

# Coverage representation in TerraLib

Vitor Dantas, Marcelo G. Metello, Melissa Lemos, Marco A. Casanova

Tecgraf – Computer Graphics Technology Group  
Pontifical Catholic University of Rio de Janeiro (PUC-Rio)  
Rua Marquês de São Vicente, 225 – CEP 22.453-900 – Rio de Janeiro – RJ – Brazil  
{vitorcd,metello,melissa,casanova}@tecgraf.puc-rio.br

**Abstract.** Coverage representations, as defined by the OGC specifications, are useful for representing a wide range of geographic phenomena. However, in most GIS projects, coverage representations do not show a common interface and the line between coverages and features with geometry is unclear. We propose an extension to the TerraLib library, using a unifying approach to manage coverage representations, including performance considerations as a major issue.

## 1. Introduction

The OGC specification introduces two basic ways of modeling geospatial features, namely *features with geometry* and *coverages* [OGC 1999]. While the former emphasizes the identity of each single feature, the later emphasizes the whole picture, evidencing relationships and the spatial distribution of earth phenomena. The decision of which one is best depends on the application, since moving from one concept to the other is possible.

A coverage defines a function that maps some *spatial domain* into a *value set*. In general, the specialization of coverages reduces to the specialization of the spatial domain. For instance, a Discrete Point Coverage is one whose spatial domain is a set of discrete points. Figure 1 shows an example of a Discrete Point Coverage whose value set has two dimensions, so that each point is associated with a temperature value and a wind speed value, i.e., to a value vector  $(t_i, s_i)$ ,  $1 \leq i \leq 6$ .

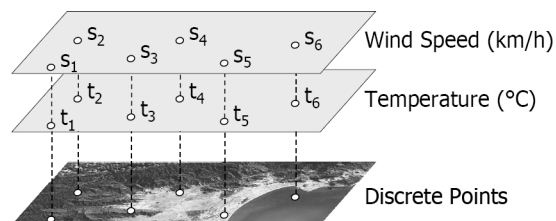


Figure 1. Discrete Point Coverage with two dimensions.

In this work, we aim at developing representations for several coverage subtypes – such as Discrete Point Coverage, Surface Coverage and Line String Coverage – and at defining a common abstract representation for them, as an extension of the TerraLib library [Câmara et al. 2000]. TerraLib is an open-source project that provides tools for the development of geographic information systems [Vinhas and Ferreira 2005].

One often finds in the literature other designations for the concept of coverage. The relation between features with geometry and coverages is similar to what has been discussed about *discrete objects* and *continuous fields* [Couclelis 1992], or *geo-objects* and *geo-fields* [Vinhas 2006].

Although most GIS projects address the problem of coverage representation, few establish a clear distinction between the concepts of features with geometry and coverages. Typical coverage representations such as regular grids (raster) are included in several projects, including GDAL, GRASS, OSSIM GMT and DeeGree [Ramsey 2007].

TerraLib currently offers implementations of coverage representations such as triangular irregular networks (TINs) and regular grids (raster) [Vinhas and Souza 2005], but all representations in TerraLib are referenced as geo-objects. Rather than replacing the current implementation for these representations, we propose a common interface for accessing them.

The paper is organized as follows. Section 2 discusses the representation of coverages in general. Section 3 presents the current implementation of Discrete Point Coverages. Section 4 discusses future work and conclusions.

## **2. Coverage Representation**

### **2.1. General requirements**

We propose the following high-level requirements for coverage representations:

1. The data structure must represent a set of geometries (the spatial domain of the coverage).
2. Each stored geometry must be associated with a value vector (the value of the coverage for the spatial region that the geometry specifies).
3. There must be support for queries that fetch the value vector associated with each stored geometry.
4. There must be support for queries that fetch the value vector in any location within the convex hull defined by the stored geometries, using an interpolation method. There must be a flexible way to change the interpolation method in use.
5. The persistent representation of geometries (in a database) must be clustered by spatial proximity to improve the performance of query processing.

Requirements 1 and 2 have to do with how to represent the coverage; requirements 3 and 4 with how to query the coverage; and requirement 5 is non-functional. Requirements 1 and 3 comply entirely with the OGC specification. We introduced requirements 4 and 5 based on what we expect to be generic application requirements.

Requirement 4 was included to provide a continuous view of the coverage, despite the fact that the support data structure is discrete (i.e., a set of geometries). Interpolation methods are used to infer the value vector at positions where it is undefined, as recommended in the discussion about discrete and continuous coverages of the OGC specification.

Requirement 5, in particular, addresses performance issues that are not considered by OGC. Clustering geometries lead to performance improvement because loading a few large chunks of data from a database is faster than loading the same amount of data in several small chunks. Besides that, if the geometries are clustered by spatial proximity, the addition of a caching mechanism will result in less frequent access to the database in a typical GIS application, in which it is reasonable to assume that access to data over time will be done by spatial proximity, e.g. a user looking at a location is more likely to move next to nearby locations.

## 2.2. High-level interface for coverage access

For the discrete setting, the OGC specification on coverages provides an abstract specification for the coverage function, named `DiscreteC_Function`, showed in Figure 2. The method `num` returns the number of geometries in the domain, while methods `domain` and `values` return an exhaustive list of the domain (geometries) and the range (value vectors) of the coverage function. The method `evaluate` is used to fetch the value vector associated with a specific element of the domain (a geometry).

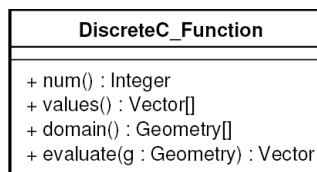


Figure 2. Discrete coverage function from the OGC abstract specification.

We follow a similar approach to define an interface for coverages in the TerraLib library, as depicted by Figure 3. This interface provides methods `begin` and `end`, which have two optional parameters of types `TePolygon` and `TeSpatialRelation`, and return objects of type `TeCoverage::iterator`. These two methods, together, play the role of the method `domain` in `DiscreteC_Function`.

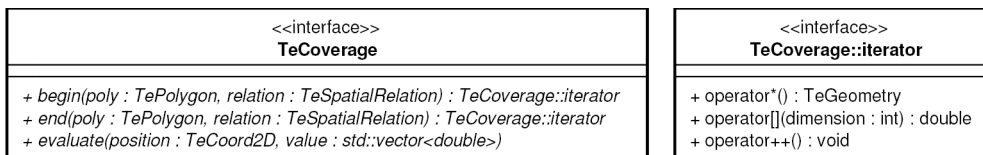


Figure 3. High-level interface for coverages in the TerraLib library.

The parameter of type `TePolygon` represents a polygon and defines a selection area. If no polygon is given, then the selection area includes the entire spatial domain. The parameter of type `TeSpatialRelation` indicates the kind of the spatial relation (e.g., intersection, crossing, overlapping) that holds between the selection area and the geometries to be selected. If no value is given, the relation defaults to intersection.

The returned iterators are used to traverse the selected geometries (instances of `TeGeometry`) and the values associated to them. Coverage iterators, represented by instances of `TeCoverage::iterator`, are manipulated using the overloaded operators “\*”, “[” and “++”. Dereferencing a coverage iterator with the “\*” operator outputs a geometry, while the operator “[” is used to get the value of a specific dimension. The increment operator “++” is used to advance to the next position of the iteration, i.e. to

the next selected geometry. The example code in Figure 4 shows how to use coverage iterators, assuming that we have access to an instance of `TeCoverage` named “c”.

```
// Make spatial query
TeCoverage::iterator it = c.begin(poly);
TeCoverage::iterator end = c.end(poly);

// Iterate over selected geometries
while (it != end) {
    TeGeometry& geom = *it;
    double value1 = it[1];
    double value2 = it[2];
}
```

**Figure 4. Example code, showing how to use coverage iterators.**

The method `evaluate` provided by the interface `TeCoverage` extends the domain of the coverage beyond the discrete setting, because one is able to access the values on arbitrary locations (through the use of interpolation methods). Otherwise, the domain would be restricted to the discrete collection of geometries, as the method `evaluate` in `DiscreteC_Function`.

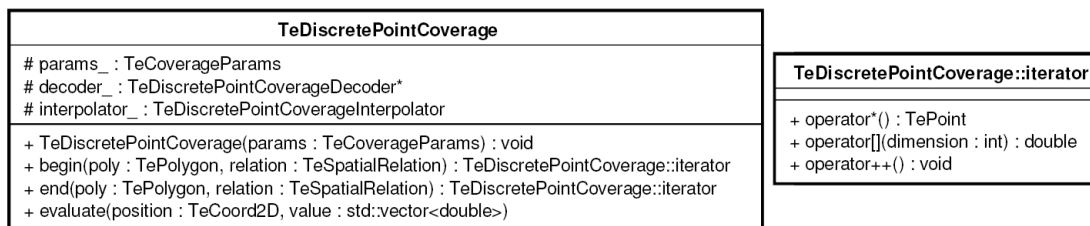
### 3. Current implementation

The first focus of our work was the definition of a representation for the simplest subtype of coverage, namely a Discrete Point Coverage. This representation assumes that the domain is a finite set of sample points. It is useful to model features such as temperature, as this kind of information is typically collected at stations distributed over the territory.

This section discusses a working implementation for a Discrete Point Coverage representation in TerraLib, including class diagrams and the database model.

#### 3.1. Current class model

`TeDiscretePointCoverage` is the main class of the Discrete Point Coverage representation, following the TerraLib naming convention. Figure 5 shows this class, omitting several attributes and methods for clarity.



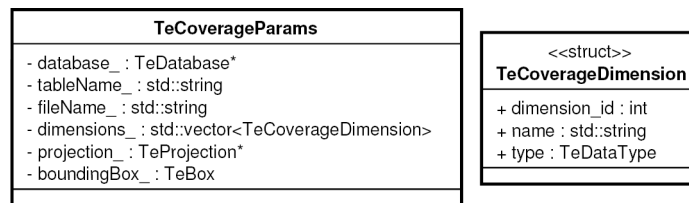
**Figure 5. Class diagram for a Discrete Point Coverage.**

The methods `begin` and `end` are equal to those from class `TeCoverage`, in section 2.2, except for the return of type `TeDiscretePointCoverage::iterator`, that is specialized to traverse discrete points (instances of `TePoint`).

The method `evaluate` can be applied to a position that does not belong to the discrete domain of the representation. This method delivers the call to the interpolator (instance of `TeDiscretePointCoverageInterpolator`), a class member which actually

implements interpolation. The interpolator class can be extended for implementing different kinds of interpolation, making it a flexibility point. If no interpolator object is provided by the application, the default implementation (*nearest neighbour*) is used.

The constructor of this class receives an instance of `TeCoverageParams`, which represents coverage parameters, and include information about where the data is stored (in a file or database) so that, after an instance of `TeDiscretePointCoverage` is constructed, one can start submitting spatial queries to the coverage. Figure 6 shows the class `TeCoverageParams`, along with the auxiliary data structure `TeCoverageDimension`. Methods were omitted for clarity.

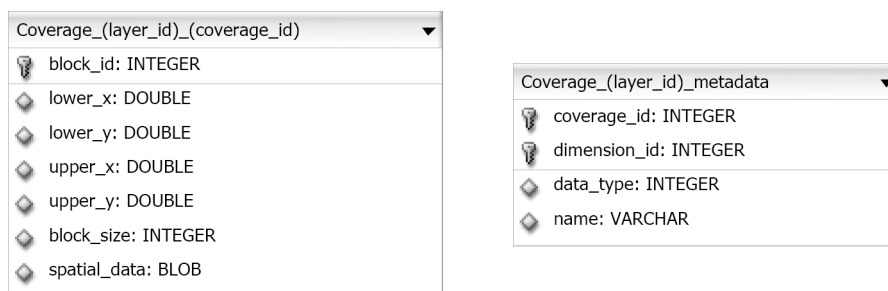


**Figure 6. Class diagrams representing parameters of coverages.**

An instance of `TeCoverageParams` contains information about how to access the persistent coverage representation from the database or from a file, what are the dimensions of the value vector, what is the minimum rectangle that contains all geometries of the coverage, what is the projection used and more.

### 3.2. Current data model

The coverage data stored in the database, including information about geometries and values associated to them, is clustered by spatial proximity of geometries. The clustering process creates blocks, which are stored in the coverage table. The name of such a table has the form `Coverage_(layer_id)_(coverage_id)`, where `(layer_id)` means the layer identifier and `(coverage_id)` means the coverage identifier. Figure 7 shows the diagram of a coverage table.



**Figure 7. Diagrams for coverage tables and coverage metadata tables.**

The primary key of this table is the block identifier, named `block_id`. For each stored block, there is also information about the minimum bounding box of all its geometries in the fields `lower_x`, `lower_y`, `upper_x` and `upper_y`, and information about its total size (in bytes) in the field `block_size`. The block contents are stored in a BLOB field named `spatial_data`, which contains a serialization of the geometries and values.

To decode the raw data stored in registers, the application needs to know the structure of the value vector of each coverage, i.e., what are the dimensions and the data

types used. For that matter, a metadata table is available, containing this kind of information. There is a single metadata table for all coverages in the same layer.

The primary key of this table is composed by the coverage identifier, named `coverage_id` and the dimension identifier, named `dimension_id`. The field `data_type` contains information about the data type of this dimension and the field `name` contains a name for the dimension (e.g., “Temperature”, “Wind speed”). The name of this table has the form `Coverage_(layer_id)_metadata`, where `(layer_id)` means the layer identifier. Figure 7 shows the diagram of a coverage metadata table.

#### 4. Conclusions and future work

We addressed in this paper the problem of formalizing coverage representations in the context of the TerraLib project. We described a common interface for all coverage types and implemented the simplest coverage representation, namely a Discrete Point Coverage. We started from the OGC abstract specification, but we also included requirements that addressed performance and other non-functional questions.

Future work includes the implementation of other coverage representations, such as Surface Coverages, along with adapting representations that are already part of TerraLib so that all implement a common interface. A memory caching mechanism, common for all coverage representations, is also a major issue, considering the performance of processing large amounts of data. We also want to make tests in applications that are built with TDK [TDK 2007] in order to acquire empirical evidence that *coverages* and *features with geometry* should be treated separately for better performance. TDK is an API that provides components to make it easier to implement GIS applications using Terralib.

#### 5. References

- Câmara, G. et al. (2000) "TerraLib: Technology in Support of GIS Innovation", II Workshop Brasileiro de Geoinformática, Caxambu, Brazil.
- Couclelis, H. (1992) "People manipulate objects (but cultivate fields)", Proceedings of International Conference on GIS, Pisa, Italy.
- OGC (2000) "The OpenGIS Abstract Specification - Topic 6: The Coverage Type and its Subtypes", version 6.0, Open Geospatial Consortium Inc, USA.
- OGC (1999) "The OpenGIS Abstract Specification - Topic 5: Features", version 4.0, Open Geospatial Consortium Inc, USA.
- Ramsey, P. (2007) "The State of Open Source GIS", Refrations Research Inc, Canada.
- TDK (2007) "Terralib Development Kit", <http://www.tecgraf.puc-rio.br/tdk>.
- Vinhas, L. (2006) "Um subsistema extensível para o armazenamento de geo-campos em bancos de dados geográficos", PhD Thesis, INPE, São José dos Campos, Brazil (in Portuguese).
- Vinhas, L. and Ferreira, K. R. (2005) "Descrição da TerraLib", In "Bancos de Dados Geográficos", Edited by M. Casanova et al, Editora MundoGEO, Brazil (in Portuguese).
- Vinhas, L. and Souza, R. C. M. (2005) "Tratamento de dados matriciais na TerraLib", In "Bancos de Dados Geográficos", Edited by M. Casanova et al, Editora MundoGEO, Brazil (in Portuguese).