Ibero-American Symposium on Computer Graphics - SIACG (2006), pp. 1–4
P. Brunet, N. Correia, and G. Baranoski (Editors)

GeoViewer3D: 3D Geographical Information Viewing

Rafael Gaitán[1], María Ten[1], Javier Lluch[1] †, Luis W. Sevilla[2‡]

[1]Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, 46022
[2]Consellería de Infraestructuras y Transporte, Avenida de Aragón
Valencia, Spain

Abstract

Our State Government is developing a Geographical Information System (GIS), called gvSIG. This project follows the open source philosophy, and uses JAVA as development platform. Consequently, it is portable and can be used by anyone around the world. In this paper, we describe the design and architecture of a prototype for browsing 3D geospatial information. GeoViewer3D is a system that handles and displays worldwide satellite imagery including textures and elevation data. It has a modular architecture and an efficient 3D rendering system based on OpenSceneGraph. The system incorporates a disk cache to improve access to GIS data. The goal of this prototype is to join the gvSIG project as 3D information viewer.

Categories and Subject Descriptors (according to ACM CCS): I.3.4 [Computer Graphics]: Graphics Utilities H.4 [Information Systems Applications]:

1. Introduction

A Geographical Information System (GISs) is an integrated system that stores, analyzes, shares, edits and displays spatial data and associated information. Recently GIS have become more important due to the great variety of application fields, like geology, tourism, navigation and military and urban planning [Aro89, LT92].

gvSIG [AR05] is a tool oriented to manage geographic information. It is characterized by a user-friendly interface with quick access to the most usual raster and vector formats. It also includes local as well as remote data access and retrieval through either the Web Map Service (WMS) or other services provided by the *Open Geospatial Consortium* (OGC) [Ope06].

gvSIG emerges inside one of our State Government offices *Conselleria de Infraestructura y Transporte*. It is a Free Software solution for a GIS client that supports the most common cartographic features. The system is completely developed in Java, so the application runs on almost all platforms.

Recent advances in 3D technology are gradually enabling the development and exploitation of 3D graphical information systems [FPM99, Jon89]. New applications, like Nasa World Wind or Google Earth have lately been developed. Still, there are only a few 3D geographical information applications.

GeoViewer3D is a system that handles and displays 3D GIS information within the *gvSIG* project. The application is able to manage multiple layers of texture and Digital Terrain Models (DGTs). The system displays loaded layer information at interactive rates using the OpenSceneGraph 3D rendering engine. The application also manages a local cache of geographical information in order to improve access to remote services or georeferenced files.

In this paper we describe the design and architecture of this 3D viewing system within the framework of the gvSIG project. We also present preliminary results obtained with our prototype implementation.

The paper is structured as follows. First, we introduce the tools used for the development of the application. Then, we present the design and architecture of the system and the main features of the application. Finally, we show some results of the 3D GIS viewer. The paper ends with conclusions and directions for future work.

† {rgaitan, mten, jlluch}@dsic.upv.es
‡ sevilla_lui@gva.es

2. Tools

In this section, we describe the tools employed in the development of our system. We use the OpenSceneGraph [OB04] (OSG) 3D graphics library and the JNI programming framework [SM03].

OSG is an open source high performance 3D graphics toolkit that provides an object oriented framework based around the concept of a scene graph. OSG is based on OpenGL rendering, but it supplies many additional options for rapid development of graphics applications like flight simulators, games, virtual reality applications and scientific visualization.

Its performance, scalability, portability and productivity are the main features of OSG. They make possible the development of high performance graphics applications with high-speed rendering optimizations. The core scene graph barely depends on any specific platform, requiring little more than Standard C++ and OpenGL to run.

JNI is a programming framework that allows Java code running on the Java virtual machine to call and be called by native applications and libraries written in other languages such as C, C++ and assembly language. Java's ability to communicate with C++ allowed us to develop a portable Java interface with OSG-based libraries. This is the interface that GeoViewer3D provides to its users.

3. System Design and Architecture

In this section we present the architecture of our system. We also describe how it solves the problem of accessing geospatial data using Java libraries.

The management of GIS data is carried out on the Java module using some of the current functionality of the *gvSIG* development project. We render 3D information using tiles. A tile contains geospatial data associated to a square or rectangular area of terrain. These data may be a texture image, typically obtained from aerial or satellite photographs, or geometric information like digital elevation data that can be triangulated to display 3D terrain.

3.1. Architecture

The system has a modular architecture in order to comply with the requirements of the *gvSIG* project. We need a solution rich enough to show geospatial information in 3D and fast enough to render at interactive rates.

We start the development by splitting the system into two native C++ libraries that export functionality to Java using JNI technology. They extend some of gvSIG's libraries in Java to support access to geospatial information.

Figure 1 shows the architecture of the system. We have four independent libraries: *libjosgviewer*, *libjosgplanets*, *cqRasterCache* and *cqRemoteServices*. The main application is responsible of managing GUI events and inter module communication.

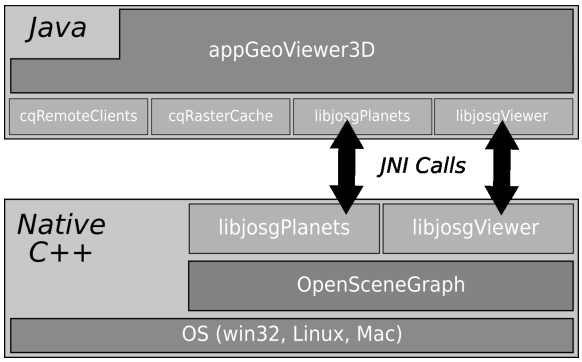

Figure 1: *Architecture of the system. The* cqRemoteClients *and* cqRasterCache *modules are based on from the* gvSIG *project.*

The *libjosgviewer* library creates an *OSG* viewer inside a Java swing application. The viewer integrates and renders the *OSG* scene graphs. The library uses *jogl* to create the rendering context and our own *JNI* wrapper on the native (C++) side to support the *OSG* viewer.

The *libjosgplanets* library supports creating a scene graph with a definition of a planet. Planets can be defined using equatorial and polar radii, geometric extension and the required projection type (Plane or Spherical). The library has a native (C++) module where the planet scene graph is created and a Java module that exports its functionality implementation.

Each planet is composed of a set of tiles. Each tile contains the geometry of the spatial region that represents. The geometry is generated by the *HeightField* component provided by *OSG*. We build a grid with the elevations obtained of the digital elevation data. If the type of the projection is spherical, then we project each vertex of the *HeightField* into the ellipsoid surface of the planet.

The *cqRemoteClients* library supports access to geographic data from remote services. The library allows access to the *OGC* and Google Maps services.

Finally, the *cqRasterCache* library manages the copies of the tiles stored in the viewer's cache. The cache is filled on demand so that we do not penalize tile loading at start-up time. When a tile is requested to the cache, the library checks if the tile is already in its filesystem. If not, retrieves the tile using a remote service or accessing to a georeferenced file located in another filesystem. Then the cache saves the tile in its own filesystem.

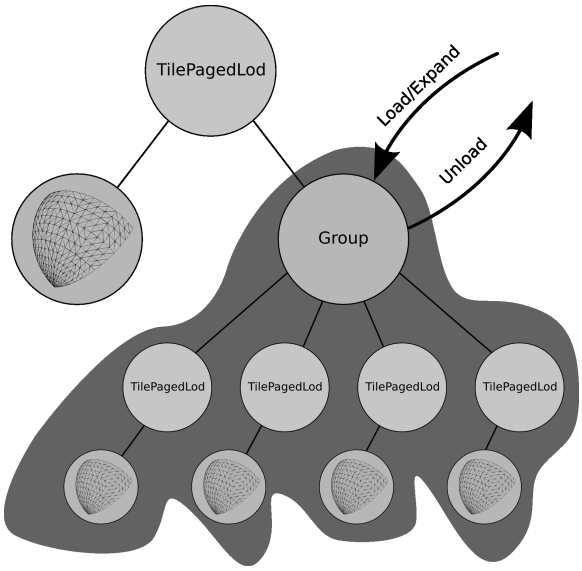

Figure 2: *Scene graph generated during navigation. Highlighted region is generated during navigation.* OSG *can also load and unload previously generated nodes from main memory.*

3.2. Dynamically Paged LOD Tiles

OpenSceneGraph supports the navigation of preprocessed paged LOD datasets. A paged LOD node is a level-of-detail group node. It also allows to load and unload children nodes to disk, thus optimizing memory use. Usually paged LOD datasets are preprocessed and saved to disk, but GIS systems are complex, dynamic and with multiple and different sources of data. Therefore prepocessing geospatial data it is not always possible. We have adapted and extended the *OSG* navigation of paged LOD datasets in order to support dynamic data.

The *libjosgplanets* library supports the creation of dynamically paged LOD tiles(*TilePagedLOD*). We describe the process of creating and subdividing tiles. We also explain how to request a texture and a digital terrain model of the tile.

The process is done during the cull traversal of the scene graph. For each visible tile we obtain the distance from the viewer to the tile. Then we compare the given distance with the visibility ranges of the *TilePagedLod* children to determine if we need to increase the level of detail. If that is the case and the current tile has not been previously subdivided then we expand the scene graph by creating four child tiles. We use quad-tree based subdivision algorithm to create the children. Then we update their distance ranges and save them to disk. If the tile had been previously subdivided then we let *OSG* proceed with the default *PagedLOD* cull traversal, allowing the paging of the nodes.

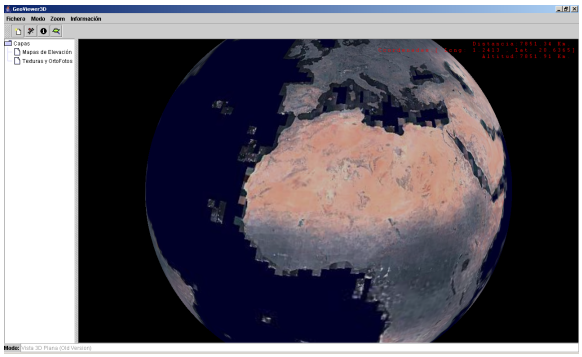

Figure 3: *The Earth rendered using Nasa WMS data.*

Figure 2 shows the scene graph generated during navigation. The group inside child one of *TilePagedLod* is expanded when required by navigation. It can also be unloaded from memory when the OpenSceneGraph PagedLOD traversal process decides. If the group was already expanded and unloaded, then the system loads from disk the required child.

The process that requests textures and digital elevation data also runs during the cull traversal of scene graph. When a request is issued, we throw a callback. The callback gets the data from any attached data source layers located on the Java side. If data arrives when the tile has been unloaded from memory(*OSG* paging process), it is discarded.

4. Results

We present some results obtained with the GeoViewer3D application. Currently the system supports any kind of planet definition (equatorial radius, polar radius, extension, etc.) on the Java side. It also allows attaching to the planet, layers of any kind currently supported by the *gvSIG* system.

Planets can be defined to be *spherical*, in order to work with geocentric referenced data, or *planar* to support cartesian coordinate system layers. Geocentric data uses latitude, longitude and height (z), while cartesian layers use x, y, and z measured in meters.

Figure 3 shows the Earth rendered using as base layer data provided by the Nasa WMS remote service. Figure 4 shows a detailed view of the same dataset. The system subdivides tiles in order to show a more resolution for closer distances.

Figure 5 shows a set of layers loaded and rendered from raster files. Raster files are georeferenced in cartesian coordinates, so the system draws a planar view of the Earth with a predefined extension in meters.

The system runs at interactive rates, 30 fps and more, even when there are cache misses. The application loads missed tiles using a separate thread to avoid stalling the navigation.

Finally, Figure 6 shows a preliminary view obtained with

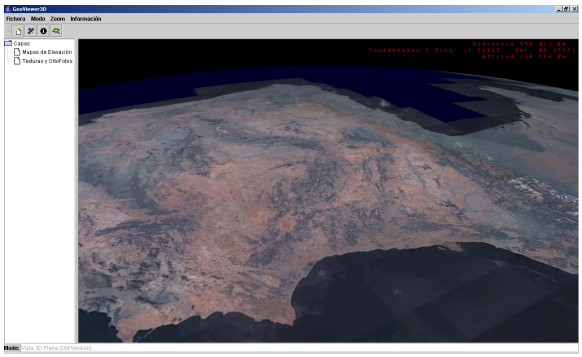

Figure 4: *Detail of the Earth rendered with higher-resolution Nasa WMS data.*

Figure 5: *View of the Earth rendered using georeferenced files in cartesian coordinate system.*

the 3D version of our viewer. The view shows a landscape with different heights rendered using texture-mapped polygons. The texture was obtained from the WMS Nasa server which maintains a worldwide satellite imagery. The polygons were generated using a height field texture obtained from the same server, which contains the digital elevation data.

5. Conclusions and Future Work

We have presented a prototype system for 3D geospatial viewing. The system renders at interactive rates using multithreading and a disk cache of processed tiles. Also, it can load and show layers of different sources, like WMS remote services, Google Map services, and local files. Finally, the system can manage layers with Digital Terrain Models.

In the future we want to create a 3D Geospatial Information System capable of analyzing, processing, editing and animating 2D and 3D geospatial data. The next step is to create a complete 3D viewer to manage vector data,

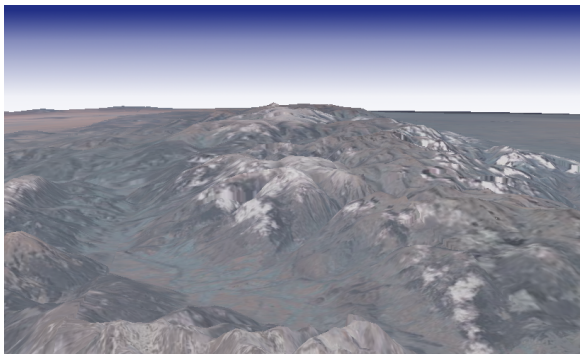

Figure 6: *A 3D view of Earth data rendered using both texture information and digital elevation data*

TINs(Triangulated Irregular Network), different symbology and Urban Digital Models.

Acknowledgements

This work was partially supported by the *Conselleria d'Insfraestructures i Transport* of the Valencian State Government (Spain), by grant TIN2005-08863-C03-01 of the Spanish Ministry of Education and Science, and by STREP project IST-004363 of the 6th FP of the European Union.

References

[AR05] ALFARO A. A. A., RICO G. C.: gvsig: Open source solutions in spatial technologies. In *GIS Planet* (2005).

[Aro89] ARONOFF S.: *Geographic Information Systems: A Management Perspective*. Ottawa, Canada: WDL Publications., 1989.

[FPM99] FLORIANI L. D., PUPPO E., MAGILLO P.: *Applications of computational geometry to geographical information systems*. Elsevier Science, 1999, ch. 7, pp. 333–388.

[Jon89] JONES. C. B.: Data structures for three-dimensional spatial information systems in geology. *International Journal of Geographical Information Systems 3(1)* (1989), 15–31.

[LT92] LAURINI R., THOMPSON D.: *Fundamentals of Spatial Information Systems*. Academic Press London, 1992.

[OB04] OSFIELD R., BURNS D.: Openscenegraph, *www.openscenegraph.org*, 2004.

[Ope06] OPEN GEOSPATIAL CONSORTIUM: *OpenGIS® Web Map Service (WMS) Implementation Specification*, March 2006.

[SM03] SUN MICROSYSTEMS I.: Java native interface, *http://java.sun.com/j2se/1.4.2/docs/guide/jni*, 2003.