TerraLib: Technology in Support of GIS Innovation

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

BIANCA MARIA PEDROSA[1], LÚBIA VINHAS[1]

ANTÔNIO MIGUEL VIEIRA MONTEIRO[1], JOÃO ARGEMIRO PAIVA[1]

MARCELO TILIO DE CARVALHO[2], MARCELO GATTASS[2]

[1] Instituto Nacional de Pesquisas Espaciais - INPE

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

{gilberto, cartaxo, bianca, lubia, miguel, miro}@dpi.inpe.br

[2] Catholic University of Rio de Janeiro

Rua Marquês de São Vicente 225, Rio de Janeiro, Brazil 22.453-900

{gattass, tilio}@tecgraf.puc-rio.br

Abstract. This work describes the development of a new GIS library (called TerraLib), that is aimed at providing a rich and powerful environment for the development of GIScience research. The motivation for this proposal is the current lack of either public or commercial GIS libraries that cater for the diversity of GIS data and algorithms, especially when viewed upon the latest advances in geographical information science. TerraLib is open source software, allowing a collaborative environment and its use for the development of multiple GIS tools.

Keywords. GIS, Spatial Analysis, Software Libraries.

1. Introduction

The last 20 years have seen dramatic developments in GIS technology and geographical information science. GIS software is now ubiquitous, and there are systems in different types and sizes, varying from the desktop to the corporate user and different solutions for Internet data access and distribution. In the vast majority of cases, such developments have been industry-driven, with few exceptions (such as the IDRISI and SPRING systems). Fierce competition and growing user demand has resulted in a number of high-quality solutions, which are largely responsible for the vast increase in the GIS marketplace.

However, the vast majority of the industry solutions is aimed at supporting basic needs of capture, archival and visualisation of spatial data. Recent technological advances have concentrated in issues such as user-friendly interfaces, interoperability across data repositories and spatial extensions of database technology. These developments have largely ignored recent advances in GIScience, which include research areas such as geostatistics (Goodvaerts, 1997), global and local spatial statistics (Getis and Ord, 1998), dynamic modelling and cellular automata (Couclelis, 1997; White and Engelen, 1997), heuristic search (Oppenshaw, 1998), environmental modelling (Burrough, 1998), point pattern analysis (Bailey and Gattrel, 1995), uncertainty assessment and modelling (Heuvelink, 1998; Felgueiras, 1999), spatial econometrics (Anselin, 1988) and neural networks for spatial data (Medeiros, 1999).

The authors posit that the geographical information community would benefit from the availability of a general, open source GIS library. This resource would make a positive impact by allowing researchers and solution developers access to wider range of tools than what is currently offered by the commercial companies. In a similar approach as the GNU and Linux efforts, such development does not happen by spontaneous growth: there has to be a core set of technologies in which further development can take place. Our proposal for the development of the **TerraLib** spatial library aims precisely at offering the GIS community a basis for further development.

The work is divided as follows. Section 2 indicates the general principles and of **TerraLib**. In Section 3, we present the main components of the library. In Section 4, we illustrate the programming environment of **TerraLib**, from a simple set of programs.

2. TerraLib Design Rationale

2.1 Generic Software Requirements

What sort of environment should a GIS library cater for? It should provide, first of all, support for the basic components of spatial data sets*: data translators, map representation, geometrical data structures and algorithms*. It also needs to support the establishment of *data models* for GIS data, without imposing strong constraints on their use. It also needs to provide *user interface* tools, which may be used for simple applications.

The need for perform efficient *data translation* is the single most important design consideration in TerraLib. Most users already have their data in an existing system and will want to perform analysis and exploration using techniques not available in their original system. There should be support for inclusion of tabular data, which has been assigned a geographical reference (as a census tract number or a zipcode).

Map representation and *cartographic projection* tools form a basic core of a GIS library. The library should support a basic core of cartographic projections and a set of techniques for reprojection and integration of spatial data. It should be very easy to add support for new projections and datum.

Algorithms form a basic core of most successful research efforts. In many GIS libraries, the misuse of object-oriented principles has resulted in classes that contain both the underlying data structure and the corresponding set of algorithms. In this case, the algorithms would be unecessarily linked to a particular type of data structure. Therefore, TerraLib algorithms are to designed as independent entities, which do not belong to a particular class.

The traditional *geometrical data structures* used for geographical data include vector data structures (points, lines, polygons, triangular meshes), raster structures (matrices) and relational structures (tables). Such structures also need underlying support for indexing structures such as R-trees.

In the case of *data models*, one of the important advances in GIScience in the 90's has been the widespread acceptance of a general conceptual data model for geographical data. In this model, geographical reality is represented as either fully definable entities (*features)* or smooth, continuous spatial variation (*fields)*. Although this simple dichotomy has been subject to objective criticism (Couclelis, 1992; Burrough and Frank, 1996), it has proven a useful frame of reference. It has adopted, with some variations, in the design of the latest generation of GIS technologies, such as SPRING (Câmara et al, 1996), Arc/Info-8 and OpenGIS.

We have designed TerraLib to allow different alternatives of *data models* to be implemented from the same software basis. Therefore, we emphasized a very loose coupling between the data model classes (on one side) and the algorithms and data structures (on the other side). Altough this de-coupling may lead to some duplication, it should be useful to be able to derive different applications based on TerraLib that might use the data model of different systems. For example, one developer may want to integrate TerraLib to existing systems such as GeoMedia or ARC/INFO-8. Another may develop OpenGIS-compliant applications.

2.2 Support for Innovative Research

The preceding discussion was centred in the support of conventional GIS applications. However, in order to be useful as a support for innovative research, **TerraLib** needs, from its conceptual base, to consider the needs of emerging GIS research areas, such as:

- Uncertainty modelling, where each field is coupled with information about spatial imprecision, which is propagated in map algebra operations (Heuvelink, 1998).
- Cellular automata, whose application to GIS requires that each cell has its own inherent set of attributes (as distinct from a single state) which represent its relevant physical, environmental, social, economic or institutional characteristics. (Couclelis, 1997; White and Engelen, 1997).
- Dynamic modelling (Burrough, 1998) requires the support for timers and interactive procedures.

- Applications such as Spatial Statistics and Spatial Interaction Models require the use of spatial proximity matrices for supporting spatial relations among elements.

2.3 Development Strategy

Since **TerraLib** was designed to support an open, collaborative development environment, some basic principles are in order:

1. The interface for each class should be kept as *minimal* as possible. The introduction of new algorithms and tools should not affect already-existing code, include keeping *include* files intact.
2. The implementation of the geometrical data structures is kept completely separate from the programming interface, using the "pimpl" idiom (Sutter, 2000).
3. There should be a maximal degree of *ortogonality* between the components of the library, and they are designed to be used *independently*.

We are not committed to a single programming style, but use the ideas of multi-paradigm programming, which advocates the combination of different techniques such as object-oriented, algorithmic and generic programming [Coplien, 1999]. This technique is particularly suited to the C++ language [Stroustroup, 1997].

Reuse considerations have also led the authors to choose C++ as the development language. We hope to adapt algorithms that have been developed for **SPRING** (Câmara et al, 1996).

3 Software Structure

3.1 Main Components

TerraLib is developed as a multi-tier library, and its first version includes:

- At the higher level, it provides abstract data classes to manage and represent geographical information, based on a field/object paradigm, viz.: `Feature, Network, Surface, CellArray`.
- At the intermediate level, it provides support for *geometrical data structures, data formats, map representation* and *algorithms*.
- At its lowest level, it provides spatial indexing structures for efficient handling of large data sets.

3.2 Geometrical and Attribute Structures

The *data structures* include:

- `TeCoord2D`[1]: a 2D coordinate.
- `TeLine`: a vector of 2D coordinates, that can be associated to a height.
- `TeLineSet`: a set of lines.
- `TePolygon`: composed of Lines and of other Polygons (its children).
- `TePolygonSet`: a set of polygons.
- `TePointSet`: a set of 2D samples.
- `TeTIN`: a triangular mesh.
- `TeGrid`: a raster data structure (used for images and grids).
- `TeTable`: an attribute table used for linking to relational DBMSs

Each instance of a data structure has a unique identifier, to enable its linking and storage in a DBMS. We illustrate some of these geometrical structures and its relationships in Figure 1.

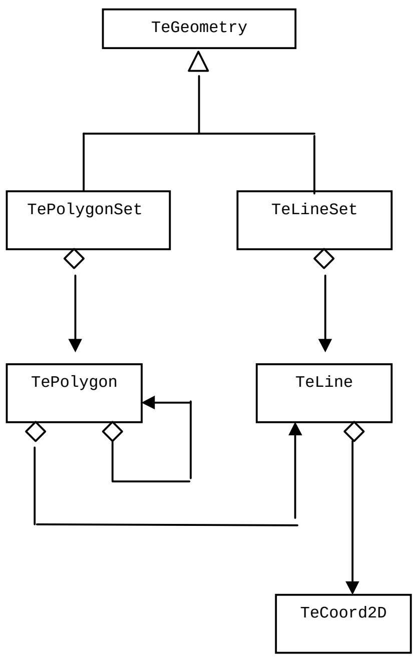

Figure 1 – Polygon and Line vector representation

[1] Throughout the text, TerraLib classes and functions are indicated with a `Te-` prefix.

As can be seen in the Figure 1, the `TePolygonSet` class is a composite of `TePolygon`. A `TePolygon` is a composition of `TeLine` and/or other `TePolygons` (its children). A `TeLineSet` is a composition of `TeLines` that are a composition of `TeCoord2D`.

A basic pattern for use in GIS geometrical structures is the Composite pattern (Gamma et al, 1996). As the Composite pattern can be used in another components of a GIS we propose a parameterised implementation of it (shown partially below):

```
template <class T>
class TeGeomComposite
{

public:

typedef vector<T> Components;
typedef Iterator<T> TeGeomIterator;

// -- Contructors
TeGeomComposite(){}

// -- Destructor
virtual ~TeGeomComposite();

// -- Methods
void Add ( const T& component );
int Size();
T& Next() const ;
T* First();
T* Last();

protected:
Components components_;
};
```

When considering the implementation of the geometry classes, we should allow for different alternatives to be tried and tested for efficiency and convenience. For example, a `TePolygonSet` can be represented as a graphical file or might be integrated into a relational database, with one polygon per tuple, in a similar fashion as the Simple Feature definition of the OpenGIS consortium (Open GIS, 1999).

To allow for different implementations of the Polygon class to be defined, we have used the so-called "pimpl" idiom (Sutter, 2000). This programming idiom proposes a separation between a class and its implementation, by using an opaque pointer to hide the implementation details.

In the case of Polygons, a special case of the Composite pattern arises: a Polygon is composed of Lines and of other Polygons (its children), as it can be seen below.

```
class TePolygon
{
public:
    // public members ...
private:
    struct TePolygonImpl *pimpl;
};
// sample implementation
// there could be alternatives
struct TePolygonImpl
{
class TeLine2DSet: public
        TeGeomComposite<TeLine2D> {}
class TeChildSet: public
       TeGeomComposite<TePolygon> {}

    TeLine2DSet  lines_;
    TeChildSet children_;
};
```

Therefore, the implementation of the geometrical data structures has required a combination of Design Patterns and Generic Programming paradigms.

3.3 Algorithms and Iterators

For Terralib algorithms, we have adopted the principles of *generic programming*: "*decide which algorithms you want; parametrize them so they work for a variety of suitable types and data structures*" (Stroustroup, 1997). Following the example of the STL library, which is now part of the C++ standard, we propose the use of the *iterator* concept, as a basis for . Iterators are a generalisation of the idea of pointers, and are used in the STL to separate the containers from the algorithm [Austern, 1998].

The algorithms are defined in terms of different types of iterators of spatial data types, such as: `TePolygonIterators`, `TeLineIterators`, `TePointIterators`, and `TeTableIterators`.

In order to illustrate the concept, we present the case of a line simplification algorithm. Such an algorithm can be applied in the case of a set of lines, as well as in the case of a set of polygons. We would like to design such a function as independent entity, which does not belong to a particular class, as shown below.

```
void TeSimplifyLines (
  TeLineIterator begin,
  TeLineIterator end ) ;
int
main()
{
  TeLineSet ls;
  TeImportE00(ls, "lines.e00");
  TeSimplifyLines (ls.FirstLine(),
                ls.LastLine());

  TePolygonSet ps;
  TeImportE00(ps, "polygons.e00");
  TeSimplifyLines (ps.FirstLine(),
                ps.LastLine());
//....
return 0;
}
```

The parameters for the `SimplifyLines` function are `TeLineIterators`. The `SimplifyLines` algorithm doesn't need to assume anything beyond minimal functionality guaranteed by the `TeLineIterator`, which include functions to have access to a `TeLineSet` by traversing it in some order.

Note that both LineSet and PolygonSet classes need to provide method that return `TeLineInterators`. The geometric structures are thus responsible for providing the Iterators that are meaningful for the use of the algorithms.

3.4 User Interfaces and Visualisation

Strictly speaking, user interfaces and visualisation are not an essential part of TerraLib. Our emphasis is an efficient and flexible set of data structures and algorithms, allowing different applications to produce their own interfaces or the linking of TerraLib programs to existing environments.

However, in many cases, users may want a simple visualisation and user interface environment that allows for rapid prototyping of ideas and new concepts. For those purposes, we provide a set of GUI and visualisation classes, which are based on the **Qt** public-domain software library (Troll Tech, 2000).

The `TeApplication` class provides a simple interface for visualisation of GIS data. It provides a method `Run()` for creating an event loop, much similar to the operation of the Motif and MFC toolkits. The method `Show()` instructs the user interface to display a data, which may be a simple geometrical structure or a more complex layer.

3.5 Support for Innovative Research

The applications described in Section 2.2 have motivated some design decisions:

- In TerraLib, each field of `Surface` type is associated with an `Uncertainty` information. This field is part of the representation of the surface and access to it is available for error propagation functions.
- A new type of field, a `CellArray` type, is introduced to handle multiple attributes for a single cell, and make it easier to define transitional rules.
- Support for a `Graph` geometrical data structure is included, to allow for both spatial interaction algorithms and different alternatives for proximity measures (O'Sullivan, 1999).

4. Programming in TerraLib

4.1 Hello, GIS World !

We consider that the best description of GIS library is achieved by showing how it should be used in practice. Thus, we present a set of programs, starting from simple examples, which illustrate the principles and practice of programming in **TerraLib**. This description assumes a familiarity with the basics of GIS data structures and algorithms, as well as with the general programming in C++. Let us start by considering a problem: what is the simplest GIS program that can be written?

```
#include <teapplication.h>
#include <tegeometry.h>
#include <tedataconversion.h>
int main ()
{
    TeApplication app;
    TePolygonSet   ps;
    TeImportShape (ps, "BR.shp" );
    app.Show (ps);
    app.Run();
}
```

This program reads a file containing a data set, in "shapefiles" format, and displays it. The first line of this program creates an instance of the `TeApplication` class, which is responsible for providing a simple interface for visualisation of GIS data. Since TerraLib is designed to be standalone, the use of this class is optional. The visualisation procedures associated to TerraLib are described in the "User Interface and Visualisation" section.

The next line indicates a data conversion procedure (in this case, an ARC/View *shapefile* containing the co-ordinates of the counties of Brazil, or *municipios* in Portuguese). TerraLib provides efficient data translation tools for formats such as SHP, MIF and E00. Note that we have defined `TeImportShape` as a function rather than a method for a `TeGeometry` class in keeping with *principle* 1 (minimal interfaces).

The next line simply indicates the existence of a method `Show()`, associated to the `TeApplication` class, which requests to the user interface the display of data. The last line indicates that the "main loop" of the `TeApplication` class is called. This command creates a window interface.

4.2 Simple Geometrical Algorithms

The next step is to apply some simple geometrical algorithms. For example, let us suppose we want to read a data set containing the counties of Brazil as a set of polygons, and we would like to generate a Voronoi diagram from the centroids of these polygons. This is done as follows:

```
#include <tegeometry.h>
#include <tedataconvert.h>
#include <tealgorithm.h>
int main ()
{
    TePolygonSet ps;
    TeImportShape( ps, "Br.shp");

    TePointSet pt;

    TeGenerateCentroids
    ( ps.FirstPoint(),
      ps.LastPoint(),
      pt.FirstPoint() );

    TePolygonSet voron;
    TeVoronoi (
    pt.FirstPoint(),
    pt.LastPoint,
    voron.FirstPoint() );

    TeExportGBR(voron, "Vor.gbr");
}
```

This program creates a `TePointSet` from an existing `TePolygonSet` (read from a SHP file), and generates a Voronoi diagram with the `TeVoronoi` function. The result is exported to a file in the GEOBR data format. Note the use of iterators as interfaces to the algorithms.

4.3 Data Management

In the preceding examples, there is no abstract modelling of this data set, no control organisation is in place, and there are no attributes associated to the geometrical structures. If we want data management and to associate attributes to the geometrical structures, we need to introduce the concept of a geographical database (or `Geodatabase` for short). The Geodatabase class supports archival of both geometrical and descriptive parts of GIS data set. Additionally, it uses existing relational data base technological solutions (such as MySQL and ODBC) as basis for management of tables.

```
#include <teapplication.h>
#include <tegeodatabase.h>
#include <telayer.h>
#include <tegeometry.h>
int main ()
{
    TeApplication app;
    TeGeoDataBase db;
    db.Open ("World");
    TePolygonSet ps;
    TeImport (ps, SHP, "BR.shp" );

    TeTable t;
    db.ImportTable(t,"CA","Br.dbf");

    TeLayer br;
    db.NewFeature(br, "Br", ps);
    db.Associate (br, t, "IBGE_ID");

    app.Show ( br );
    app.Run();
}
```

The above program requires the instantiation of a `GeoDatabase`, which is responsible for data management and handling attribute information. In this program, a higher-level structure (a `Layer` consisting of `Features`) is created from lower-level structures (a `PolygonSet` and a `Table`), in four steps:

- a geometry is created from existing data;
- a table is created from an external source;
- a new layer is instantiated, from an existing geometry;
- the attributes and the geometry are associated, by means of one attribute which indicates a spatial index.

This procedure allows for the association of many attribute sets to the same geometry (and vice-versa).

These examples show that even the simplest GIS application needs some procedures for data management and abstract modelling. Otherwise, the library user is left with a substantial burden to construct higher-level structures. They also show the application of our design rationale in some simple, but significant examples.

5. Conclusion

This paper outlines the rationale for the initial version of TerraLib, an open source software library aimed at offering the GIScience community a basis for shared development. This proposal is based on more than 15 years experience in GIS software development by the groups of the of Brazil's National Institute for Space Research (INPE) and the Catholic University of Rio de Janeiro. By making a version of the proposal available for public discussion, we hope to attract partners for long-term co-operative partnerships.

Acknowledgements

This work has been partially financed by a joint NSF/CNPq project on "Interoperabilty on GIS" (process CNPq 480322/99).

References

ANSELIN, L. 1988. *Spatial Econometrics, Methods And Models*. Dordrecht: Kluwer Academic.

ANSELIN, L. 1998. Interactive techniques and exploratory spatial data analysis. In: P. Longley, M. Goodchild, D. Maguire and D. Rhind (eds*.)*, *Geographical Information Systems: principles, techniques, management and applications*, pp. 251–264. New York: Wiley.

AUSTERN, M.H., 1999. *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*. Reading, Addison-Wesley, 1999.

BAILEY,T.; GATTRELL, A., 1995. *Spatial Data Analysis by Example*. London, Longman.

BURROUGH, P.; FRANK, A., 1996. (eds) *Geographic Obejcts with Indeterminate Boundaries*. London, Taylor and Francis.

BURROUGH, P, 1998. "Dinamic Modelling and GIS". In: Longley, P., Brooks, S., McDonnell, R., Macmillan, B. (eds), *"Geocomputation: A Primer"*. New York, John Wiley & Sons.

CÂMARA, G.; SOUZA, R.C.M.; FREITAS, U.M.; GARRIDO, J.C.P., 1996 "SPRING: Integrating Remote Sensing and GIS with Object-Oriented Data Modelling". *Computers and Graphics*, vol.15 , n.6, pp.13-22.

CÂMARA,G.; SOUZA, R.C.M.; MONTEIRO, M.V.; PAIVA, J.; GARRIDO, J., 1999. "Handling Complexity in GIS Interface Design". In: *Proceedings of the I Brazilian Workshop on GeoInformatics*, Campinas, São Paulo. www.dpi.inpe.br/geoinfo99.

COPLIEN, J. *Multi-Paradigm Design for C++*. Reading, Addison-Wesley, 1999.

COUCLELIS, H., 1992. "People Manipulate Objects (but Cultivate Fields): Beyond the Raster-Vector Debate in GIS". In: Frank, A.; Campari, I. and Fomentini, U. (eds) *Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pp. 65-77. Berlin, Springer.

COUCLELIS, H., 1997, "From cellular automata to urban models: new principles for model development and implementation", *Environment and Planning B: Planning & Design*, 24, 165-174.

DEUTSCH, C.; JOURNEL, A. 1998. *GSLIB: Geostatistical Software Library and User's Guide*. New York, Oxford Univeristy Press.

FELGUEIRAS, C. 1999. "Modelagem Ambiental com Tratamento de Incertezas em Sistemas de Informação Geográfica: O Paradigma Geoestatístico por Indicação". PhD Thesis in Computer Science, INPE (in Portuguese).

GETIS, A., ORD J. K., 1996. "Local spatial statistics: an overview". In: *Spatial Analysis: Modelling in a GIS Environment*. (LONGLEY,P.; BATTY,M., eds), pp. 261-277. New York, John Wiley.

GOOVAERTS, P. 1997. *Geostatistics for Natural Resources Evaluation*. New York, Oxford University Press.

HEUVELINK, G. 1998. *Error Propagation in Environmental Modelling with GIS*. London, Taylor and Francis.

LEONDES,C. (ed), 1997. *Image Processing and Pattern Recognition (Neural Network Systems Techniques and Applications Series, Vol 5)*. New York, Academic Press.

LONGLEY, P., 1998. (ed) *Geocomputation: A Primer*. New York, John Wiley and Sons, 1998.

MEDEIROS, J.S., 1999. *Geographical Databases and Artificial Neural Networks: Technologies in Support of Land Management*. PhD Thesis in Geography, University of São Paulo (in Portuguese).

OPENSHAW, S., 1998. “Building automated Geographical Analysis and Exploration Machines”. In: *Geocomputation: A primer* (Longley, P. A., Brooks, S. M. and Mcdonnell, B. (eds)), p. 95-115. Chichester, Macmillan Wiley.

O'SULLIVAN, D. 1999. Exploring the structure of space: towards geo-computational theory. In: *Proc. IV International Conference on GeoComputation*. Mary Washington College, USA. `<www.geovista.psu.edu/geocomp/geocomp99>.`

STROUSTRUP, B. 1997. *The C++ Programming Language*. Reading, Addison-Wesley.

SUTTER,H., 2000. *Exceptional C++*. Reading, Addison-Wesley.

TROLLTECH, 2000. *The Qt Interface Library*. www.trolltech.com.

WHITE, R.; ENGELEN, G., 1997 ‘Cellular Automata as the Basis of Integrated Dynamic Regional Modelling’ *Environment and Planning B*, Vol.24, pp.235-246.