Cross-Dimensional Sensor Data Management[1,2]

Peter Baumann, Andreas Dehmel, Paula Furtado, Roland Ritsch, Norbert Widmann
FORWISS (Bavarian Research Center for Knowledge Based Systems)
Orleansstr. 34, D-81667 Munich, Germany
phone: +49-89-48095-200, fax +49-89-48095-203
Email: {baumann,dehmel,furtado,ritsch,widmann}@forwiss.de

Abstract

Spatio-temporal data ranging from 1-D time series to 4-D measurements and simulations have become of central importance in terms of both data volumes generated and the need for ubiquitous high-level information in a variety of applications. Management of the large variety of data and their combination, however, poses high challenges wrt. structural variety of multi-sensor data, evaluation complexity, and the sheer data volumes on hand.

The multidimensional database management system (DBMS) RasDaMan has been developed to provide the same quality of service for the multidimensional raster data category as is available on alphanumeric data since long. Based on a strict separation of external retrieval facilities and internal data management, the client/server system accomplishes storage, retrieval, and manipulation of raster data of any size and dimension. An SQL-based query language allows to phrase queries up to the complexity of the Discrete Fourier Transform.

Internally, retrieval is supported through extensive query and storage optimization. The streamlined query engine optimizes disk access and networks transfer times and handles data format and compression issues in a transparent manner.

As RasDaMan is unlimited in terms of dimensions, sensor data can be immediately coupled, e.g., with a data warehouse residing in the same physical database. The DBMS is fully operational and being marketed.

In this contribution, we present RasDaMan in the context of remote sensing and show how multi-sensor queries can be formulated and processed efficiently.

1 Introduction

Data capture in airborne remote sensing results in a variety of raster data ranging from simple photographic 2-D images of the earth surface to complex multidimensional, multi-band data generated by radar instruments. The recorded data has common key properties: Data items can be modeled as a multidimensional array over an arbitrary base type and they tend to be large, both in terms of individual size and in terms of quantities arising. The term *multidimensional discrete data* (MDD) has been coined for this information category. In recent years more and more remote sensing data is continuously generated by an ever increasing number of sensors and platforms. Technological advances in storage technology make it feasible to persistently store huge amounts of data.

---

[1] Presented at the Fourth International Airborne Remote Sensing Conference and Exhibition/21st Canadian Symposium on Remote Sensing, Ottawa, Ontario, Canada, 21-24 June 1999.
[2] Research in part sponsored by the European Community under Esprit research grant no. 20073.

The software for managing this large amounts of multidimensional data, however, has been lagging behind. It would be desirable to have easily accessible databases storing this valuable data online. Commonly storage is done in files on a large number of tapes involving a lot of time and effort to retrieve the data needed for processing and analysis. The challenge for database technology is to offer versatile, high-level MDD services to enable efficient access to the sensor information. Remote sensing data is particularly challenging for MDD management in several respects. A wide variety of data has to be processed jointly in many cases; e.g., projecting a multi-band satellite image on top of a 3-D representation of a DEM. This requires MDD of different dimensionalities to be handled.

Current database technology is far from being able to meet this challenge. State-of-the-art relational database management systems (DBMSs) can only store MDD as unstructured BLOBs (binary large objects) without any support for high-level operations such as subselects, access to bands, or operations on MDD. New DBMS technologies like object-relational DBMSs (e.g. Informix Universal Server [Ston-95]) and object DBMSs [Loom-95] (e.g. O_2 [Banc-92]) offer the possibility to implement user defined data types managing remote sensing data. Both technologies cannot offer a high-level query language and specialized storage structures for MDD.

Image processing systems, such as EASI/PACE or ERDAS IMAGINE, are widely used in remote sensing. These systems, however, lack the storage capabilities provide by database technology. Their advanced image processing algorithms exceed functionality offered by an MDD database system. A close coupling between an image processing system and a specialized DBMS could offer the benefits of both technologies.

In the RasDaMan project, a specialized raster DBMS was developed offering the same quality of service on MDD as it is available on conventional alphanumeric data. On the logical level, a MDD model is offered which allows structure definition of multidimensional arrays with fixed or variable bounds as well as retrieval and manipulation through both an ODMG-conformant [Catt-96] C++ API and a declarative query language enhancing standard SQL [ISO-92]. On the physical level, a specialized MDD storage manager is employed together with algebraic MDD query optimization. One goal of the RasDaMan project was, by providing high-level raster data management services, to make application development faster, less error-prone, and easier to maintain; as a consequence of the resulting lightweight applications, less expensive client hardware can be employed. In this contribution, we demonstrate how a raster DBMS like RasDaMan can be applied in remote sensing applications. In Section 2, the RasDaMan DBMS interface is presented using example queries on remote sensing data. Storage of MDD is discussed in Section 3. Finally, Section 4 gives a conclusion.

2 Cross-dimensional Raster Data Manipulation

On the conceptual (user) level, MDD definition, manipulation, and retrieval is done in a declarative style through a data definition language, RasDL, and a query interface, RasQL. A C++ API, RasLib, accomplishes access to the database from within a programming language, including classes and methods for procedural data definition (the RasDL processor actually is a client using RasLib) and RasQL query invocation. In this Section, we introduce central concepts of these two interfaces. Window dumps shown have been produced using the RasDaMan visual front-end rview (another RasLib client) which allows interactive database querying and offers various visualization techniques.

2.1 MDD Definition

Arrays can be built upon any valid C/C++ type, be it atomic or composed ("struct"). Syntax is based on ODMG's definition language, with arrays defined through a template `marray<b,d>` which is instantiated

with the array base type `b` and the array extent `d` (in RasDaMan called the array's *spatial domain*). Spatial domain specification - the only syntactic extension to ODMG's ODL - contains lower and upper array bound for each dimension. Thus, a database collection `LandsatCollection` over Landsat images can be defined by

```
typedef struct { unsigned short band1, ..., band7; } LandsatPixel;
typedef marray< LandsatPixel, [ 1:5760, 1:7020 ] > LandsatImage;
typedef set< LandsatImage > LandsatCollection;
```

An asterisk "*" in any position denotes a free boundary which disables range checking for that particular bound during object lifetime. Thus, a set of 4-D climate simulations, each one consisting of a time series of 16x32x64 3-D atmospheric values over an unspecified time range, can be described by

```
typedef marray< float, [ 1:*, 1:16, 1:32, 1:64 ] > ClimateSimulation;
typedef set< ClimateSimulation > ClimateCollection;
```

The RasDL processor takes such MDD definitions and generates both the data dictionary information and a C++ header file for proper application interfacing. Once the database structure is defined this way, the database can be populated and searched.

2.2 MDD Search and Manipulation

The RasQL query language is based on standard SQL92. Like SQL, a query returns a set of items (in this case multidimensional units) taken from some database collection. For choosing elements from the collection as well as for tailoring each MDD element from the result set, multidimensional operators are available. We will look at the geometric operators (which perform a cutout from an MDD), induced operations (which simultaneously change all the values of an MDD), and aggregation (which deduces summary information). Furthermore, data format conversion handling will be shown.

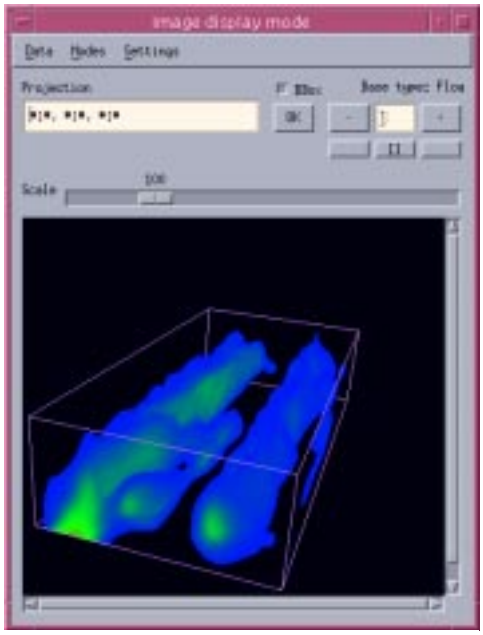

Fig. 1: 3-D section through 4-D climate simulation; data courtesy of DKRZ (German Climate Computing Centre)

2.2.1 Operations on Cell Subsets

This includes trimming (rectangular cutouts) and section (extraction of lower-dimensional subarrays). It is accomplished by attaching a domain specification to an MDD which, for every dimension, states the interval or section point, resp.

Example 1: The following query retrieves a 2000x3000 cutout from all Landsat images in `LandsatCollection`:

```
SELECT c[ 1001:3000, 1001:4000 ]
FROM   LandsatCollection AS c
```

A section extracts an (n-1)-D layer from an n-D MDD. To this end, the spatial domain specification contains for each dimension the section *position* rather than a trim *interval*. Trim and section can be combined freely within one expression; obviously, each section indicator reduces the result dimension by one.

Example 2: Let us assume that `ClimateCollection` contains one 4-D cube. The following query, then, extracts a 3-D volume at time frame 1020 from this cube:

```
SELECT c[ 1020, *:*, *:*, *:* ]
FROM   ClimateCollection AS c
```

Notably, due to server-based query processing only the result set - 128 kB in this case - will be transferred to the client. As a further example for a cross-dimensional query capabilities, a height profile is extracted from a 2-D DEM (see Fig. 2 below).

Example 3: Select a height profile at x-coordinate 400 from the DEM in the collection rocckies:

```
SELECT d[ 400, *:* ]
FROM   DEMCollection AS d
```

2.2.2 Induced Operations

For each operation available on the MDD cell type, a corresponding so-called induced operation is provided which simultaneously applies the base operation to all cells of an MDD. Both unary (e.g., record access or contrast enhancement) and binary operations (e.g., masking an image) can be induced.

Example 4: The query "*Band 3 of all Landsat TM images, with intensity reduced by 10*" is written as

```
SELECT c.band3 - 10
FROM   LandsatCollection AS c
```

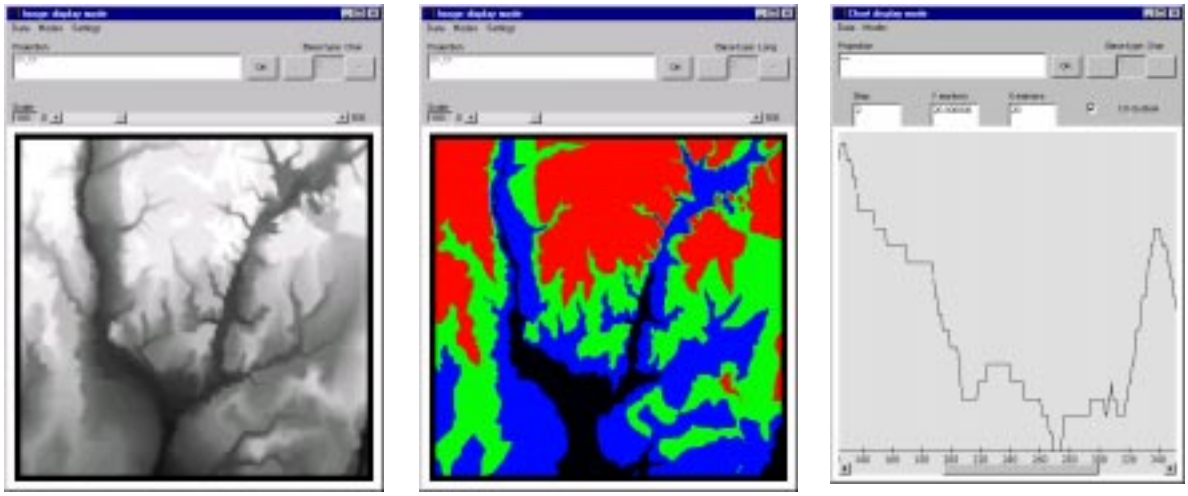

Figure 2: Classification of a 2-D DEM (left) into 4 elevation levels (middle) and selection of a 1-D height profile (right).

Example 5: Classification, i.e. mapping of cell values to other values, is a common operation when dealing with remote sensing data, but also in other areas, such as business data analysis. Doing this in the DBMS on a potentially large set of images greatly eases the programmer's task. Our example (Fig. 2) classifies 8-bit DEM data of the Rocky Mountains into four distinct height levels ranging from 0 to 3.

```
SELECT (d >= 64  AND d < 128)*1 + (d >= 128 AND d < 192)*2 +
       (d >= 192 AND d < 256)*3
FROM   DEMCollection AS d
```

Example 6: Fig. 3 shows the three visible bands of a satellite view on the South Spanish coast. The query below masks out all pixels from this image which are considered non-vegetation due to their color value.

```
SELECT c * (c.green > 130 AND c.red < 110 AND c.blue < 140)
FROM   LandsatCollection AS c
```

Obviously, more complex algorithms have to be used in a real-live applications. However, even a simple algorithm such as the one above can be useful to preselect relevant images, thereby significantly reducing network traffic and client load.

In general, MDD expressions can be used in the `SELECT` part of a query and, if the outermost expression result type is Boolean, also in the `WHERE` part. Therefore, we need a means to ”collapse” MDD-valued expressions, such as induced comparisons, into scalar Boolean values. This is accomplished through condensers.

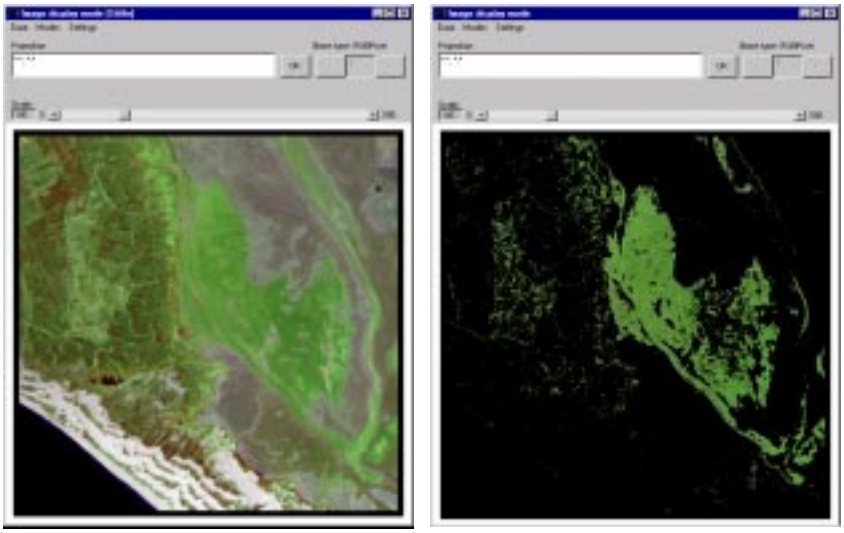

Figure 3: Searching for vegetation (right) in a visible band image (left); image courtesy of Spanish National Geographic Institute.

2.2.3 Deriving Summary Data

Condenser operations - in standard SQL called aggregation - allow to summarize over all values from some spatial domain. A number of condensers are available, such as `sum_cells`, `count_cells`, `max_cells`, `min_cells`, and `avg_cells` over some cell area, as well as `some_cells` and `all_cells` quantifiers.

Example 7: The following expression counts all cell occurrences containing values between 17 and 42 (semithresholding).

```
count_cells( a > 17 AND a < 42 )
```

Example 8: The query "*Identifiers of all Landsat images which, according to the above criterion, contain more than 70% vegetation.*" can be formulated as

```
SELECT oid( c )
FROM   LandsatCollection AS c
WHERE  count_cells( c.green > 130 AND c.red < 110 AND c.blue < 140 )
       / count_cells( c )
       > 0.7
```

To process such queries efficiently, two system features have received particular attention. First, internal optimizations take care that, regardless of the actual query complexity, each cell will be touched at most once. Second, only the object identifiers (OIDs) are actually moved to the client, leaving the bulk load on the server, i.e., close to the data source.

2.2.4 Partial Updates

Because of the large size of the data items, we feel that, despite ODMG OQL includes only retrieval, support for partial updates is useful, too. To this end, the SQL update statement has been included in RasQL with an extension to specify an MDD part on the left hand side of an assignment clause.

Example 9: Inserting half of band5 sensor data into an existing Landsat image can be expressed as

```
UPDATE LandsatCollection
SET    img[ *:*, 1:3510 ].band5 ASSIGN <spectral band sensor data>
WHERE  oid(img) = ...
```

2.2.5 Data independence.

Data independence frees the application from the need to cope with different image formats. By default, RasDaMan delivers MDD in the client machine's hardware representation for arrays, ready for further processing. On request, MDD are accepted or delivered using a particular data exchange format indicated in the query. Of course, the format must be able to host the structure to be encoded – a seven-band multispectral image cannot be cast into GIF, whereas a single band can.

Example 10: The application reads a Landsat image in BIL format, say, from tape and provides it to the query through formal parameter `$1`, which is replaced by a user-specified filename in the frontend. Function `inv_bil()` converts this into the database internal representation prior to storing it:

```
INSERT INTO LandsatCollection
VALUES ASSIGN inv_bil( $1 )
```

Conversely, a single band of the Landsat image stored earlier is delivered in TGA format by applying function `tga()`:

```
SELECT tga( c.band3 )
FROM   LandsatCollection AS c
```

Besides the operators shown there is a set of general operations based on a formal algebraic framework [Bau-99] which allows to perform signal processing up to the Discrete Fourier Transform.

2.3 Data Access through Programming Language

Application programmers use the ODMG-conformant C++ library RasLib to store, manipulate, and retrieve persistent MDD in the RasDaMan DBMS. For convenience, some MDD functionality is also available in RasLib to perform operations such as trimming and section of arrays on client side. A template class for MDD objects has been implemented to integrate this capability in the C++ type system. These MDD objects support the classical C++ way of accessing arrays by overloading `operator[]`. Objects of such a class can be made persistent by inserting them into persistent DBMS collections.

Recall that MDD objects always are delivered in the client's main memory format (except when a conversion function such as `tiff()` has been applied). A method allows to retrieve a pointer to the C++ array, thereby giving access to MDD with C++ functions (albeit this sacrifices the multidimensional semantics as the low-level C++ array concept has to be used[3]). This way, tight integration of RasDaMan with C++ class libraries, e.g., imaging packages, is possible. Here is a brief code example for Landsat image traversal following its retrieval:

---

[3] "*The built-in arrays* [of C/C++] *are a major source of errors – especially when they are used to build multidimensional arrays*." [Str-97]

```
typedef struct { unsigned short band1, ..., band7; } cell_type;
r_Marray<cell_type> mdd;
cell_type* p;
unsigned short maxVal = 0;
...
p = mdd.get_array();
for (i = 0; i++; i < 5760*7020) {
    maxVal = max(maxVal, (p++)->band3);
}
```

3 Storage Model

Queries are evaluated close to the bulk data source, i.e., on the server [Furt-97]. Storage and access optimization also take place there. The underlying storage model is based on multidimensional tiling. Maintenance of large images by subdividing them into tiles (also called mosaics) as the unit of access has a long tradition in imaging. In the database community subdivision of images into equally sized tiles was suggested in [Tamu-80]. RasDaMan has generalized this concept to the n-dimensional case and arbitrary tiling (cf. Fig. 4), supporting various strategies such as user-defined hot spots with system-generated completion of tiling and tiling based on access frequencies [Furt-99]. Tiles also form the units for lossless transparent data compression. To quickly determine the tiles affected by a particular query, a specialized multidimensional index is maintained.

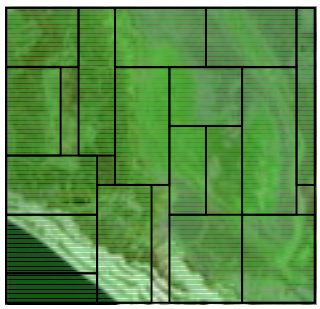

Fig. 4: arbitrary 2-D tiling

RasDaMan makes use of a conventional DBMS to actually store arrays and the associated meta information. As the array semantics is resolved to a simple "set of blob" semantics, only very simple services are required from the base DBMS. The peculiarities of the base DBMS are encapsulated in a driver interface. This prepares RasDaMan for easy portability between different base DBMSs and storage systems. As a matter of fact, RasDaMan first has been implemented based on the ODBMS O_2, and later has been ported to relational systems such as Oracle.

4 Conclusion

With the ever-increasing need to maintain growing sets of sensor data in their originally acquired form, geo information providers are faced with severe information management problems. Furthermore, besides 2-D imagery of varying resolution and sensor types, additional information sources have to be regarded delivering 1-D time data as well as higher-dimensional entities, such as 4-D climate data. As these all share common underlying properties, namely that of arrays in a programming language sense, it is highly feasible to maintain them in a DBMS which supports this unifying structuring paradigm.

Responding to this research issue, RasDaMan has been designed to offer versatile retrieval functionality on raster data of any size and dimension. By extending standard SQL with operations on multidimensional entities, queries can be formulated flexibly and *ad hoc*. We feel that a particular advantage is that the same mechanisms apply for all dimensions and that various dimensions can be handled within one and the same query; frequently queries can even be phrased without explicitly mentioning the domain, thereby allowing to set up general query libraries. Storage and access optimization are done transparently to the user, but under control of the database administrator. This makes RasDaMan suitable as a raster management component in advanced GISs.

Meanwhile, RasDaMan has become a commercial product[4]. The server is running on common Unix platforms like Solaris, HP-UX or Linux. The client API and the visual front-end are also available on Windows NT. In parallel, research on RasDaMan continues; on the agenda being issues such as parallelization and optimized tertiary storage management.

References

[Banc-92] F. Bancilhon, C. Delobel, P. Kanellakis: *Building an Object-Oriented Database System*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.

[Bau-99] P. Baumann: *A Database Array Algebra for Spatio-Temporal Data and Beyond*. Proc. NGITS'99, Zikhron-Yaakov, Israel, July 5-8, 1999 (accepted for publication)

[Catt-96] R. Cattell: *The Object Database Standard: ODMG-93*. Morgan Kaufmann Publishers, 1996.

[Furt-97] P. Furtado, R. Ritsch, N. Widmann, P. Zoller, P. Baumann: *Object-Oriented Design of a Database Engine for Multidimensional Discrete Data*. Proc. of the OOIS '97 Conference, Brisbane, Australia, November 1997.

[Furt-99] P. Furtado, P. Baumann: *Storage of Multidimensional Arrays based on Arbitrary Tiling.* Proc. 15th Int. Conf. on Data Engineering (ICDE), March 23-26, Sydney, Australia, March 1999.

[ISO-92] The International Organization for Standardization (ISO): *Database Language SQL*. ISO 9075, 1992(E), 1992.

[Loom-95] M. Loomis: *Object Databases: The Essentials*. Addison-Wesley, 1995.

[Ston-95] M. Stonebreaker, D. Moore: *Object-Relational DBMSs: The Next Great Wave*. Morgan Kaufmann Publishers; 1995.

[Str-97] B. Stroustrup: *The C++ Programming Language*. Third Edition, Addison Wesley, Reading-Massachusetts, 1997.

[Tamu-80] H. Tamura: *Image Database Management for Pattern Information Processing Studies.* In: S. Chang, K. Fu (ed.): *Pictorial Information Systems*. LNCS 80, pp. 198-227, Springer 1980.

[4] see www.active-knowledge.de