GDMS-R: A mixed SQL to manage raster and vector data

Thomas Leduc, Erwan Bocher, Fernando González Cortés, Guillaume Moreau

GDMS-R: A mixed SQL to manage raster and vector data

Thomas, LEDUC[1], Erwan, Bocher[2], Fernando, González Cortés[3], Guillaume, Moreau[4]

[1]IRSTV, CERMA Laboratory, ENSA Nantes, rue Massenet, BP 81931
44319, Nantes, cedex 3, France
thomas.leduc@cerma.archi.fr

[2]IRSTV, Ecole Centrale de Nantes, rue de la Noë
44321 Nantes, cedex 3, France
erwan.bocher@ec-nantes.fr

[3]IRSTV, CERMA Laboratory, ENSA Nantes, rue Massenet, BP 81931
44319, Nantes, cedex 3, France
fergonco@gmail.com

[4]IRSTV, CERMA Laboratory, Ecole Centrale de Nantes, rue de la Noë
44321 Nantes, cedex 3, France
guillaume.moreau@ec-nantes.fr

Abstract. To evaluate urbanization impact on territories, an accurate knowledge of the urban and peri-urban fabrics is unavoidable. To provide advanced characterization of the terrain, modern GIS applications target even wider geographic areas at finer resolutions but they also have to mix data of different types such as Digital Elevation Model (raster layer), buildings (polygonal layer) and roads (polylines layer). Processing both raster and vector data with the same semantic and in an efficient way presents significant challenges to GIS insofar as underlying granularities but also data layout and processing patterns might be absolutely different.

We have already focused on the definition and the implementation of an abstraction layer called GDMS (Generic Datasource Management System) to handle and process vector data. Main objectives with GDMS, were to provide the user not only a simple and powerful API but also a spatial SQL derived language. Moreover, as an intermediate layer between the user and the information source, GDMS intends to reduce the coupling between the processes and the specificities of each underlying format. As a consequence, former work may easily be reused in a much larger set of scenarii. The learning curve is consequently even simpler.

In this paper, we propose a raster extension to the GDMS layer called GDMS-R. Even if, there is currently no OGC standard concerning raster processing (using well-known SQL language), there already exists a de facto standard called Map Algebra defined by C. D. Tomlin in 1990 and commonly implemented in a wide set of GIS. Our objective is a bit different insofar as we propose to extend SQL language. We present the integration of Map Algebra concepts in GDMS through the GRAP (GeoRAster Processing) language. As for GDMS, reuse is enhanced by the possibility of being vendor-independent (middle-ware approach) and the extension capabilities of the underlying SQL language.

To demonstrate the capabilities of GDMS-R, we present a use case relative to the deep impact of increased urbanization on the vulnerability of peri-urban hydro-systems: impact of the linear constraints on the runoff water pathways and accumulation that uses both vector and raster data in an unified way.

Keywords: processing language, spatial SQL, raster, GDMS

1 Introduction

As mentioned in [22], “the first major component of [a geographic] information system is its data, the second component is a set of data-processing capabilities”. Data that need to be processed in a GIS depend on the nature of the spatial phenomenon involved. In a vector or drawing-based GIS, atomic data corresponds to vertices, (chains of) line segments, polygons… whereas in a raster or image-based GIS, the atomic data (or location) corresponds to a pixel or grid cell (i.e. a regularly spaced planimetric position).

All data relative to a particular geographic area of interest are stored in a single map called a layer. It corresponds to a bounded plane surface, where all atomic data share the same scale, orientation and planimetric projection.

Vector layers most often represent human-drawn zones and are easily scalable, they are compatible with topological operations. They usually represent in a compact manner borders of homogeneous zones whereas raster data is most often the result of an automated acquisition process where the mesh is linked to the nature of the sensor rather than the phenomenon that is analyzed. Their accuracy is limited to mesh resolution and data storing might be an issue. They have the advantage of being able to compare heterogeneous data with very simple operations. To process those layers (the digital cartographic variable), the need for a spatial query formalism has been clearly identified and thoroughly studied over the last years. As noticed by [11], "many studies focusing on the management of vector data in [Spatial DataBase] have progressed rapidly in the past".

In this paper, we propose GDMS-R, a raster extension to the GDMS abstraction layer [3]. Even if, there is currently no OGC standard concerning raster processing (using well known SQL language), there already exists a *de facto* standard called Map Algebra defined by [20] and commonly implemented in a wide set of GIS. Our objective is a bit different insofar as we propose to extend SQL language. We present the integration of Map Algebra concepts in GDMS through the GRAP (GeoRAster Processing) language. As for GDMS, reuse is enhanced by the possibility of being vendor-independent (middle-ware approach) and the extension capabilities of the underlying SQL language.

The rest of the paper is organized as follow; first, we will present the motivation for our architecture (need for a common language) and most common proposals, then we will introduce Map Algebra and the GDMS concepts. Afterwards, we show how they have been combined into GDMS-R. We will finish by a use case that demonstrates the potentialities of our approach.

2 Related work

2.1 Spatial query formalism

Numbers of approaches have been proposed to address the need for formalizing spatial queries. Some of them have implemented spatial extension to the well-known Structured Query Language (SQL), assuming that it was much more powerful and easy to retrieve and analyze spatial data using the familiar select - from - where statement instead of procedural commands like macro language [7]. Indeed, the SQL is a world famous declarative programming language dedicated to CRUD (CRUD stands for the ability to Create new entry, as well as Read, Update and Delete existing entry in a data set) operations in the context of Relational DataBases Management System (R-DBMS). In their reviews of proposed query languages, among many other spatial data
manipulation languages, [6, 7] successively mention:

- [8], its graphic query language (GQL) structured in SQL fashion is "designed to conform with ISO standard 7942 (graphical kernel system) and ISO standard 9075 (database language SQL)";
- [15], its Geoql is an extension of SQL which provides spatial operators,
- [7], its Spatial SQL is defined as a query and presentation language designed to preserve SQL concepts but also to incorporate spatial operations and relationships.

The processing of vector-encoded data through a spatially enabled and extended SQL language has thus already been normalized by the Open Geospatial Consortium in the Open-GIS "Simple Features Specification for SQL" [9, 10]. There also exist several efficient and industrial implementations that spatially enable R-DBMS (such as the PostGIS, a spatial extension for PostgreSQL).

2.2 Motivations

Even if, as noticed by [11], the amount of raster data has "exploded recently as a result of the quick development of the remote sensing technique", no real framework to handle raster data type in a

spatial SQL way has been defined. Indeed, there is no OGC standard relative to raster processing. Nevertheless, there already exists a *de facto* standard called Map Algebra. As written in [13] “the term ’Map Algebra’ was first introduced in the late 1970s [23] and has since been used in loose reference to a set of conventions, capabilities, and analytical techniques that have been widely adopted for raster-based GIS [20, 21]”.

The purpose of this algebra-like language is to create new map layers using existing map layers (treated as input variables) and sequenced dedicated operations. It takes a location-oriented perspective or “worm’s eye” view by computing new values for locations based on the location itself, its neighborhood, a related zone, or the entire dataset.

[4] makes an inventory of some recent extensions to Map Algebra. Thus, among some others, it mentions successively:

- the GeoAlgebra presented in [18], an extension of map algebra that allows for flexible definitions of neighborhoods,
- the MapScript language presented in [16] “that allows control structures and dynamical models to be incorporated into map algebra”,
- [13] “propose an extension of map algebra for spatio-temporal data handling”.

Map algebra has been incorporated in many GIS and remote sensing image processing softwares (such as ArcGIS Spatial Analyst developed by ESRI Inc. or GRASS raster analysis modules…). For [13] “One of the reasons for the widespread adoption of conventional map algebra for raster processing in GIS is its simple syntax and the ability to string together multiple functions to create more complex models”.

In the DBMS context, there already exists efficient raster data support in Oracle 10g (with the Oracle GeoRaster object) and some prototype in PostgreSQL/PostGIS (with the PGRaster object)… With its GSQL-R, [11] demonstrates that an extended SQL “provides a convenient way for raster data access and manipulation”.

3 Map Algebra and GDMS concepts

3.1 Overall view of GDMS

GDMS [3] is related with a similar layer used in the gvSIG project to manage the alphanumeric data access. This layer is called GDBMS [2] and one of its main limitation is that it can only be used for alphanumeric purposes. GDMS work is a general refactoring that mainly adds spatial functionalities and a more powerful SQL processor to GDBMS.

As noticed in [3], the heterogeneity of source types makes difficult the reuse of algorithms that are tightly coupled with the specified file format, database vendor, etc. With an intermediate layer between the user and the information source, the work developed by the former will not be coupled with the specificities of each format but with the intermediate layer itself, letting the work to be reused in a much more wide set of scenarios and of course simplify the learning curve for new developers.

One of the problems that such a layer arises is that it has to be able to access any potential source type. As the current number of source types is huge and can grow indefinitely, the user has to be able to extend the access capabilities of the layer. By introducing the concept of driver, GDMS intends to potentially fit any situation, even if the types of sources are not well known or currently the layer does not have the driver to access it.

With GDMS, the idea was to postpone all problems of interoperability across data repositories, but also about the SQL semantics by developing a highly flexible, portable and standards compliant tool to build SQL queries. Currently, GDMS is dedicated for GIS clients but in the future it will also be a software component for SDI to process spatial data according to Web Processing Service.

GDMS provides an API that lets the user operate independently of source type. However, this API is

not friendly enough for an end-user and reduces the acceptance of the solution by final users. With the purpose of simplifying both access and manipulation of data sources, GDMS provides a SQL processor that lets the execution of the common Data Manipulation Language statements against any source mapped by a driver. To avoid introducing a new grammar, GDMS fully preserves the SQL-92 grammar and adds to this standard geometric concepts and spatial functions as in OGC simple features SQL specification. As an analogy to spatial SQL for RDBMS, GDMS provides an extended SQL query language on heterogeneous data types.

It is the main purpose of GDMS to improve data creation and sharing. As in an SDI the consumption of data is as important as the production and sharing of data, the SQL processor in GDMS allows data feedback as if they were new data sources (materialized views). This means that the result of SQL queries can easily be integrated into the SDI as a new data source. Those data will be ready to be used by further SQL statements as any other existing data source.

Finally, to improve further code reuse, GDMS introduces the concept of Geo-knowledge repository. GDMS allows an extension to the semantics of the SQL language in terms of functions and custom queries. Functions and custom queries are artifacts that contain the implementation of some operations on the data and can be reused just by referencing its name into an SQL statement. This way, some user can implement a buffer operation and other can reuse it just by calling buffer in an SQL query: SELECT buffer(the_geom, 20) FROM mydata. The collection of all this artifacts is the Geo-knowledge repository. The purpose of GDMS is to maintain such a repository and encourage the feedback of new artifacts from the user to make it a growing knowledge base.

The Geo-knowledge repository is more than a library of functions that can be extended by new user-defined add-ons; we aim to go a step further with this concept. Indeed, the goal is to provide a high level of spatial semantics:

- checking: a “just in time” validation process of input data type, range… made by a dedicated spatial SQL pre-processor,
- sharing: future releases will embed network-group-ware capacity because users need also to share spatial process and not only geographic data,
- interoperability: the GDMS core is fully OGC compliant, so it is possible to copy and paste spatial SQL scripts from/into PostgreSQL/PostGIS.

At last, our Geo-knowledge repository provides not only spatial functionalities but also description and (sometimes) useful use-cases for each of them.

3.2 Coupling GDMS with Map Algebra

GDMS features are inspired by the relational model theory for database management and its tuple calculus realization. So, in GDMS, a DataSource is an array of rows, each of them may contains several columns (i.e. fields of any Value type: from a JTS Geometry [5] to an alphanumeric attribute value).

In term of granularity, classical GDMS is able to deal with atomic objects such as JTS Point, LineString, Polygon… or integer values.

In the pure imaging and computer vision domains, the atomic element is the pixel (which stands for picture element). But, for efficiency reason (images encoding through pixels array are suitable for wide areas such as the one used by remote sensing technologies or geo-computational simulations… it would be a nonsense to explicitly deal with SQL tables of pixels fields), the single data type used by Tomlin’s model is the map. The mixed SQL we will present does not explicitly process pixels, but layers or maps (that can be stored as arrays of pixels).

[4] notices that Tomlin’s Map Algebra defines three types of functions:

- “local functions: involve matching locations in different map layers, as in classify as high risk all areas without vegetation with slope greater than 15%,
- *focal* functions: involve proximal locations in the same layer, as in the expression calculate the local mean of the map values,
- *zonal* functions summarize values at locations in a layer contained in zones defined in another

layer, as in example given a map of a city and a Digital Terrain Model, calculate the mean altitude for each city".

In [22], Tomlin mentions a fourth major class of map algebraic operation called incremental functions. "The incremental operations compute a new value for every location as a function of its lineal, areal, or surficial form on a specified layer. These operations may indicate each location's length or shape as part of a lineal network; its surface area, frontage, or shape as part of an areal pattern; or its slope, aspect, drainage direction(s), or volume as part of a surficial form".

As an example, we present two operations using the spatial SQL syntax defined in GDMS-R. The first one is an instance of a focal function (a mean computation), while the second one belongs to the fourth major class (even if the incremental classification label does not really suit us):

SELECT FocalMean(raster, 'rectangle', 3, 3) FROM mydem;
SELECT CropRaster(raster, the_geom) FROM mydem, zoneofinterest;

A mapping exists between GDMS-R and Map Algebra so all operations can be executed in a similar manner. For example, to know the maximum slope by parcels, the following query is used:

SELECT slope(raster) FROM myDem into slope;
SELECT zonalmax(parcels.raster, slope.raster) FROM parcels, slope;

4 Implementation details

4.1 GDMS architecture and data model

GDMS has a layered architecture (fig. 1) where the upper layer contains the SQL interface and the Geo-knowledge repository and the lower one is the driver layer. It contains details that are specific to each the source types. In case of the ESRI Shapefile driver, it contains implementation details to open files, decode the information and transform the contents into the GDMS internal model. PostgreSQL drivers will open the connection and will transform the information of the table into the internal model. This layer also provides extensibility to GDMS in terms of data source supported types. It is possible to create a driver to support potentially any type of data source.

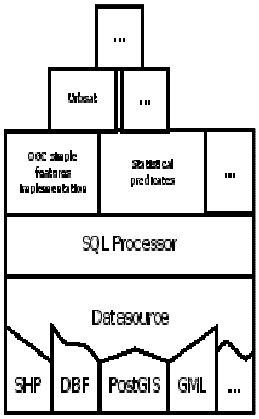

Fig. 1. GDMS layered architecture

In spite of its extensibility, the driver layer is absolutely heterogeneous and building a SQL processor on top of such a layer would be very complicated. To solve this problem we place the Datasource layer on top of the drivers one. The main responsibility of this layer is the creation of a common API to manage all source types.

The SQL processor is built on top of this Datasource layer so it uses the common interface it provides to access any of the available data source types. Consequently it is possible to mix different types of data sources in the same SQL statement. For example it is possible to join a shapefile with a CSV file and a PostgreSQL table in one single SQL query. To reference the sources in a SQL statement it is

necessary to map a name to each involved data source before.

We have seen that the Datasource layer deals with the heterogeneity of the drivers and provides a common interface to the upper layers. We will now focus on the data model that is used to implement all features. It consists of one main entity called DataSource. A DataSource is an abstraction that allows the management of one unique data source. This means that it is necessary to have as many DataSource instances as the number of sources to be accessed.

This abstraction has a typical tabular structure that uses rows to store each of the elements of the data source. This structure is close to the JDBC standard, where each row consists of a set of fields with a common structure for all rows in the DataSource. It is possible to see a relationship between this model and the GeoAPI model (currently being standardized in ISO 19123) where FeatureCollection and Feature respectively match DataSource and its rows. Finally the field values of each row match the attribute values of a Feature.

All the information about field types is stored into a Metadata object that contains all field names, field types and restrictions for the types, such as length for string values, and coordinate reference system for spatial types. The model in this point allows a wide range of types for the fields that are compliant with the JDBC standard for alphanumeric types or with the OpenGIS Simple Features Specification for spatial types. Also note that there is no restriction in field types for a DataSource, it can have zero or more fields of any type so it is as possible to have several spatial fields as to have just alphanumeric.

4.2 GDMS-R: a data model fully integrated into GDMS

GRAP (GeoRAster Processing) provides the core functionality required to support the use and the process of images. It is based on the ImageJ [17, 1] library. ImageJ is an image processing and analysis program (but it is also an API) that support a huge amount of image formats, written in Java, by Wayne Rasband at the U.S. National Institutes of Health.

In our own GRAP library, a raster is wrapped in a common object called a GeoRaster. A GRAP GeoRaster object is an association of an array of pixels stored in an ImageJ Image-Plus object and a set of some spatial fields (mandatory to locate the raster in the earth!) such as: a projection system, an envelop, a pixel size… GRAP's metadata matches the world file specifications. Indeed, the first pixel of the pixels grid (with row and column indexes both equals to zero, at the upper left corner) corresponds, in the real world, to the centroid of the corresponding UpperLeft rectangle.

In GDMS presentation relative to the internal data model, we have written that the DataSource was the abstract entry point for each unique data source (of geometry type). This assertion remains true for data source of raster type. Indeed, DataSource abstraction may also be convenient to store and to handle tabular set of multiple GeoRaster columns. The same way we did for GeometryValue (it is the type of any geometric field, based on the JTS Geometry object class), we have implemented a RasterValue (it is the type of any GeoRaster field, see fig. 2).

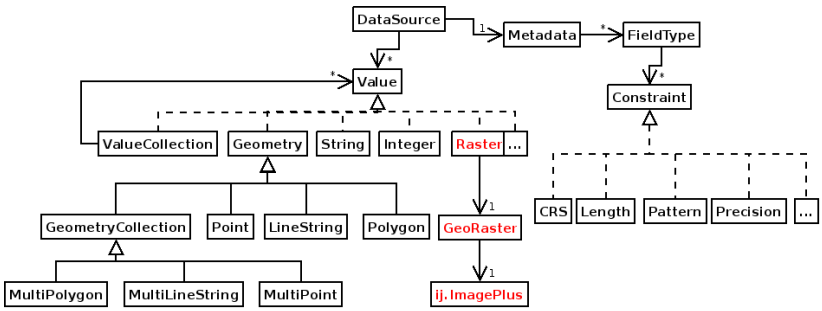

Fig. 2. GDMS-R extended data model

The main advantage of this approach, is that with this kind of GRAP and GDMS coupling, we are able to handle in a single DataSource quite heterogeneous and complex data set. Indeed a DataSource is able to store multiple rows of multiple columns of any Value type. Due to the object-oriented programming paradigm we use, and because both Raster-Value and GeometryValue are subclasses

of Value (that is to say more specialized versions of Value…), we are able to store in the same row several geometric and raster fields!

In the same way, it is also possible to store in a single GDMS DataSource as many rows of raster than there are small elements of image maps covering a larger area in tiling placement technique!

4.3 GDMS-R extensibility through raster functions

Even if ImageJ is a quite comprehensive library to process raster data, GRAP API offers the possibility to implement new process on GeoRaster. This extension mechanism has to be made through GRAP Operation (which is a Java interface dedicated to the implementation of GeoRaster processing).

Since new GRAP Operation has been implemented, it is possible to benefit from GDMS extension mechanisms. Indeed, GDMS might be customizable trough queries that are able to deal with DataSource of any kind (as input or output variables). The only requirement is to fulfill some preprogrammed prerequisites concerning both input and output DataSources. Indeed, polymorphism is not always available for each GDMS SQL query.

5 Example case study

In this section, the main objective is to present a proof of concept concerning the relevance of this mixed SQL in advanced spatial hydrological analysis. What will be presented, will be scripted using the GDMS-R spatial SQL language independently from the input layer type. Here is a general survey of the main steps of the processing suite:

- select a Region Of Interest in the DEM and “crop” it,
- compute the D8 flow direction grid (using the D8 method defined by [14]),
- compute the flow accumulation grid,
- compute the flow accumulation grid taking linear constraints into account,
- compute the watersheds and convert them to vector.

The need of a mixed SQL is obvious here because we want to take linear constraints into account to increase the accuracy of the simulations (in both flow routing and flow accumulation algorithms). Two input data are used: a DEM (Digital Elevation Model) stored in an ESRI ASCII grid format and a roads ESRI Shapefile that contains a set of polylines (see fig. 3).

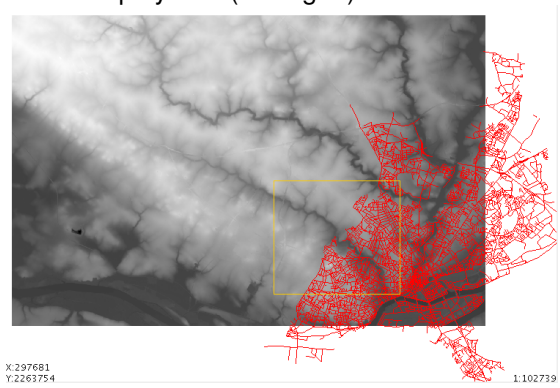

Fig. 3. Input data (DEM, roads in red) and area of interest (orange fence)

At first, we decide to limit the analysis zone to a smaller area of interest (also represented in fig. 3). This operation is expressed as:

SELECT CropRaster(raster, the_geom) AS raster FROM dem, fence;

where fence is the limited area. By calling the create table statement, we can store the Crop-Raster result into a GDMS-R internal format.

CREATE TABLE smalldem AS SELECT CropRaster(raster, the_geom) AS raster FROM dem, fence;

The following examples show a sequence of spatial hydrological analysis to compare the impact of roads on the runoff paths. We use algorithms: ”D8 grid direction” and “D8 accumulation” [14]. This first one computes for every cell the maximum downslope (see fig. 4) and sets the cell with an integer encoded value (such as 1 for East, 2 for North-East…). The second computes the contributing area for each grid cell according to the grid direction. For a cell, it represents its own contribution added to the contribution of upslope neighbors than drain into it. It is used to evaluate the potential water flow accumulation. D8 algorithms are applied this way:

```
CREATE TABLE direction AS SELECT D8Direction(raster) AS raster FROM smalldem;
CREATE TABLE accumulation AS SELECT D8Accumulation(raster) AS raster FROM direction;
```

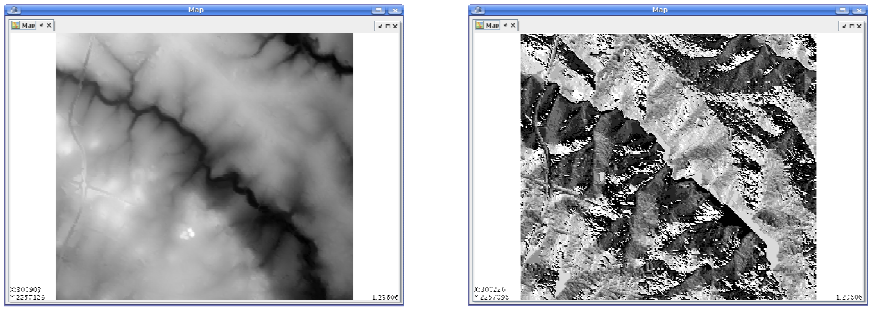

Fig. 4. Original DEM and D8 grid direction result

To take into account the impact of roads, we have modified the grid accumulation algorithm. A new function has been created; it requires both a grid direction and a constrained grid. The latter represents the roads as a set of connected pixels with values greater than or equal to 1. For this, the road file is transformed into a raster using the RasterizeLine function (see fig. 5). Finally, the output raster datasource is included to the D8ContrainedAccumulation call.

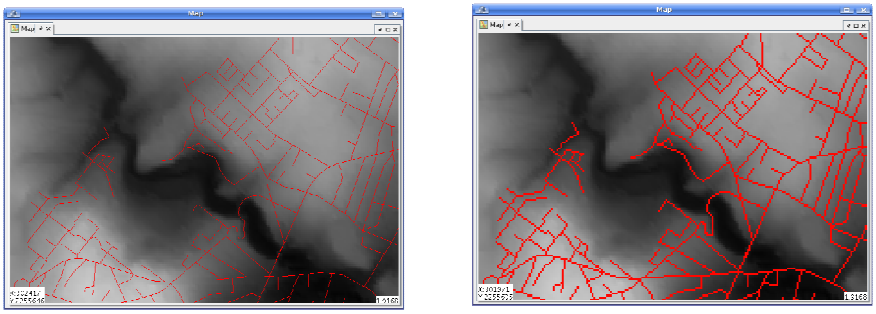

Fig. 5. Roads rasterization process: from a roads as shapefile (left) to a roads as grid (right)

```
SELECT RasterizeLine(the_geom, raster, 1) AS the_geom FROM roads, smalldem;
SELECT D8ConstrainedAccumulation(d.raster,c.raster)
    AS raster FROM direction AS d,rasterizedroads AS c;
```

Fig. 6 presents a comparison between those two types of accumulation grid. As it can be seen, the blue water flow is stopped by the road.

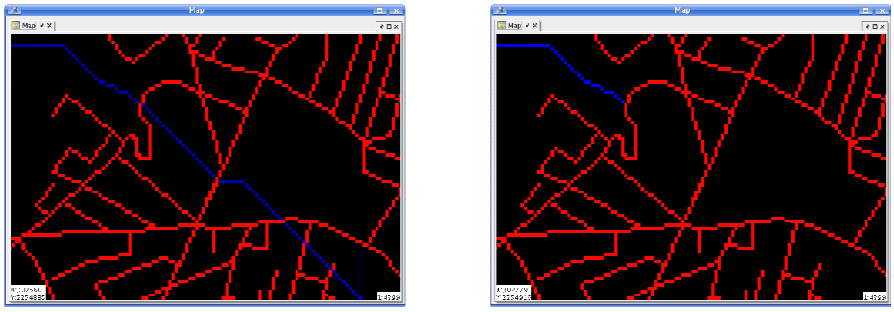

Fig. 6. D8 grid accumulation (not constrained and constrained)

We will now proceed with further GDMS-R capabilities in computing all the watersheds as well as their area and their maximum slopes (see fig. 7).

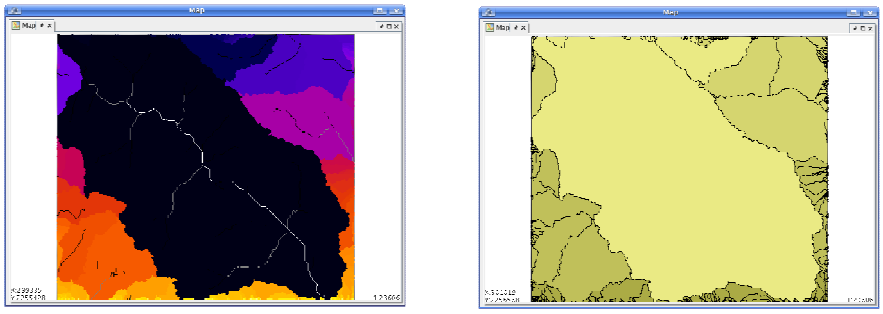

Fig. 7. Watershed as raster data and as polygons

```
CREATE TABLE watersheds AS SELECT D8Watersheds(rs) AS rs FROM direction;
CREATE TABLE watershedspolygons AS SELECT RastertoPolygons(rs) FROM watersheds;
SELECT Area(the_geom) AS the_geom, * FROM watershedspolygons;
```

The last statement builds the maximum slope for each watershed. It applies the Tomlin zonal syntax and shows how to combine several functions.

```
CREATE TABLE maxslope AS
SELECT ZonalMax(w.raster,D8Slope(d.raster)) AS raster, * FROM watersheds AS w, dem AS d;
```

where watersheds is an integer-valued raster that identifies the zone for each cell. DS8lope is a function that computes the maximum slope for each cell around a 3x3 neighborhood.

6 Conclusion and future works

In this paper, we have presented GDMS-R a raster extension to the GDMS abstraction layer: it provides a convenient and efficient way to address and to process spatial data of both vector and raster types. This layer is thus able to produce either a raster, either a vector resulting DataSource layer, starting from any type of input DataSource (it depends of course on the corresponding spatial process).

What is now obvious, is that this abstraction layer is a major performer in the domain of the spatial data geo-processing. Indeed, on the client-side, it has already been strongly coupled with a desktop GIS called OrbisGIS6. On the server-side, a prototype based on GDMS-R has also been developed.

Future work requires addressing level of detail (i.e. sampling level) as far as raster data are concerned; whatever the resolution of the input data, it will be useful to be able to specify the adequate resolution for a geo-process. This will also lead to include further algorithms dedicated to raster-vector transform, thus being able to overcome differences in meshing methods, mesh size... Some work has already been started in that direction in order to study the sensitivity of mesh type, direction and size [12].

The other main working direction deals with Spatial Data Infrastructure and we will look forward at encapsulating GDMS-R, on the server/dispatcher-side, in a Web Processing Service. This is one of the future main steps in our roadmap.

7 Acknowledgments

The AVuPUR project was funded by the French Agence Nationale de la Recherche (ANR) under contract ANR-07-VULN-01.

8 Availability

A GDMS-R implementation included in the OrbisGIS frontend and developed under GPL license, is available for public download at: http://sourcesup.cru.fr/projects/orbisgis

References

[1] M. Abramoff, P. Magelhaes, and S. Ram. Image Processing with ImageJ. In *Biophotonics International*, volume 11, pages 36–42, 2004.

[2] A. Anguix and G. Carri□n. gvSIG: Open Source Solutions in spatial technologies. In *GIS Planet*, Estoril, Portugal, may-june 2005.

[3] E. Bocher, T. Leduc, G. Moreau, and F. González Cortés. GDMS: an abstraction layer to enhance Spatial Data Infrastructures usability. In *11th AGILE International Conference on Geographic Information Science - AGILE'2008*, Girona, Spain, may 2008.

[4] G. Câmara, D. Palomo, R. Cartaxo Modesto de Souza, and O. R. Fradico de Oliveira. Towards a generalized map algebra: principles and data types. In *Proceedings of the 7th Brazilian Symposium on GeoInformatics - GEOINFO 2005*, Campos do Jordão, Brazil, November 2005. ISBN: 85-17-00022-6.

[5] M. Davis and J. Aquino. JTS Topology Suite - Technical Specifications. Technical report, VIVID Solutions, October 2003. www.vividsolutions.com/JTS/bin/JTS Technical Specs.pdf.

[6] J. L. de Oliveira and C. B. Medeiros. User interface issues in geographic information systems. Technical Report IC-96-06, July 1996.

[7] M. J. Egenhofer. Spatial SQL: A Query and Presentation Language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[8] P.-C. Goh. A graphic query language for cartographic and land information systems. *International Journal of Geographical Information Systems*, 3(3):245–255, 1989.

[9] J. R. Herring. OpenGIS Implementation Specification for Geographic information – Simple feature access - Part 1: Common architecture, October 2006. www.opengeospatial.org/standards/sfs.

[10] J. R. Herring. OpenGIS Implementation Specification for Geographic information – Simple feature access - Part 2: SQL option, October 2006. www.opengeospatial.org/standards/sfs.

[11] Y. Liu, Y. Wang, Y. Zhang, X. Lin, and S. Qin. GSQL-R: A query language supporting raster data. In *EEE International Geoscience and Remote Sensing Symposium, IGARSS*, volume 7, pages 4414–4417, september 2004.

[12] N. Long, E. Bocher, T. Leduc, and G. Moreau. Sensitivity of spatial indicators for urban terrain characterization. In *IEEE International Geoscience and Remote Sensing Symposium, IGARSS-2008*, Boston, Massachusetts, U.S.A., july 2008.

[13] J. Mennis, R. Viger, and C. D. Tomlin. Cubic map algebra functions for spatio-temporal analysis. *Cartography and Geographic Information Science*, 32(1):17–32, 2005.

[14] J. O'Callaghan and D. Mark. The extraction of drainage networks from digital elevation data. *Computer vision, graphics, and image processing*, 28(3):323–344, 1984.

[15] B. C. Ooi. Efficient Query Processing in Geographic Information Systems, volume 471 of *Lecture Notes in Computer Science*. Springer, 1990.

[16] D. Pullar. Mapscript: A map algebra programming language incorporating neighborhood analysis. *Geoinformatica*, 5(2):145–163, 2001.

[17] W. S. Rasband. ImageJ. http://rsb.info.nih.gov/ij/, 1997-2007. U. S. National Institutes of Health, Bethesda, Maryland, USA.

[18] M. Takeyama and H. Couclelis. Map dynamics: integrating cellular automata and GIS through Geo-Algebra. *International Journal of Geographical Information Systems*, 11:73–91, 1997.

[19] D. G. Tarboton. A new method for the determination of flow directions and contributing areas in grid digital elevation models. *Water Resources Research*, 33(2):309–319, 1997.

[20] C. D. Tomlin. *Geographic Information Systems and Cartographic Modeling*. Prentice Hall, 1990.

[21] C. D. Tomlin. Cartographic modeling. In D. R. M. Goodchild, D. Maguire, editor, *Geographical information systems: principles and applications*, pages 361–374. Harlow, Essex, United Kingdom: Longman Group Ltd, Harlow, Essex, United Kingdom, 1991.

[22] C. D. Tomlin. Map algebra: one perspective. *Landscape and Urban Planning*, Elsevier Science, 30:3–12, 1994.

[23] C. D. Tomlin and J. K. Berry. A mathematical structure for cartographic modeling in environmental analysis. In *American Congress on Surveying Mapping, 39th annual symposium*, pages 269–283, Falls Church, Virginia, U.S.A., 1979.