Continuous Interaction with TDK: Improving the User Experience in Terralib

Marcelo Metello, Mário de Sá Vera, Melissa Lemos, Leone Pereira Masiero, Marcelo Tilio Monteiro de Carvalho

Tecgraf—Computer Graphics Technology Group,
Pontifical Catholic University of Rio de Janeiro,
Rua Marquês de São Vicente 225, Gávea, Rio de Janeiro, RJ 22453-900, Brazil

{metello,mvera,melissa,leone,tilio}@tecgraf.puc-rio.br

Abstract. *Historically, visual display has always played a very important role in GIS applications. However, visual exploration tools do not scale well when applied to huge spatial data sets. More recently, faster processing hardware and more sophisticated computer graphics has been used to improve user experience for geospatial data visualization. Some applications have recently introduced the concept that the user should be able to navigate through the data in a smooth and continuous way without being blocked waiting for data to be loaded, even if this means seeing incomplete data for some time. Google Earth and NASA World Wind are recent examples of this concept. This paper describes an architecture incorporated in the TDK (TerraLib Development Kit) open source API to provide support for applications that want this kind of user interaction with Terralib geographical data.*

1. Introduction

In a Geographic Information System (GIS) data visualization plays a fundamental role in the system user experience. From a location based application to a more sophisticated spatial data manipulation graphic tool the user wants to have a pleasant interaction with the data visualization interface in place.

If we consider the dualism of GIS data models where thematic and spatial data are together [Oosterom 1988, Longley et al 2001] we can understand the particularity of the user visual experience in a GIS scenario. The spatial data component requires a special paradigm for visualization in order to give the user a realistic interaction with the system.

Another consideration is the requirement for an efficient processing of data for a pleasant user experience. In terms of data scaling factors GIS definitely falls into one of major high scale heavy databases. So while visualizing a GIS database information the data requests are not as easy to be addressed as in the majority of other fields.

For the past several decades the user experience with GIS data visualization has been restricted to a simple “geographic map like” paradigm. In this legacy model the user navigates over the data by panning over the data with the visualization window. This user interface model was inherited from the standards brought by modern

Graphical User Interfaces solutions [Mesa 1998] from the 80s. The inventors of this GUI paradigm were definitely not aware of the particularities involved in a GIS data visualization application. As a consequence of this non predictable usage of the standard two dimensional window oriented paradigm the GIS data visualization suffers of several potential drawbacks: (i) the data navigation as a blocking prone experience; (ii) the system is not responsive all the time.

One major evolution experienced by the game development technology has been the usage of computer graphics techniques to give the user a navigation experience more realistic. These techniques have been adopted in the GIS community by systems such as Google Earth [Google 2007] and World Wind [NASA 2007] to improve user experience. These systems have implemented what we define here as *continuous interaction* as opposed to *discrete interaction.*

By *discrete interaction* we mean that the user clearly sees that the visualization area "jumps" from one point to another which is perceptively away. On the other side we call *continuous interaction* [Faconti et al 2000, Smith et al 1999] when the user sees the visualization area moving in a smooth way so that the user does not perceive any discontinuity in the visualization area trajectory. For a truly rich navigation experience we consider the use of *continuous interaction* fundamental.

The objective of this work was to provide the open source project TDK (TerraLib Development Kit) [Tilio and Vera 2005, TDK 2007] with support for continuous interaction. TDK is an API build on top of Terralib [Câmara et al 2000] with the purpose of providing components and services to make it easier to implement GIS applications using Terralib. Terralib is a GIS C++ library, available from the Internet as open source, devoted to the development of multiple GIS tools. Its main aim is to enable the development of a new generation of GIS applications, based on the technological advances on spatial databases. The architecture presented here was implemented and incorporated into TDK (in version 2.1).

We followed two strategies to reach the objective: (i) development of a spatial data responsive cache using spatial access methods for taking the spatial proximity in consideration for efficiency and fast data feedback; (ii) absorption of Computer Graphics high performance navigation techniques applied currently to flight simulators and gaming in general. These techniques aim to make full use of all the resources available in the Graphics Processing Unit (GPU) of modern personal computers.

The GPU is a single-chip processor and dedicated graphics rendering device used in a personal computer or game console. Modern GPU's such as the ATI Radeon [ATI 2002] and GeForce [nVIDIA 2002] are designed to handle well 3D computer graphics and since they are optimized for graphics, they make use of parallelization and are extremely fast for operations such as texture mapping, rendering triangles, color operations and interpolation, coordinate system transformation and vector and matrix operations.

The contributions of this paper lie in exploring how data preprocessing and data storage in cache in a format close to the one required by the graphics toolkit (which is *OpenGL* [OpenGL 2007] in this project) may become a viable strategy for rendering maps quickly and make the whole system more user-responsive. We also describe an implementation effort that was carried out to test the ideas described in this paper and prove that it is possible to provide *continuous interaction* with data provided by Terralib.

The paper is organized as follows. Section 2 describes in detail the system architecture proposed in the paper. Section 3 discusses experimental results obtained with an implementation of the system. Finally, Section 4 contains conclusions and future work.

2. System Architecture

This section shall begin with some definitions followed by the description of all the steps of the rendering process and finally presents the general system architecture.

2.1 Definitions

Terralib defines the concept of *theme*, which was absorbed by *TDK*. In the context of this paper, a *theme* is a set of geographic objects of the same nature plus all style information needed to render each object. Each object has its own geometries (such as points, lines and polygons) and alphanumeric attributes. A visual style can be defined for all objects in a theme, for a group of objects (based on conditions on their attributes) or for an individual object. Every *theme* will have all its objects grouped into *data blocks* by spatial proximity. It allows spatial indexing with more efficiency as compared to treating each object individually. We also call *map* the image created by rendering a vector of themes in a given visualization window.

2.2 The Rendering Process

The Figure 1 shows the dataflow process that begins with the geographic data stored initially in a *Terralib* database. This data is requested according to *map* rendering demands. When loaded, every piece of data is preprocessed into a format friendly to the graphics toolkit. After that the preprocessed data is stored in a visualization cache to be used by the rendering routines and sent to the GPU.

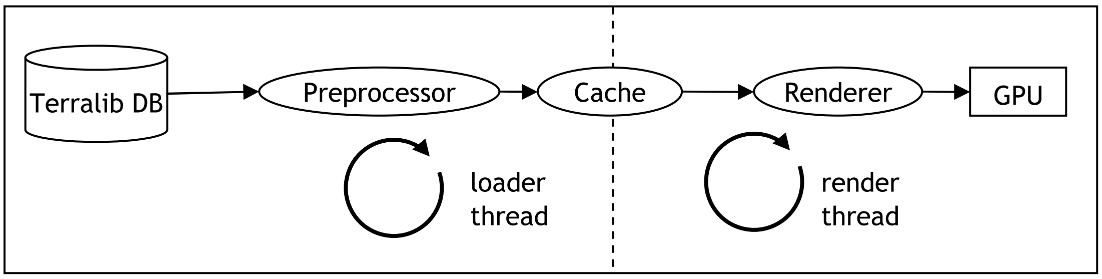

Figure 1. Rendering Process.

2.3 Data Format and Preprocessing

As it was already mentioned, a memory cache is needed in order to render *maps* quickly and make the whole system more user-responsive and interactive. A number of techniques have been developed to meet these requirements [Certain et al, 1996, Matos et al, 1998, Pinheiro et al, 2004].

Our approach was consider that data should be stored in memory in a format as close to the graphics API format as possible in order to improve rendering performance. That's why the data is preprocessed before being stored in the cache. This preprocessing step includes: (i) geographic projection conversion; (ii) polygon tessellation (breaking them in triangles) and (iii) style grouping (putting together geometries that will be drawn with the same style).

The reason for projection conversion is that converting geometries between two geographic projection systems may involve quite complex calculations and it is very costly compared to the total rendering time, therefore converting the geometries in advance and only once helps improving rendering performance. The polygon tessellation process consists in generating a triangle mesh that will cover exactly the same area as the original polygon. Although it is costly, the tessellation is necessary to make full use of all the GPU features such as parallelization and pipelining since its architecture is more prepared to handle triangles. Lastly, style grouping is also an optimization for the GPU because it allows rendering the same data with less state changes, such as changes in the current color, line width, textures and so on.

By doing all preprocessing in advance, every time the system needs to render a *map* it will do it in the fastest possible way if the data is in the memory cache. But what should happen if it is not? From the user point of view, it is highly desirable that the system is responsive and that it gives the most feedback as quickly as possible. This way, in this situation, the system renders whatever is in memory and starts loading the missing parts without blocking the user interaction. The system was designed as multithreaded exactly to handle this case: load data while interacting with the user.

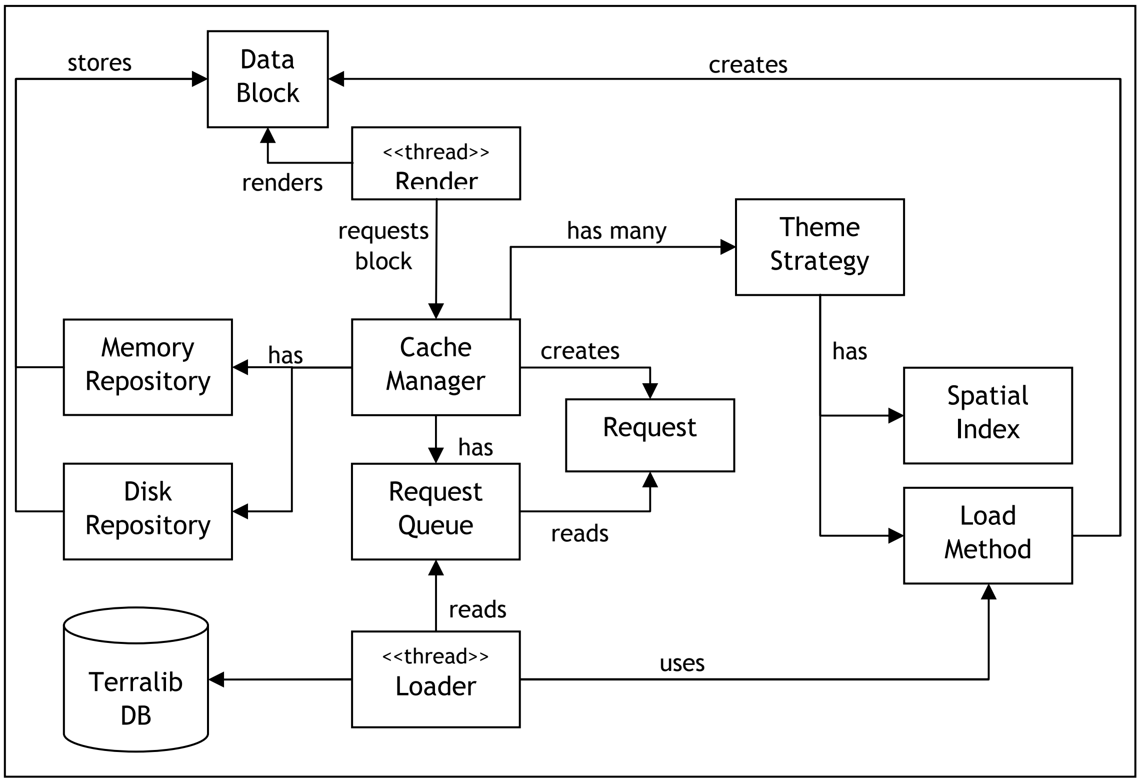

Figure 2. Functional Diagram.

There are currently two threads in the system: the *loader thread* loads data from the database to the cache and the *render thread* renders *maps* and handles user input. These two threads communicate through the cache: the *render thread* reads from the memory cache and the *loader thread* puts data in it. Besides that, when the *render thread* needs data that is not in memory, it queues a request to the *loader thread* using a *request queue*. The *loader thread* continuously checks the request queue looking for new requests.

The functional diagram (Figure 2) shows the main components of the system. The process starts when the *render thread* requests a *data block* to the *cache manager*, who checks whether the block is already in the *memory repository*. If it is, just return it. If it is not, check if it is in the *disk repository*. If it is, load it to the *memory repository* and return it. If it is not, load it from the database to the *memory repository* and return it.

Whenever the *memory repository* is full and a data block needs to be loaded, the *cache manager* should pick one other block in memory chosen by some heuristics (e.g. the block that is not used by the largest period) and save it in the *disk repository*. If the *disk repository* is full, then it needs to pick a block and just delete it to free some space. This way the cache is composed by a three-level hierarchy [Matos et al. 1998] that should, by the use of good heuristics, optimize the whole process of data visualization by using all resources available in an efficient way.

2.4 Immediate Response

In some cases, especially when *continuous interaction* is desired, the *render thread* cannot wait for the loading process to finish, even if it consists of just one block. In this case, it should put a flag named "immediate response" in its request. This flag will make the cache return a block only if it is already in memory. If it is not, it will queue a request for the *loader thread* and return a null block. This is how a behavior like Google Earth or NASA World Wind can be achieved with this cache system. By always returning the request call immediately, the user interaction is guaranteed not to stop, even if all data needed to render the *map* has not been loaded yet. In this case, the user will see an incomplete map while the rest of the data is being loaded.

2.5 Hints from the Render Thread

Imagine the following scenario: the user starts navigating through some path in a faster way than the cache can load all the data along it. Using a naïve strategy, what will happen is that, when the user stops navigating, he will wait for the system to load all data along the path because a lot of requests were queued in the way, even if only a small part of all this data is still visible to him.

The *render thread* can give hints to the cache to help it optimize the loading task. For instance, it can provide the information of a rectangle, inside which is all the data the user is currently visualizing. This information can be extremely interesting to the cache because, if it has not started loading a *data block* and this block is not within the visualization area anymore it does not have to be loaded. In the scenario just mentioned above, after the user stops moving, only the data visible to him will be loaded.

2.6 Points of Flexibility: Data Format and Indexing

The system architecture offers two great points of flexibility (hot spots): the *data block* and the *theme strategy* components.

The *data block* is a class that defines the format in which the geospatial data will be stored in memory plus the preprocessing functions. Any application can write its own *data block* class by implementing a common interface and provide a factory [Gamma et al 1995] of *data blocks* to the *cache manager*. This way the cache engine can hold spatial data in any format. This is very convenient for fast rendering because the data can be stored in the most convenient format for the graphics toolkit.

The other flexibility point is the *theme strategy*. This class implements a common interface and is responsible for the blockage and spatial indexing of a *theme* and the loading method. By blockage of a *theme* we mean the way the objects of a *Terralib theme* will be grouped into blocks (usually by some way that takes spatial proximity into consideration). The spatial indexing strategy is a data structure used to spatially query the data blocks in memory, more specifically it answers questions like, for example, which blocks intersect a window query. The default structure provided is a R-Tree [Oosterom, P.V.,1999], but any application can implement other strategy. The loading method is simply a method that loads a block. This flexibility is important because it allows the use of indexed persistence formats in an efficient way. It also allows any application to load data from data sources other than a *Terralib* database.

3. Results

The *VIPE* [TDK 2007] application is built on top of *TDK* and was used to test the results of the architecture proposed in this paper. This section presents experimental results we have obtained with three distinct versions: *VIPE v0*, *VIPE v1* and *VIPE v2*, this last one was built especially to test the concepts and architecture proposed in this paper. The others versions are explained in better detail later in this section.

Since *VIPE v2* could successfully reproduce the user experience of systems like Google Earth and World Wind in terms of rendering speed and system responsiveness, it would be very interesting to compare it with them. However, we found no direct way to make a full comparison between these systems.

There are basically three measurable factors critical to systems like these: database loading performance, disk loading performance and rendering performance. The database loading performance cannot be compared because Google Earth and World Wind cannot access a *Terralib* database. A disk loading performance comparison could be unfair since in our system the data on disk is already preprocessed and efficiently blocked while the other systems would have to access a common file format like shape files [ESRI 1998] or GML[OGC 2007]. Finally, a rendering performance comparison could also be unfair since the other systems render data in a 3D environment while ours works only in 2D for now, besides there are significant differences in the style representation capabilities for rendering geographic features in each of these systems. Because of all the factors just mentioned, we have decided to compare *VIPE v2* with one other existing application, namely *Terraview v3.1.4* [INPE-

DPI 2007] and two previous versions of *VIPE* (*v0* and *v1*) because all of them are built on top of *Terralib*.

The test set used was a *Terralib* database build from data provided by IBGE (the Brazilian Institute for Geography and Statistics) and consist of the following *themes* (all of them covering all the territory of Brazil): municipal district boundaries(5,621 polygons), permanent rivers (179,701 polylines), highways (30,969 polylines with), cities and capitals (5,771 points). The total amount of coordinates of all these *themes* is 2,903,498. The machine in which the tests were executed was a Pentium Dual Core 3.00 GHz, 1GB RAM with a nVIDIA GeForce 7300 SE/7200 GS graphics card.

The *Terraview* and *VIPE v0* systems are very similar with respect to the rendering process. Neither implements any sort of caching mechanism. Every time they need to render a *map*, they just generate a SQL query to the *Terralib* database to fetch all geometries that intersect the visualization window (i.e. the area being rendered) and render the *map* in a loop consisting of three steps: fetch a geometry from the record set returned from the database, do all the processing needed and finally render the geometry. The loop goes on until there are no more geometries intersecting the visualization area. It becomes clear that it is not possible to compare database loading performance or rendering performance individually because these operations are interleaved in these systems. Therefore, the comparison was made by the whole operation (loading and rendering).

The *VIPE v1* system implements a cache mechanism and it stores raw database data in the *Terralib* format. All the processing is done at rendering time. In order to test the rendering performance with and without the data in the cache, two time measures were taken, one for rendering right after connecting to the database (without any data in the cache) and one right after this first (with all the data in the cache). Special care was taken to configure the cache so that the entire map would fit in memory.

All tests were run multiple times and the average time was recorded. This way was preferred to testing with a profiling tool because we wanted results as close to the user experience as possible. It was interesting to compare different versions of the same system to have a better idea of how the evolutions on the architecture influenced the user experience.

Table 1. Experimental Results Comparison.

System	**Average Time for Map Rendering**	
	Uncached data	**Cached data**
Terraview	37.38s	-
VIPE v0	53.21s	-
VIPE v1	60.20s	8.14s
VIPE v2	114.40s	0.10s

We observed in Table 1 that both *VIPE v1* and *VIPE v2* performed better when the data was in the cache. This confirms the utility of caching in reducing the time for map rendering.

We can also see that with cached data, *VIPE v2* was much more efficient than *VIPE v1*. This occurred because *VIPE v2* caches preprocessed data in a format very close to the one requested by the graphics toolkit. This test shows us that this sort of preprocessing can bring extreme performance improvement in rendering time, especially in a toolkit that explores well the GPU features such as *OpenGL*.

The performance achieved by *VIPE v2* was enough to implement *continuous interaction* for a reasonably large data set at an acceptable frame rate. It is a consensus in the computer graphics area that the minimum frame rate for real time scientific visualization is about six frames per second, which allows a maximum rendering time of approximately 0.16 seconds.

It is worth mentioning that, during the test with uncached data, *VIPE v2* rendered all data loaded so far about 1,000 times while the others only needed to render once. This is necessary because *VIPE v2* allows continuous interaction. That certainly contributed for it taking more time to finish loading all the data than the others. A positive side of it is that, while it was loading the data, the user navigation was not blocked.

Finally, the systems without caching mechanisms (*Terraview* and *VIPE v0*) seem to load data more quickly than the others. This accounts probably to the fact that their rendering processes do not have the preprocessing and caching steps (illustraded in Figure 1). Both *VIPE v1* and *VIPE v2* need to organize data in memory to take advantage of the spatial dimension of data and therefore, some amount of extra processing is needed. In the case of *VIPE v2*, the data has to be grouped into blocks by spatial proximity and also, a spatial index structure for the data in memory has to be maintained. Besides that, in this test, it rendered all its cached data about 1,000 times.

Since *VIPE v2* rendering time improved approximately by a factor of 80 compared to the *VIPE v1* system (by comparing their average times with cached data), the conclusion we take is that, in GIS, it is worth paying attention to the GPU capabilities and make an effort to provide data in an efficient format to it.

4. Conclusion and Future Work

In this paper we proposed an architecture implemented in TDK that preprocesses data and stores data in a memory cache to reduce the time for rendering maps and to make the GIS applications more user-responsive. This strategy allows the user to navigate through geospatial data in a smooth and continuous way.

We also presented experimental results we have obtained that confirm that this strategy makes a big difference in rendering performance in GIS applications.

This project opens a lot of space for future work. The main lines would be: raster support, optimization, applying these caching techniques to servers and taking advantage of *OpenGL* and the GPU for 3D rendering and navigation.

Although raster data (i.e. data represented by a rectangular grid, such as images) is not supported in this first version of the cache, it would certainly be a very useful feature to add in the nearby future. Raster support would probably require multi-resolution, which means that it would be necessary to manage multiple versions of the same data, one for each different resolution.

With regards to optimization, it would be interesting to study different formats to store spatial data in the database. For instance, the objects could be stored already in blocks, each block in a record of a table. New algorithms for theme blockage and spatial indexing could also be tried. Finally, there is a lot of space for research and improvement in cache management policies and heuristics. One good thing would be to make these policies and heuristics flexibility points so that each application could override the default and write some better suited for its specific use.

One more improvement could be a prefetcher component, who would try to predict what data would be requested to the cache in the near future and load it in advance. It could consider for instance all data around the area currently being visualized.

Since the main purpose of all this system is to render *maps* quickly using cache and GPU, it seems that it would fit perfectly in GIS servers that render lots of maps. By using proper cache management heuristics a server would quite likely improve its performance by making full use of all its resources.

Another challenge, this time much more difficult, would be to present the data from the *Terralib* database in a 3D navigator. Although the task may be done in a simple way just by laying the 2D geometries on a sphere surface, it would be far more interesting to create some sort of translators from a 2D object to 3D and place them in a real 3D environment with terrain modeling, sky, oceans and everything else to make it look nicer.

5. References

ATI (2002). "Radeon 9700", http://www.ati.com/ .

Bar-Ze'ev, A. (2007). "How Google Earth [Really] works", http://www.realityprime.com/articles/how-google-earth-really-works.

Câmara, G., Souza, R., Pedrosa, B., Vinhas, L., Monteiro, A., Paiva, J., Carvalho, M., Gattass, M. (2000). "TerraLib: Technology in Support of GIS Innovation", II Brazilian Symposium on GeoInformatics, São Paulo.

Certain, A., Popovi´c, J., DeRose, T., Duchamp, T., Salesin, D. and Stuetzle, W. (1996). "Interactive multiresolution surface viewing. In SIGGRAPH 96 Conference Proceedings, pages 91–98.

ESRI (1998) "ESRI Shapefile Technical Description. An ESRI White Paper", http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

Faconti, G. and Massink, M. (2000). "Continuity in human computer interaction", *ACM SIGCHI Bulletin*, Vol. 32(4) - Sept./Oct.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (2005). "Design Patterns: Elements of Reusable Object-Oriented Software", Addison-Wesley.

Google (2007) "Google Earth", http://earth.google.com/.

INPE-DPI (2007) TerraView, http://www.dpi.inpe.br/terraview.

Longley, P.A., Goodchild, M.F., Maguire, D.J., Rhind, D.W. (2001). "Geographic Information System and Science", John Wiley.

Matos, A., Gomes, J. and Velho, L. (1998). "Cache Management for Real Time Visualization of 2D Data Sets". In *Proceedings of SIBGRAPI 98*, pages 111–118. SBC - Sociedade Brasileira de Computacao, IEEE Press.

Mesa, A. F. (1998) "A History of the Graphical User Interface", http://applemuseum.bott.org/sections/gui.html/ .

Nasa (2007) "World Wind", http://worldwind.arc.nasa.gov/.

NVIDIA (2002) "GeForce FX", http://www.nvidia.com/.

OpenGL (2007) "The Industry's Foundation for High Performance Graphics", http://www.opengl.org/.

OGC - Open Geospatial Consortium (2007) "GML - the Geography Markup Language", http://www.opengis.net/gml/.

Pinheiro, S., Celes, W and Velho, L. (2004). "Um sistema de cache preditivo para o processamento em tempo-real de grandes volumes de dados gráficos". In *SIBGRAPI Workshop of Thesis and Dissertations*.

Smith, S., Duke, D. (1999). "The Hybrid World of Virtual Environments", Eurographics, Volume 18, Number 3.

S. Sun and X. Zhou, "Semantic Caching for Web-Based Spatial Applications", in Proceeding of APWeb 2005 (LNCS 3399), pages 783-794, March 29- April 1, 2005, Shanghai, China.

TDK (2007). "Terralib Development Kit", http://www.tecgraf.puc-rio.br/tdk

Tilio, M. And Vera, M.S. (2005). "Desenvolvimento de Aplicativos com a Terralib", In Bancos de Dados Geográficos, Edited by Casanova, M A, Câmara, G., Davis Jr, C.A., Vinhas, L. Queiroz, G.R., MundoGEO, pages 477-506.

Oosterom, P. V. (1999). "Spatial access methods. Chapter in Geographical Information Systems Principles, Technical Issues, Management Issues, and Applications", edited by Longley, Goodchild, Maguire en Rhind, John Wiley pages 385-400 (vol.1), 1999.

Oosterom, P. V. (1988). "Spatial Data Structures in Geographic Information Systems", Computer Science in the Netherlands, pages 463 - 477.