DUBLIN INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
ESTABLISHED 1887
MEMBER of the EUROPEAN UNIVERSITY ASSOCIATION

School of Computing
Scoil na Ríomhaireachta

The development of OpenMap™ to support PostgreSQL/PostGIS queries and spatial data types.

Declan Lynch, Dublin Institute of Technology, Control Systems and Electrical Engineering, Kevin Street, Dublin 8, Ireland, Declan.Lynch@dit.ie
Patrick Browne, School of Computing, Dublin Institute of Technology , Kevin Street, Dublin 8, Ireland. Patrick.Browne@comp.dit.ie
Mark Foley, School of Computing, Dublin Institute of Technology , Kevin Street, Dublin 8, Ireland. mark.foley@dit.ie

1. Introduction

The theme of this paper is to report on the development of two graphical clients using OpenMap™ (BBN-Technologies 2004). The first client supports the native geometric data types of the PostgreSQL DBMS. This development was exploratory in nature. Our main goal was to develop a client that supports the spatial objects of PostGIS(PostGIS 2004), the spatial extension of PostgreSQL(PostgreSQL 2004). Both clients allow the user to issue SQL commands to the DBMS using different JDBC extensions that support spatial data types. The resulting data is displayed in OpenMap™. In both clients a layer was created capable of converting the database types into the graphic objects required by the OpenMap™ framework.

There are many interdependencies between spatial database technology and GIS. For example, the database provides the storage and retrieval tools needed by the GIS. It is the feasibility of using PostgreSQL/PostGIS as the backend to a GIS that motivated this work. This paper is based on a final year undergraduate project undertaken at the School of Computing, Dublin Institute of Technology. The work was done by the first author and supervised by the second and third authors.

We first discuss the software used throughout this work. We then look at the OpenMap™ client implementation for the native geometric data types of PostgreSQL. Then we present an example of the PostgreSQL/PostGIS client. Finally, we summarize our conclusions.

2. Software used

2.1 PostgreSQL

PostgreSQL is an object-relational database management system (ORDBMS) based on POSTGRES. PostgreSQL supports SQL92 and SQL99 and offers many modern features: complex queries, foreign keys, triggers, views, transactional integrity, multi-version concurrency control. Also, PostgreSQL can be extended by the developer in many ways, for example by adding new: data types, functions, operators, aggregate functions, index methods and procedural languages. PostgreSQL, version 7.4.1, was used throughout this work.

2.2 PostGIS spatial extensions for PostgreSQL.

PostGIS is an extension to the PostgreSQL that allows geographic objects to be stored and queried in the database. PostGIS was developed by Refractions Research Inc, a GIS and database consulting company, who produced PostGIS as a research project in open source spatial database technology. PostGIS makes use of the extensibility of PostgreSQL. It consists of a set Abstract Data Types (ADTs). PostGIS implements the OGC Simple Features Specification for SQL (OGSFSSQL). PostGIS version 0.8.2 was used throughout this work. As of version 0.8.2, PostGIS conforms to the vast majority of the OGSFSSQL. PostGIS provides a far richer set of server-side functionality to a GIS than that provided by the PostgreSQL native geometric types. It is important to note that PostgreSQL native geometric data types were never intended to conform to OGC specifications. In general, the graphical information stored in a relational database system like PostgreSQL/PostGIS can be divided into one of four different categories: Point, Line, Polygon, and Text. These are used to construct higher level *geographic* objects or features with the following components:

- A spatial or geometric attribute, or the spatial extent. This component describes the location, shape, orientation and size of the object in either two-dimensional or three-dimensional space.
- A non-spatial attribute. These attributes are also referred to as thematic or descriptive attributes, for example the name of a lake, or the population of a town.

Point, Line and Polygon (and collections of each) form part of the OpenGIS Simple Features Specification (OGSFS). The OGSFSSQL is used by the PostGIS extension to PostgreSQL. Note there are differences between the OGC's general and SQL standards. Both PostgreSQL and PostGIS distributions include distinct JDBC extension software to enable a Java client to extract the complex spatial data from the database. The OGSFSSQL specifies an Application Programming Interface (API) providing for publishing, storage, access and operations on OGSFS. The OGSFS are specified in the OpenGIS Abstract Specifications.

2.3 The graphical client OpenMap™

The database is only part of a GIS. We also need a graphical client to view and interact with the data stored in the database. An existing open source JavaBeans™ based programmer's toolkit called OpenMap™, developed by BBN Technologies, was chosen. Using the OpenMap™ framework allows applications to be quickly built. However, as with all complex frameworks, it is a non-trivial task to learn the internal structure of the OpenMap™ framework.

The OpenMap™ framework by default is capable of reading spatial data from ESRI shape files, which are a custom file format developed by ESRI for use with their ArcInfo GIS product. The test data supplied with the OpenMap™ distribution is all in this format. The PostGIS spatial extension includes a tool for converting ESRI shape file data into PostGIS tables conforming to OGSFSSQL. So both client and DBMS can read industry standard shape files and PostGIS can convert them to OGSFSSQL. Our goal is to get OpenMap™ to directly access OGSFSSQL data from with PostgreSQL/PostGIS.

3. JDBC for PostgreSQL native types

PostgreSQL comes equipped with a customized JDBC driver. In this work the PostgreSQL JDBC driver, version 7.4.1, was used. The PostgreSQL 7.4.1 DBMS provides seven native geometric data types. These are: box, circle, line, line segment, path, point, polygon. The line data type is included though not implemented (Sept 2004). A large set of operations to enable SQL queries on the geometric data types is provided (e.g. basic functions, set operations and spatial analysis queries). Note that topological operators, such as adjacency, are not supported. See PostGIS site for full list. There is also a set of functions allowing the performance of computations on geometric objects. Among these functions are area, which computes the area of an object with a surface, and length, which computes the length of a linear object.

To make use of native geometric data types one must use PostgreSQL extensions to the JDBC in a package called `org.postgresql.geometric`. This package contains seven classes to support the geometric data types. The classes are: `PGbox`, `PGcircle`, `PGline`, `PGlseg`, `PGpath`, `PGpoint`, `PGPolygon`. Each of these classes extends the `PGobject` class contained in the package `org.postgresql.util`. The `PGobject` class provides the state and behaviour expected of a generic geometry type and acts as a template for the subclasses. This class can be used to store objects of any of its subclasses retrieved from a database. A `PGobject` instance can be got using `java.sql.ResultSet.getObject()` method. A `PGobject` instance can then be tested for class and cast to that class for further processing. Each of the classes implements the Java `Serializable` and `Cloneable` interfaces. These interfaces describe the functionality for objects of the class to be serialized and streamed over a network and to be cloned respectively.

4. Programming OpenMap™ to query PostgreSQL native types

As OpenMap™ is a framework it has a default behaviour (the Hollywood Principle). On start up the OpenMap™ client sends pre-formed SQL commands to the server. One SQL command is sent per layer to be displayed, the syntax of the command is retrieved from the appropriate properties file. The PostgreSQL JDBC extension provides classes to encapsulate geometric data types for use in a Java program [as described in section 3]. The PostgreSQL server returns encapsulated objects representing each geometric object that is to be displayed in a layer, based on the result of executing the query. The client converts these objects into the graphical objects required by the framework and they are displayed onscreen. OpenMap™ provides a drawing tool that allows the user draw geographic objects in a map. Our client allows the user to save these objects as geometric types in a chosen table in a PostgreSQL database. We also developed a tool that enables the user to issue ad-hoc SQL queries to the database; data satisfying a query is returned from the database and displayed in the client. Space does not permit a full description, the four classes are briefly described in Table 1.

Java Class	Purpose
`PostgreSpatialLayer`	Retrieves geometric data from a PostgreSQL database and display the geometric objects.
`PgsqlExport`	Provides methods for exporting an `OMGraphicList` to a PostgreSQL database as native geometric data types
`PgsqlDrawingToolLayer`	Makes use of the `PgsqlExport` class.
`PgslSqlTool`	Allows the user send ad-hoc SQL queries to PostgreSQL.

Table 1. Java classes that were developed for the PostgreSQL client application

It was necessary to construct a set of mappings between the native data type, the JDBC data types, and the OpenMap™ data types. The mapping that were chosen are listed in Table 2.

PostgreSQL Geometric Data Type	PostgreSQL JDBC Extension Class	OpenMap OMGraphic Class
`Point`	`PGpoint`	`OMRaster`
`Lseg`	`PGlseg`	`OMLine`
`Box`	`PGbox`	`OMRect`
`Circle`	`PGcircle`	`OMCircle`
`Path`	`PGpath`	`OMPoly`
`Polygon`	`PGpolygon`	`OMPoly`

Table 2. Mapping PostgreSQL data types to OMGraphic classes

5. An example of using PostgreSQL native data type with OpenMap™

The example schemas were taken from a standard text book on spatial databases (Rigaux, Scholl, and Voissard). The `geometry` and `name` data for relations `LandUse` and `Section` is displayed in the client as shown in the screenshot in Figure 1.

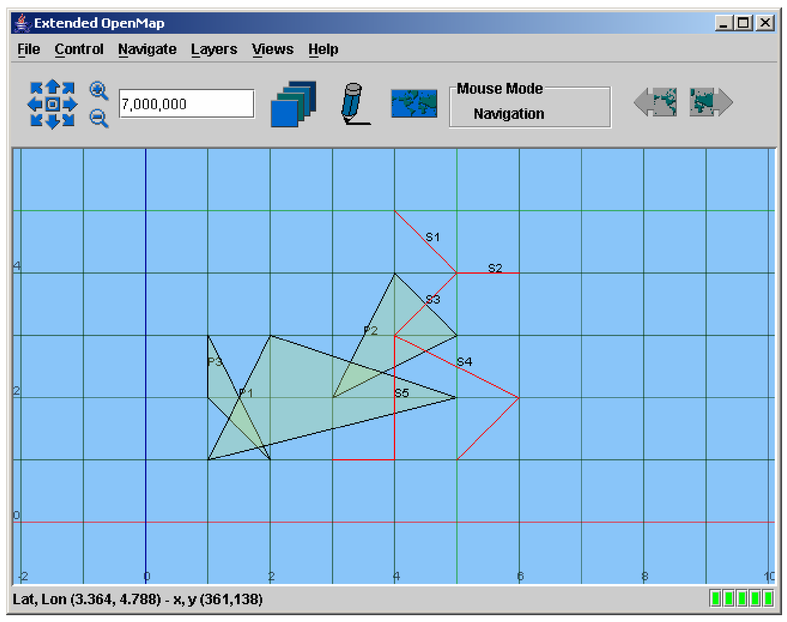

Figure 1. Example geometric data displayed in Java client

Here is an example query "Display all land parcels intersected by section S5?". This can be implemented in SQL as:

```
SELECT l.geometry
FROM LandUse l, Section s
WHERE s.geometry ?# path(l.geometry)
AND s.name = 'S5';
```

The result of inputting and executing this query using the client SQL interface can be seen in the screenshot of Figure 2. The land parcels that satisfy the query are highlighted.

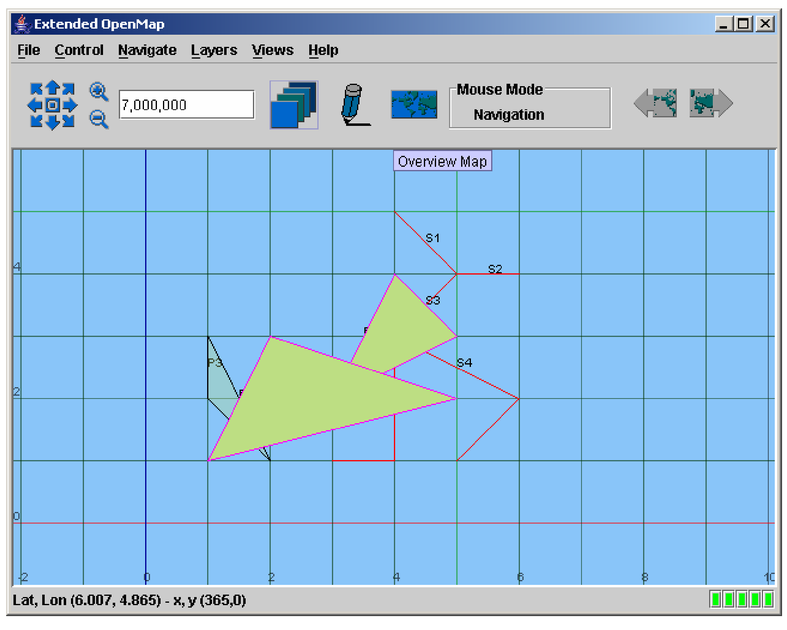

Figure 2. Query result data displayed in Java client

6. An example of using PostGIS data type with OpenMap™

The implementation of the PostGIS was similar, though more complex, than that for the PostgreSQL native types. The data set used covers all of Ireland at a scale of 1:250,000. Hence we do not describe the development or schemas, but present the mappings between the PostGIS data type, the JDBC data types, and the OpenMap™ data types. The mapping that were chosen are listed in Table 3.

PostGIS Geometric Data Type	PostGIS JDBC Extension Class	OpenMap OMGraphics Class
POINT	org.postgis.Geometry.Point	OMRaster
LINESTRING	org.postgis.Geometry.LineString	OMPoly (open)
POLYGON	org.postgis.Geometry.Polygon	OMPoly (closed)
MULTIPOINT	org.postgis.Geometry.MultiPoint	OMRaster (collection)
MULTILINESTRING	org.postgis.Geometry.MultiLineString	OMPoly (collection)
MULTIPOLYGON	org.postgis.Geometry.MultiPolygon	OMPoly (collection)

Table 3. Mapping PostGIS data types to OMGraphic classes

We provide just one sample query. Once the data is loaded into OpenMap™ the user can perform all the operations provided by OpenMap™. For example Figure 3 show the zoom operation.

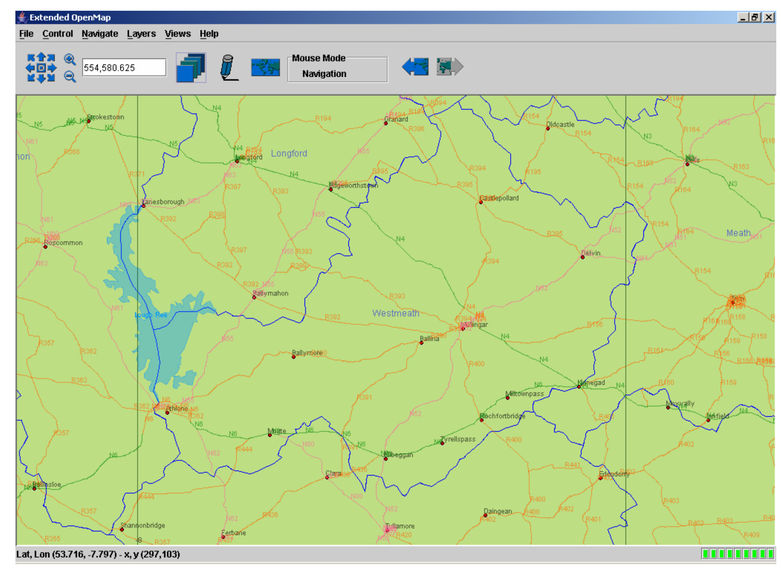

Figure 3. PostGIS client with data zoomed in on County Westmeath

Now we ask the system: "Display all counties that border Kildare". This query can be implemented using the following SQL command:

```
select c1.name as name,transform
(c1.the_geom,4326) as the_geom
from county c1,county c2
where
touches(c1.the_geom,c2.the_geom)
and
c2.name='Kildare';
```

The result of executing the above query can be seen in Figure 4, at an appropriate zoom level. The counties whose geometric attribute satisfy the query are highlighted.

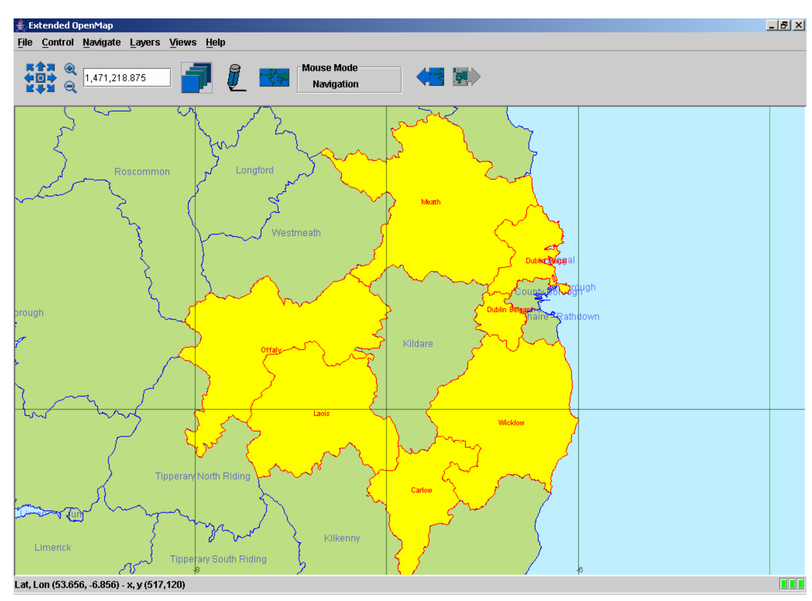

Figure 4. Query result displayed in PostGIS client.

7. Conclusion

The simple PostgreSQL client served as learning experience. It helped us understand the nuances of both OpenMap™ and the non-standard spatial JDBC extension. Obviously the development of the PostGIS client is of more value and of more interest in the long term. To the best of our knowledge no queryable OpenMap™ client exited for PostgreSQL/PostGIS before this project. We feel that the ability to send ad-hoc SQL queries to PostgreSQL/PostGIS and display the results is a useful contribution to the open source GIS movement. From our own research perspective the client allows us to examine other issues, such security and authentication within GIS. We must commend the quality of the open source products and the assistance that we received from the developers of those products. Obviously many improvements could be made, the deployment on the web would be interesting. We did not compare the functionality of our project to say Quantum GIS (QGIS 2006) which has PostgreSQL/PostGIS connectivity.

8. References

(QGIS 2006) Quantum GIS (2006): http://qgis.org

(OpenMAP 2004) BBN-Technologies. : http://openmap.bbn.com

(PostGIS 2004) Refractions Research: http://postgis.refractions.net

(PostgreSQL 2004): http://www.postgresql.org/

Philippe Rigaux, Michel Scholl, Agnès Voisard :Spatial Databases: With Application to GIS, Morgan Kaufmann, ISBN: 1558605886