Postgis Raster Plugin for Quantum GIS

Maurício Carvalho Mathias de Paulo[1,2], **Lubia Vinhas** [2]

[1]Diretoria de Serviço Geográfico – DSG
Quartel General do Exército, Bloco "F", 2o Piso, Ala Norte
CEP:70630-901 – SMU – Brasília – DF, Brasil

[2]Instituto Nacional de Pesquisas Espaciais – INPE
Caixa Postal 515 – 12245-970 – São José dos Campos - SP, Brasil

{mauricio, lubia}@dpi.inpe.br

Abstract. *This article describes a graphic tool implementation to upload and visualize raster data stored in a PostgreSQL database with PostGIS extension. The implementation was done as a Quantum GIS plugin called WktRaster, released as an open source software. The plugin is one of the earliest efforts to explore raster data storage, visualization and processing using the PostGIS Raster extension in a Geographic Information Systems' environment. The main goal is to assist users in exploring the database design flexibility that PostGIS raster type introduces.*

1. Introduction

In the GIS (Geographic Information Systems) community there has been a long debate about how to represent spatial information. Two different approaches coexist, the discrete (object) representation in which spatial data is represented by vector data structures and the continuous (field) representation in which spatial data is represented by raster data structures. Cadastral data (such as a map of census sectors of a city) is an example of vector data and satellite imagery is an example of raster data. In their daily work, GIS users deal with both vector and raster data.

DBMS (Database Management Systems) have evolved to be capable of managing spatial data in a user friendly and efficient way providing spatial abstract data types, indexing mechanisms and functions tailored for spatial data, generically called spatial extensions. One of the most known open source object-relational DBMS is PostgreSQL, that has a spatial extension called PostGIS [PostGIS 2011] . PostGIS' spatial types represent vector data in conformance with OGC (Open Geospatial Consortium) and ISO (International Standards Organization) standards, allowing the storage and manipulation of spatial data using the SQL (Structured Query Language). Even though this is a reality for vector data, there is still a lack of standards for raster data processing and storage in spatial extensions.

Initially PostGIS only had vector data types. A first attempt to support raster data in PostGIS (called PGRaster) was discontinued and in 2010 the Postgis Raster project replaced it's predecessor [PostGIS 2011]. Although PostGIS Raster is an ongoing project, PostGIS version 2.0 is going to include raster support [Obe and Hsu 2011]. This article discusses PostGIS Raster and describes a tool developed to upload raster data to PostGIS as well as to retrieve and visualize the data stored in database, that was released as a plugin for Quantum GIS, a free and open source GIS.

This article starts with an introduction on geographic raster database storage. Section 3 describes the concepts applied in PostGIS Raster. Section 4 describes the Quantum GIS plugin implemented to manage PostGIS Raster layers. Section 5 outlines an example usage of SQL queries to perform processing and visualization of raster data stored using PostGIS' functions.

2. Related work

The main demands that drive implementations are visualization and processing efficiency, interoperability and adequacy to client-server capabilities. The approaches for storing and processing raster data might vary but the partitioning of raster in tiles and maintenance of resolution pyramids are found in most implementations of raster storage.

The partitioning of a raster in tiles, usually of fixed size (for example 128 x 128 cells), allows the indexing of each raster part independently, resulting in efficiency gain when just a raster partition should be processed or visualized. Resolution pyramids are multi-level auxiliary structures that store downsampled (and also tiled) versions of the original raster data. The bottom level contains the original resolution, while higher levels contain the subsequent lower resolution versions. This feature is especially useful for visualization purposes, when applications can retrieve the raster at a level according to a desired zoom level [Vinhas et al. 2003].

One approach to store raster data is using files in a file system. This is the principle of many softwares compliant with the Web Map Tile Service (WMTS) such as *gdal2tiles* [Masó et al. 2010]. In this case, the raster data server is the file system itself. The main advantage is that the data is served the same way that it is stored. One disadvantage is lack of integration with processing functions so the server works only as a repository for raster data.

Some implementations focuses on using the DBMS as a repository for the metadata about the raster, and the data itself is stored as a long binary with no semantics. This approach is similar to the file system's, with the advantage that it maintains vector and raster data stored in the same DBMS. There might be some efficiency loss due to the interaction with the DBMS for data retrieval before being decoded and handled by client applications.

Performance improvement can be achieved by specialized raster DBMS such as PARADISE and RASDAMAN, however, as non-standard servers, they increase the management needs of GIS applications working on top of them [Imran 2009]. An intermediate approach is to construct tiling and multi-pyramids using BLOB (Binary Large Object) type, available on most object-relational DBMS, and adequate indexing and compression mechanisms for efficient data retrieval. The TerraLib library follows this approach providing inside the library data upload and retrieval methods using SQL [Vinhas et al. 2003].

The GeoRaster feature of Oracle Spatial (the spatial extension of Oracle DBMS) allows user to store, index, query, analyze, and deliver raster data and associated metadata. Similarly there is the RasterLite extension for SQLite. Although with different implementations both aimed at the same goal: provide a raster data type that can be handled similarly to the vector data types. The concept implemented by Oracle's GeoRaster doesn't follow the expected behavior. Instead, an auxiliary table using the SDO_Raster

type is created in order to store the gridded data that was assigned to a row in a GeoRaster table. This means that the object and the data are not stored together as expected [Oracle Corporation 2011].

PostGIS Raster follows the same concept of similarity to vector data types behavior, but the raster type objects are complete self-describing values in a table's column. The project also offers a set of SQL functions (like ST_Intersects) to operate seamlessly on vector and raster data. As an example, consider the computation of an elevation profile following a vector feature (for example a road) from a grid (or raster) dataset with elevation values.

3. PostGIS Raster

3.1. Storage

PostGIS Raster aims to implement a minimalistic yet complete raster data structure in the database. There is a single raster type that can be used to define a database entity's attribute. In other words, PostGIS Raster allows the definition of a table where one (or more) columns is of RASTER type. No restriction is implicitly imposed on the raster instances in the column. This means that each row may contain raster tiles that may have different sizes, overlap or snap to different grids [Refractions Research 2011]. Another PostGIS Raster's feature is the option to maintain the raster data itself in it's original data file outside the DBMS, while a representation is stored in the database for organization and storage purposes.

This implementation is flexible enough to allow distinct uses of the Raster type to support the concept of coverages. For example, one row in a table can represent a complete image, a tile of an image or one image that is part of a large mosaic of images. These possibilities fit most users' needs.

3.2. Loading data

Once the Raster type is available in the spatial extension, it is necessary to have tools to insert data into the DBMS. PostGIS 2.0 provides a Python script ("raster2pgsql.py") to upload raster data to a database with PostGIS Raster installed. The script depends only on GDAL [Warmerdam 2008] but has no graphic interface. Sometimes it's difficult for the user to call the script with the right combination of parameters in order to make the best use of PostGIS Raster storage flexibility.

GDAL's command line utilities are able to read raster data stored in database and save it back as a raster file, although, in order to facilitate raster data loading and retrieval from PostGIS database, a graphic tool integrated to a GIS was developed and is described in the next section.

4. The Quantun GIS Plugin for PostGIS Raster

Quantum GIS (or simply QGIS) is an open source GIS able to visualize, analyze and edit vector and raster data. Since it's earliest versions it has been able to handle vector data stored in PostGIS with no raster support. In order to read and write raster data Quantum GIS uses GDAL that since version 1.6 includes a PostGIS Raster driver. Only since version 1.8 the driver became stable and capable of reading irregularly tiled layers

[Refractions Research 2011]. Since GDAL is currently the only library able to read PostGIS's Raster layers and is actively supported by the PostGIS Raster team, Quantum GIS' support for PostGIS Raster was a natural transition.

QGIS Plugin API (Application Programming Interface) provides support for users to add customized tools for specific functionalities The recommended environment for creating plugins is using Python programming language and Qt interface toolkit. Once a plugin is developed and tested it can be published in QGIS' plugin repository[1] maintained by it's development team.

4.1. Loading

QGIS PostGIS Raster Plugin's first goal is to provide a graphical user interface to facilitate the uploading of single and/or multiple raster data to PostGIS Raster, as can be seen in Figure 1.

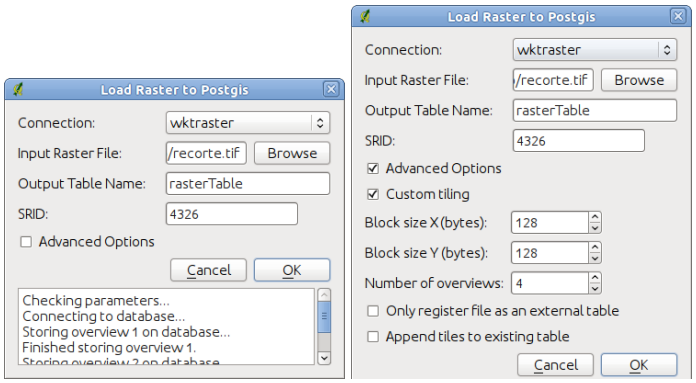

Figure 1. QGIS Plugin interface to upload raster to PostGIS

Connection information is read from QGIS' settings. By default the interface exposes only the minimum parameters needed to upload a single raster file to one table. The interface allows the user to select the number of levels, in the multi-resolution pyramid, to be built. The command line script did not have this functionality, so if a user wanted to create four levels the script would need to be called four times.

Another important parameter that the plugin deals with is the Spatial Reference Identifier (SRID) in which the raster is georeferenced. This is a unique integer identifier, given by a specific authority that should be registered in the database. OGC recognizes the EPSG (European Petroleum Survey Group) as the standard authority to provide SRIDs and some raster formats (for example GeoTiff) include this information. The plugin is able to extract this information and if it is available, avoids the task of finding it manually.

The "Advanced Options" allow users to select the table configuration that should be used, permitting the construction of mosaics and coverages in one or multiple tables. For example, in order to build a mosaic in a single table the user should append many files to one existing table. Since the tiles are self-descriptive the GDAL driver is able to read the mosaic as a single raster coverage.

[1] http://pyqgis.org/

4.2. Visualization

Once a PostGIS database is populated with raster data, users want to explore and visualize them in QGIS. The PostGIS Raster plugin also supports this as shown in Figure 2. Initially the plugin lists all tables that contains raster columns.

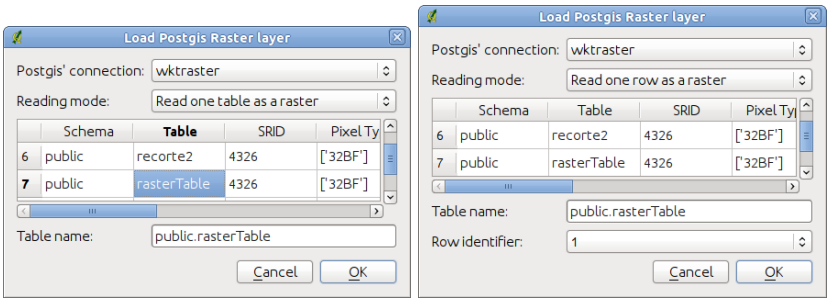

Figure 2. Interface to add a PostGIS raster layer to a Quantum GIS project

The "Reading mode" options allows the user to select how to interpret raster table's arrangement in the database. Since one table can store one raster grid (tiled or untiled) or a mosaic of raster data, this option gives interpretation flexibility.

Choosing the option "Read one row as a raster" the program reads raster data from a single row. It can be a whole image (in a mosaic) or a single tile from a tiled raster table. Whenever the user wants to see the data stored in a table as whole, the option "Read one table as a raster" can be chosen. In this option if each row represents one image, the mosaic is going to be shown. This option also allows vizualization of whole images when a table represents a tiled image. Since some tables can store huge mosaics, the plugin offers the option to "Read the table's vector representation". This allows the user to know what regions each row covers before visualizing the desired area.

4.3. Raster metadata reading

In order to identify the available raster tables, the implementation uses PostgreSQL's internal tables together with the PostGIS Raster's metadata table. This approach allows the plugin to list tables that contain raster columns even if there is no row in the metadata table describing it. This feature makes it possible to explore every storage option described in Section 4.2.

Since PostGIS Raster type is defined as a new PostgreSQL type, it's definition is stored in the "pg_type" table. The query in Listing 1 makes use of this concept by searching every table's attribute for the raster type identifier. This is accomplished by joining the "pg_class" and the "pg_attribute" tables that respectively store tables and attributes lists.

Listing 1. Query to list every existing raster table

```
SELECT relname, attname FROM pg_class AS cla JOIN pg_attribute AS att
    ON att.attrelid = cla.oid AND att.atttypid = 'raster'::regtype ;
```

5. Raster analysis using SQL

Using the WktRaster plugin together with other plugins available it is possible to process raster and vector data on databases with PostGIS enabled. A valid procedure for processing data is to use the "CREATE TABLE AS" statement. There must be an unique integer

column named "rid" to load the table as a Quantum GIS' raster layer and a column named "rast" that contains the raster tiles. Using this syntax the user can store the results of a query as a new raster table and open it for visualization using the WktRaster plugin on Quantum GIS.

Listing 2 shows an example SQL query that applies a threshold of height's above 300m to a raster elevation table. The result of this query is a table containing two columns. One column contains the raster data and the other column contains the unique integer.

Listing 2. Example raster processing using the WktRaster plugin

```
CREATE TABLE testeraster AS SELECT rid, st_mapalgebra(rast, 'CASE WHEN
    rast BETWEEN 0 and 500 THEN 0 WHEN rast BETWEEN 500 and 1000 THEN
    rast ELSE 0 END') as rast from recorte2;
```

6. Conclusion

The plugin developed allows Quantum GIS' users to upload and visualize raster data stored in PostgreSQL databases with PostGIS enabled. This is one of the first implementations that explores the georeferenced raster storage options of PostgreSQL in a GIS environment. This opens path for raster and vector processing using the SQL language.

The plugin explores many of the possible storage options available through the raster type. This flexibility allows easy image mosaicking and improves raster database design options.

The raster visualization speed can be tunned using tiling and pyramids. This can be a workaround for the current GDAL's driver speed as it is in early development and lacks many features that are still being implemented.

References

Imran, M. (2009). Extending an open source spatial database with geospatial image support: An image mining perspective.

Masó, J., Pomakis, K., and Julià, N. (2010). Opengis web map tile service implementation standard.

Obe, R. O. and Hsu, L. S. (2011). *PostGIS in Action*. Manning Publications Co.

Oracle Corporation (2011). Georaster overview and concepts. `http://download.oracle.com/docs/html/B10827_01/geor_intro.htm`, [accessed on Aug 9].

PostGIS (2011). Postgis. `http://www.postgis.org`, [accessed on Aug 9].

Refractions Research (2011). Chapter 8, raster reference. `http://postgis.refractions.net/documentation/manual-svn/RT_reference.html`, [accessed on Aug 9].

Vinhas, L., de Souza, R., and Câmara, G. (2003). Image data handling in spatial databases. In *V Brazilian Symposium on Geoinformatics, Campos do Jordão, Brazil*. Citeseer.

Warmerdam, F. (2008). The geospatial data abstraction library. *Open Source Approaches in Spatial Data Handling*, pages 87–104.