IMAGE DATA HANDLING IN SPATIAL DATABASES

LÚBIA VINHAS, RICARDO CARTAXO MODESTO DE SOUZA, GILBERTO CÂMARA[1]

[1] Instituto Nacional de Pesquisas Espaciais - INPE

Av. dos Astronautas, 1758, São José dos Campos (SP), Brazil 12227-001

{lubia, cartaxo, gilberto}@dpi.inpe.br

ABSTRACT

The recent advances in database technology have enabled the development of a new generation of spatial databases, where the DBMS is able to manage spatial and non-spatial data types together. Most spatial databases can deal with vector geometries (e.g., polygons, lines and points), but have limited facilities for handling image data. However, the widespread availability of high-resolution remote sensing images has improved considerably the application of images to environmental monitoring and urban management. Therefore, it is increasingly important to build databases capable of dealing with images together with other spatial and non-spatial data types. With this motivation, this paper describes a solution for efficient handling of large image data sets in a standard object-relational database management system. By means of adequate indexing, compression and retrieval techniques, satisfactory performances can be achieved using a standard DBMS, even for very large satellite images. This work is part of the development of the TerraLib library, which aims to provide a comprehensive environment for the development of GIS applications.

1 Introduction

The recent technological advances in database technology are providing the support for major advances in non-conventional database applications. In the area of geographical information systems (GIS), database technology has enabled the complete integration of spatial data types in object-relational data base management systems, creating a new generation of spatial databases (Shekhar, Chawla et al. 1999). This integration is bound to change completely the development of GIS technology, enabling a transition from the monolithic systems of today (that contain hundreds of functions) to a generation of spatial information appliances, small systems tailored to specific user needs (Egenhofer 1999). Three major challenges in spatial database construction are: (a) the efficient handling of spatial data types, which include both vector (i.e., polygons, lines and points) and raster data structures; (b) the availability of tools for query and manipulation of spatial data; (c) the support for advanced applications, such as mobile GIS, spatio-temporal data models (Hornsby and Egenhofer 2000), geographical ontologies (Fonseca, Egenhofer et al. 2002) and dynamic modelling and cellular automata (Couclelis 1997).

One area of special interest is the efficient handling of raster data, especially satellite images. Remotely sensed imagery is one of the most pervasive sources of spatial data currently available to researchers who are interested in large-scale geographic phenomena. The variety of spatial and spectral resolutions for remote sensing images is large, ranging from IKONOS 1-meter panchromatic images to the polarimetric radar images soon to be part of the next generation of RADARSAT and JERS satellites. Recent advances in remote sensing technology, with the deployment of a new generation of sensors, have improved considerably such application areas as environmental monitoring and urban management.

The construction of spatial databases that handle raster data types has been studied in the database literature and the main approach taken has been to develop specialized data servers, as in the case of PARADISE (Patel, Yu et al. 1997) and RASDAMAN (Reiner, Hahn et al. 2002). The chief advantage of this approach is the capacity of performance improvements, especially in the case of large image databases. The main drawback of this approach is the need for a specialized, non-standard server, which would greatly increase the management needs for most GIS applications. Therefore, the approach taken by the authors was to include raster data management into object-

relational database management systems. We consider that, by means of adequate indexing, compression and retrieval techniques, satisfactory performances can be achieved using a standard DBMS, even for very large satellite images. This work is part of the development of the **TerraLib** library, which aims to provide a comprehensive environment for the development of GIS applications (Câmara, Souza et al. 2000).

With this motivation, this work describes the conception and implementation of the image data handling facilities in **TerraLib**. These facilities include: (a) efficient storage and indexing; (b) decoders for the different image data formats; (c) basic data manipulation functions; (d) iterators to provide a convenient way of accessing the image data and developing image processing algorithms. These facilities are supported in different DBMS, including ORACLE, PostgreSQL, mySQL and Access database management systems therefore allowing easy interface with existing user environments.

2 The Raster Data Type in TerraLib

Images in TerraLib are handled by a generic raster data type and are usually associated with a cartographical projection and may be multi-dimensional, e.g., the different spectral bands of a remote sensing image. Images are usually associated with different file formats, such as TIFF and JPEG, and may refer to different representations: (a) a synthetic image that has an associated look-up table, where the pixel value is associated with a RGB triple; (b) a multi-band image; (c) an image which is associated with a categorical coverage; (d) an image where each pixel has an ID of a unique spatial object. We have defined three basic classes in TerraLib:

- A generic raster data structure (`TeRaster`).
- A class that stores the metadata about a raster data set (`TeRasterParams`), including: image type, number of lines, columns, and dimensions, resolution, bounding box, cartographical projection, rotation and skew parameters, and compression technique used.
- A class that handles different data formats, including storage in a DBMS (`TeDecoder`).

The two basic functions available to the `TeRaster` class are `setElement` and `getElement`, which store and retrieve an individual raster value. In order to decouple this class from the different storage alternatives, requests to the `TeRaster` class are handled by the `TeDecoder` class, as shown in. This is an example of the "Strategy" design pattern (Gamma, Helm et al. 1995). The `TeDecoder` class is specialized into derived classes that handle specific data formats, such as JPEG files, TIFF files or images stored in a DBMS. The Figure 2.1 shows the relationship between the `TeRaster`, `TeRasterParams` and `TeDecoder` classes.

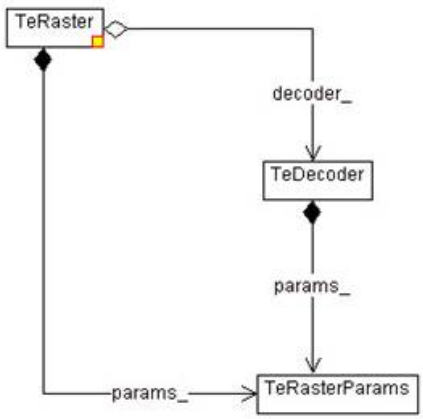

Figure 2.1 Relationship between classes that handle imagery in TerraLib

3 Spatial Database Interface to Raster Data

A TerraLib database is a set of "geographical layers", where each layer consists of a specific type of geographical data (such a soil map, an image or a cadastral map). For object-relational DBMS data storage, each layer is associated with a set of tables, which store its spatial and non-spatial components. There will be a different table associated with each layer for storing each of the different geometries (point, line, polygon and raster). In TerraLib, access to spatial data stored in a DBMS is achieved by a generic Application Programming Interface (API), which encapsulates the internal differences of each database system. This API maps TerraLib's spatial data types into the characteristics of each specific DBMS, using spatial indexing or in-built optimization facilities, if available. Currently, TerraLib supports the Oracle, mySQL, PostgreSQL and Access systems. Two abstract classes have been defined for this API: `TeDatabase` and `TeDatabasePortal`,

which allow: (a) establishment of the connection with the database server; (b) executing SQL commands; (c) defining indexes; (d) defining referential integrity; (e) executing spatial or non-spatial queries that return a set of records (Ferreira, Queiroz et al. 2002).

In order to store the raster data in a DBMS, previous works in the literature (Patel, Yu et al. 1997; Reiner, Hahn et al. 2002) have shown than a combination of *multi-resolution pyramid* and a *tiling* scheme is the most appropriate strategy for handling large image files. The tiling scheme is used as a spatial index, such that when retrieving a section of an image, only the relevant tiles will be retrieved and decompressed. The multi-resolution pyramid is very useful for visualization of large data sets, to avoid unnecessary data access. This approach was adopted in TerraLib. Each image is associated with a database table, called the "raster table", where each record stores a tile (or block) of the image. The fields of the raster table are shown in Table 1.

For image storage in a TerraLib database, each band of the image is decomposed in a set of disjoint tiles with width W and height H given in number of pixels. Each tile is stored in the field spatial data (as a long binary) of a record in a raster table. Tiles can be compressed before storage. Global information about a raster layer, such as tile dimension, compression technique, number of rows and columns, number of bands, bounding box, and pixel resolution is stored in a complementary table.

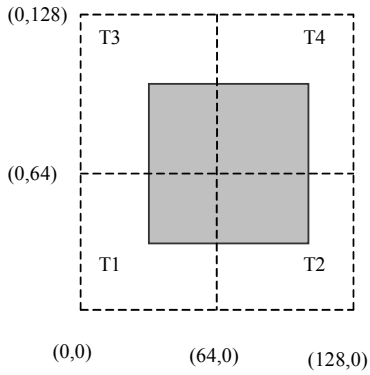

Figure 3.1 Decomposition and identification of tiles.

TABLE 1 – THE RASTER TABLE IN TERRALIB

block_id	string	Tile unique identifier
lower_x	real	Tile bounding box minimum X coordinate
lower_y	real	Tile bounding box minimum Y coordinate
upper_x	real	Tile bounding box maximum X coordinate
upper_y	real	Tile bounding box maximum X coordinate
band_id	int	Band
resolution_factor	int	Tile resolution factor
subband	int	Subband information
spatial_data	binary	Raster tile
block_size	int	Size in bytes of a tile (after compression)

The bands of the image are divided in blocks of dimension 2^n x 2^m (the default is 512 x 512) to simplify the construction of a multi-resolution structure. The starting point for the tiling division must correspond to a known geographical reference. This allows a definition of a function that returns the same block identification for all the pixels that belong to a given tile, and is used as primary key of the raster table.

For example, consider a 100 x100 image with a pixel resolution of 1 x 1 meter. The lower-left pixel is associated with the geographical coordinate (20, 20) and the upper right pixel associated with the geographical coordinate (120,120). Geographical coordinates are given in the cartographical projection associated with the image. The decomposition of this image (represented by the gray rectangle) in tiles of 64 x 64 pixels will generate the four tiles T1, T2, T3 and T4 shown in Figure 3.1.

The identification of each tile is built using its position in the subdivision of the space by axes positioned at the coordinates multiples of the block size in the vertical and horizontal dimension. So the tile T1 will have the identification X0Y0 projection coordinates, T2 will have the identification X1Y0, T3 will have the identification X1Y0 and T4 will have X1Y1. Given a pixel with coordinates *(x,y)* the identification of block to where it belongs is given by Xn_iYn_j where: $n_i = x$ % 64, $n_j = y$ % 64 and % represents the integer division operator.

4 The multi-resolution raster structure

Operations associated with visualization of large image data sets are constrained by the properties of the output devices. For example, consider a 20,000 x 20,000 image being displayed in a 1,000 x 1,000 pixel canvas. The user may start from a general view of the image and may want to zoom to a selected area, and then to scroll the image looking for areas of interest. In each situation, there should be no need for simultaneous retrieval of all image data. Ideally, the amount of data retrieved should be on the order of magnitude of the display capacity of the visualization device. This requirement can be fulfilled if the spatial database is organized as a multi-resolution structure. This multi-resolution pyramid stores, at its lowest level, the tiles associated with the full resolution image. At higher level, the tiles are grouped into lower-resolution images, which are used for efficient image visualization. Given this motivation, **TerraLib** implements a multi-resolution pyramid. Starting with the full resolution of the image the basic tiles are created. Then we build the subsequent levels decreasing the resolution by a factor of two. Each set of four tiles is merged into one tile, which will contain a reduced-resolution image. This process is repeated for all basic tiles and applied to the next superior level, until at the uppermost level a single tile contains a much-reduced version of the image. The combination of multi-resolution and tiling is equivalent to a spatial indexing scheme that considers both the input and output data requirements. When a user defines an area of interest for image visualization, the indexing scheme will use the geographical bounding box of the tiles and the display resolution to retrieve and subsequently process only the tiles that cover a certain area, with the required spatial resolution. A similar process is used in the specialized applications PARADISE (Patel, Yu et al. 1997) and RASDAMAN (Reiner, Hahn et al. 2002).

5 Query Processing and Data Access

There are two ways of retrieving image data in TerraLib: (a) the use of SQL queries directly in the raster table, followed by the decompressing of each tile; (b) a set of two functions (`setElement` and `getElement`), which allow the access to every element of an image and can be used to build algorithms that operate on the individual pixels of image; (c) a set of image iterators, that allow access to the image data in a sequential basis, as described in the next section. In the first case, the application programmer uses the spatial indexing scheme described in the previous section directly.

In the second case (`setElement` and `getElement` functions), the application programmer does not have to know the specific aspects of spatial database storage and retrieval. He will request access to a specific pixel, which will be provided by TerraLib's database decoder. When dealing with individual pixels, it is important to consider efficiency issues. For that purpose, TerraLib implements a block cache mechanism to reduce the amount for fetch operations from the database. As explained in the previous section, the tiling and multi-resolution scheme provides spatial indexing. The unique identifier associated with each tile is used as a primary key for queries and for the cache system. Whenever a pixel with coordinates *(i,j)* of a band *b*, in a resolution level *r* is required for retrieval, if the tile is not in the cache, it is retrieved, decompressed and stored in memory. The maximum number of tiles to be kept in the cache is defined by the application. When a tile that is not in the cache is requested and there is no more space available, the least recently used tile is discarded. In case of any modification of the pixel values, the tile is updated in the database before being discarded.

6 Raster Iterators for Generic Programming with Images

A geographical information system (GIS) should not only provide methods for storage and retrieval in a spatial database, but must include facilities for the development of algorithms that deal with geographic information. A large number of algorithms do not depend on some particular implementation of a data structure but only on a few fundamental semantic properties, such as the ability to get from one element of the data structure to the next, and to compare two elements of the data structure. For example, an algorithm for computing a histogram of a spatial data set does not need to know if the data is organized as a set of points, a set of polygons, or an image. All that is needed is the ability to look into a list of spatial elements, and to obtain the value of each element of the list. Therefore, GIS software development can be much improved by using emerging paradigm of generic programming (Austern 1998). Generic programming is based on the idea that there are fundamental laws that govern the behavior of software components, which are independent of the data structures. Therefore, generic programming aims at

designing interoperable modules based on the separation of algorithms from the data structures.

In order to enable generic programming techniques to be used in connection with images in TerraLib, the class `TeRaster` provides *iterators* that allow the sequential traversal of an image. The simplest *iterator* is defined by considering its starting as the first column of the first line of the image; each forward operation moves to the next column up to the end of the line and than to the next line until the last column of the last line. Iterators are useful because they allow algorithms to be written in terms of requirements over data structure instead of in terms of the data structure itself. We have also implemented iterators that traverse the elements of a certain portion of the image delimited by a polygon.

7 An Application Example

In TerraLib, we have implemented the infra-structure described in this paper for the ORACLE, mySQL, PostgreSQL and Access database management systems. An example of application program that used this infra-structure is shown in Figure 7.1.

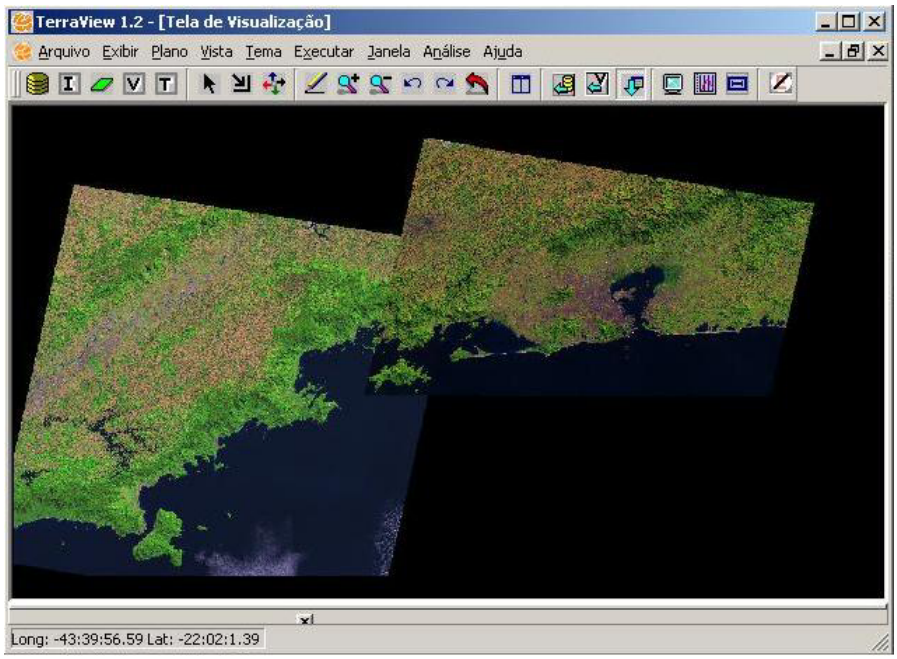

Figure 7.1 An example of application

The raster representation of a layer was built from a mosaic of two images files in GeoTiff format. The first image (left side of the canvas) has 7020 x 7984 pixels and 3 bands, the second image (right side of the canvas) has 7414 x 8239 pixels and also 3 bands, and both images occupy 335Mb as files in GeoTiff format.

The images where divided in blocks of 512 x 512 pixels in a 5 level multi-resolution pyramid. They were stored in an ACCESS system, using the JPEG format at 75% quality level as the compression algorithm. The resulting representation of the raster data in the database is composed of 1500 tiles that after compression occupied 47.2Mb of storage in the database. It should be noticed that the tiling scheme worked properly so that the mosaic was correctly built. In Figure 7.1 the application decompressed only 5 tiles of the uppermost level of resolution.

To execute a zooming operation over approximately a quarter of the canvas (as showing in Figure 7.2) the application decompressed 16 tiles of the third resolution level. Finally to execute a zooming to a detail level (shown in Figure 7.3) the application decompressed 6 tiles of full resolution.

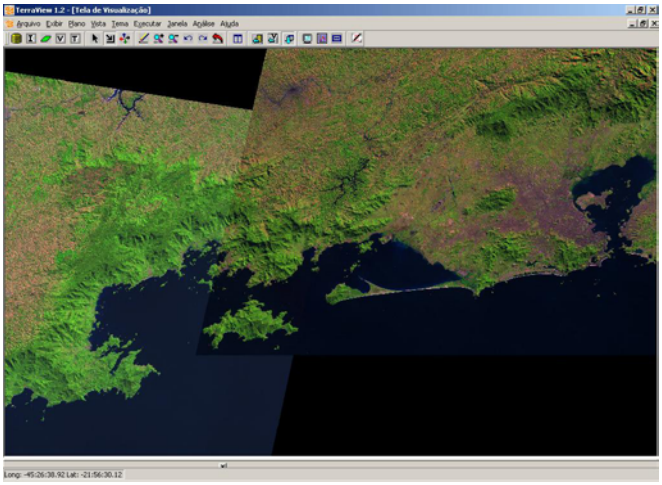

Figure 7.2 A zooming operation.

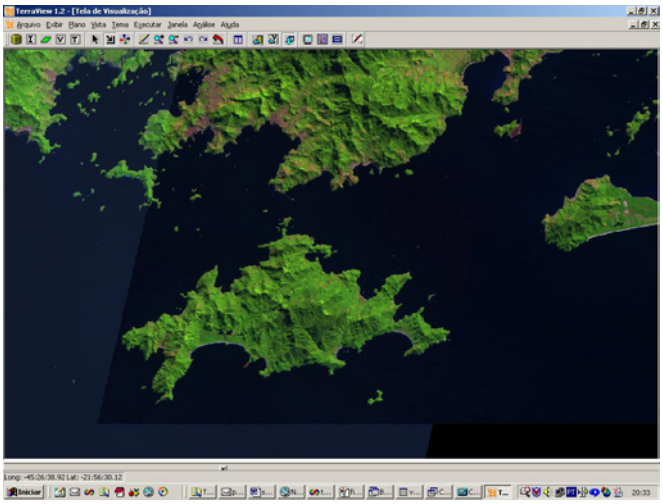

Figure 7.3 A zooming to detail operation.

8 Conclusions

The implementation of an image data-handling scheme in a generic object-relational database system, together with functions to handle other spatial and non-spatial data types, has important advantages over specialized solutions that have been proposed in the literature. We have shown that the object-relational infra-structure can be combined to enable spatial indexing, which allows efficient data retrieval and query processing. By means of adequate indexing, compression and retrieval techniques, satisfactory performances can be achieved using a standard DBMS, even for very large satellite images

We are currently investigating additional aspects of image data handling such as: (a) compromises between tile size and performance; (b) different compression techniques, such as wavelets; (c) different alternatives for building multi-resolution pyramids; (d) new types of raster iterators, which operate over a pixel neighborhood instead of a single element.

Acknowledgements

The TerraLib spatial library is available as open source in www.terralib.org. The authors would like to acknowledge the support of TerraLib teams from INPE (Antonio Miguel Monteiro, João Argemiro Paiva, Karine Reis Ferreira and Gilberto Ribeiro) and TECGRAF (Marcelo Tilio Monteiro de Carvalho and Marco Casanova).

References

Austern, M. (1998). Generic Programming and the STL : Using and Extending the C++ Standard Template Library. Reading, MA, Addison-Wesley.

Câmara, G., R. Souza, et al. (2000). TerraLib: Technology in Support of GIS Innovation. II Workshop Brasileiro de Geoinformática, GeoInfo2000, São Paulo.

Couclelis, H. (1997). "From Cellular Automata to Urban Models: New Principles for Model Development and Implementation." Environment and Planning B: Planning and Design **24**: 165-174.

Egenhofer, M. (1999). Spatial Information Appliances: A Next Generation of Geographic Information Systems. First Brazilian Workshop on GeoInformatics, Campinas, Brazil.

Ferreira, K. R., G. Queiroz, et al. (2002). Arquitetura de Software para Construção de Bancos de Dados Geográficos com SGBD Objeto-Relacionais. XVII Simpósio Brasileiro de Banco de Dados, Gramado, RS.

Fonseca, F., M. Egenhofer, et al. (2002). "Using Ontologies for Integrated Geographic Information Systems." Transactions in GIS **6**(3): 231-257.

Gamma, E., R. Helm, et al. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA, Addison-Wesley.

Hornsby, K. and M. Egenhofer (2000). "Identity-Based Change: A Foundation for Spatio-Temporal Knowledge Representation." International Journal of Geographical Information Science **14**(3): 207-224.

Patel, J., J. Yu, et al. (1997). Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. SIGMOD Conference, Tucson, Arizona.

Reiner, B., K. Hahn, et al. (2002). Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. 3th International Conference on Database and Expert Systems Applications (DEXA), Aix en Provence, France.

Shekhar, S., S. Chawla, et al. (1999). "Spatial Databases: Accomplishments and Research Needs." IEEE Transactions on Knowledge and Data Engineering **11**(1): 45-55.