tables to jointly hold the data: a *relational* tabular interchange format. Materials data, because of their heterarchically organized definitions and open-ended nature, present particularly difficult problems for data interchange. A solution to some of the problems is presented here, and it is expected that it may also be of use in other technical data transfer applications.

INTRODUCTION

A well-known commercial database format (dBase, by Ashton-Tate Inc.) is already being used by many materials properties database managers to upload test data into their databases. The main disadvantage of this format is that it is defined in binary. A plain-text version of this format is presented in this paper and maintains two-way "intertranslatability" in that no information is lost in conversion in either direction. It is called the Cambridge Tabular Data Interchange Format (CTDIF).

The aims of this format are to be

1. The simplest useful format
2. A flat-file tablelike representation
3. Convertible to and from dBase III *.dbf files
4. Plain-text to aid editing and word processing
5. Possible to include as an external reference in an SGML document
6. Compatible with being made conformant to MIL-STD-1840A
7. Designed for machine readability, not just a way to represent tables of data for presentation to people
8. Extensible to arbitrary complexity

In the past, the *associativity* and *meanings* of names and terms in interchange formats have been confused (a glossary of terms used in this paper is found in Table I). By separating these two aspects, this paper attempts to show that associativity

0095-2338/91/1631-0297$02.50/0