Universidade do Minho
Escola de Engenharia

Paulo Adelino Dias Almeida

An Open Source Virtual Globe for Android

Dissertação de Mestrado
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação do
Professor Jorge Gustavo Rocha

Outubro de 2013

Agradecimentos

Em primeiro lugar gostaria de expressar um importante agradecimento ao Professor Jorge Gustavo Rocha, por todo o seu apoio, motivação e partilha de conhecimentos, que tornaram esta tese possível.

Queria também agradecer aos colegas que partilharam o gabinete comigo durante a duração deste projecto e proporcionaram um óptimo ambiente de trabalho.

Por último, agradeço aos meus pais e à minha irmã por terem sempre acreditado em mim e terem sempre me oferecido todo o seu apoio e carinho.

Abstract

Virtual globes have a number of key benefits as a platform for communicating and visualizing geospatial data over traditional technologies. Virtual globes have increased in popularity and several implementations are available that cater to different audiences from education to industry.

Despite these advantages, an open source virtual globe solution is still not available for mobile environments.

Our goal is the development on an open source globe for Android, able to receive 3D scenes from a W3DS server. We present the architecture and the implementation decisions. We choose to develop the virtual globe on top of osgEarth which takes advantage of the OpenSceneGraph toolkit. Based on this decision, we explain how osgEarth was extended to consume new 3D data sources and how it was ported to the Android platform. Porting to Android requires major changes in the OpenGL API usage. Embedded devices only support a subset of the OpenGL API.

We provide a virtual globe application that runs natively on the Android operating system. It is implemented on top of the osgEarth framework. osgEarth was ported to Android and expanded to support additional features. Pointers to the source code repositories are provided.

With the work developed in this project, mobile virtual globe solutions can be customized and deployed, providing powerful visualizations and more intuitive interactions.

Resumo

Nos últimos anos, aplicações de globo virtual sofreram um grande aumento na sua popularidade e proliferação. Este tipo de aplicação oferece um grande conjunto de vantagens em relação às soluções tradicionais para a visualização e interação com dados geoespaciais.

Estas vantagens levaram a um elevado interesse na presença desta solução em ambientes móveis. No entanto, uma solução *open source* para globos virtuais em Android ainda não se encontra disponível.

O objectivo principal deste trabalho é então disponibilizar em Android uma solução de globo virtual open source. O globo implementado terá também de ser capaz de consumir o serviço W3DS recentemente especificado.

Apresentamos a arquitectura da nossa solução e as escolhas realizadas. Escolhemos basear a nossa solução no osgEarth, *framework* de globos virtuais que recorre ao OpenSceneGraph para as suas necessidades de *rendering*. Esta decisão implicou um processo de *porting* destas *libraries* para Android, efectuando todas as adaptações necessárias. De especial importância a adaptação do código dos *shaders* responsáveis pelo *rendering* gráfico, uma vez que em Android apenas há disponível o OpenGL ES, especificação limitada do OpenGL. O osgEarth foi também expandido de forma a ser capaz de consumir o W3DS.

Disponibilizamos uma solução de globo virtual que corre nativamente em Android e é capaz de consumir o W3DS. A *framework* osgEarth foi assim expandida com novas funcionalidades e passou também a estar disponivél para Android.

Com o trabalho realizado, globos virtuais móveis podem ser personalizados e implementados fácilmente.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

There is a growing interest in the visualisation of and interaction with geographic information. Recent years have seen the appearance and popularisation of a high number of solutions where geospatial and georeferenced data play a key role. Consequently several efforts have been made in developing standards and services for the representation and processing of geographic information. Along with these efforts, the need for a uniform and unifying medium for the visualisation and interaction of this data became an obvious priority. This led to the development of several client applications with this purpose, however most of these applications have been limited to a specific type of data, functionality, or scientific domain.

Thus, an event that greatly contributed for the increase of popularity that geographic solutions and systems have displayed among the general public was the appearance of virtual globe applications. This type of application allows the user to interact with a 3D multi-resolution representation of the planet, while integrating several types and sources of geographic information.

Virtual globes allow us to display attributes of the geographic data that a classical top-down view can not easily convey. Information related to the topography and elevation of the terrain, or 3D dimensions of buildings and city infrastructure are intuitively perceived when represented in a virtual globe.

Virtual globe solutions also provide a uniform and familiar interface for interacting with data from a wide range of sources and disciplines. For example, the

digital terrain model of a given area can be presented with integrated information related to the topography of the terrain and the weather data collected for that area. All this information is then accessed, navigated, and controlled in a uniform manner.

The representation of data about underground phenomena and structures, that usually requires specialised and expensive software to render and analyse also gains greatly by the intuitive and flexible representation offered in virtual globes. The possibility of performing and record virtual fly-throughs and output them in a video format allows for a visually compelling, easily distributable method to illustrate spatial data. Thus, virtual globes make data much more accessible, which is important for communicating scientific data to a broad audience.

The growing interest that geographic applications have been presenting, together with the numerous benefits associated with the treatment of this type of information on a virtual globe, resulted in the desire of having this type of solution available in all kinds of platforms, specifically mobile devices.

Initially this interest demonstrated by the public was impossible to be fulfilled. The limited capabilities associated with these devices meant that applications such as a virtual globe would have its implementation in mobile devices either made impossible or extremely limited. However, in the last few years we have witnessed a great increase in processing, storage, and graphic capabilities of mobile devices, making its differences to a desktop platform progressively less significant. This opens the possibility for the development of applications and functionalities that have, so far, been impossible to implement in these types of devices. With this elimination of previous limitations, the associated solutions based on removed functionalities, simplifications, and server-side processing are no longer an imposition.

Obviously some limitations are still present when compared to desktop platforms. These limitations are always be a concern to the developer. In the particular case of interactive 3D rendering, the necessity to increase the complexity or the number of the entities rendered will always be limited by the capabilities of the mobile device.

1.1 Motivation

The objective of this project was to develop a 3D viewer for geographic and georeferenced information that would run natively on mobile devices, specifically on devices running the Android operating system. As an additional requirement, the viewer application would also have to be able to consume the newly specified Web 3D Service (W3DS).

Taking into account the previously referred advantages associated with virtual globes when compared to traditional geospatial technologies, in particular in terms of uniformity and usability, this application was chosen as the preferred solution for our problem. Also, seeing as the development of a plugin for the consumption of the newly specified W3DS service is part of our main goal it becomes fundamental to have an easily expandable system.

This expandability is an feature most commonly associated with open-source solutions. These kind of solutions are generally the base for the development of new capabilities and functionalities. They are usually preferred in new case studies for providing a great level of liberty and flexibility to the developer, in contrast to proprietary solutions that only provide a standard API.

Despite all the interest and potential present in virtual globes, we verified that there still exists a small number of open source implementations available. This lack of solutions is even more pronounced in mobile platforms, where open source implementations are practically non-existent.

1.2 Problem

This work follows from previous projects where several geographic information services were developed. These services were implemented to manage and provide efficient access to georeferenced information of city infrastructures, specifically infrastructures related to telecommunications. From the implementation of these services, the need for client applications able to present the existing information and allowing a set of users to interact with it became apparent. After the development of client applications for the desktop environment the need for a solution present in mobile platforms was revealed as being of much more utility.

Because of its considerable popularity and widespread use, and also because of the greater ease and support for software development, the Android operative system was chosen as the mobile platform used in the implementation of our solution.

The recent increase in popularity and general use of virtual globe applications in conjunction with the significant set of advantages, previously presented, that this type of solution offers led to its choice for client application in our system.

1.3 Goal

In this project we propose the development of an open-source virtual globe solution for the Android operating system. The implemented solution should be able to display 2D and 3D geographic information correctly positioned on the globe. The developed client application should be responsible for the rendering of the graphic components displayed. Another main requirement for the developed system is the capability of the client application to consume the newly specified W3DS.

This ultimate goal can be disaggregated into a set of specific tasks, from which we can highlight the following:

- Identification and study of existing open source implementations for virtual globe solutions in the desktop environment;
- Study and implementation of the necessary steps for the process of cross compiling applications for the Android Operative System;
- Preparation and execution of the porting process for all the necessary software modules;
- Analyse and comparison of the existing OpenGL support in the desktop environment in relation to embedded solutions;
- Development of shaders compatible with mobile device's GPU;

- Development of all the necessary logic to asynchronously consume geographic information services, in particular WMS and W3DS;

- Study and development of the best practises for applying styles on the client side.

1.4 Thesis structure

The remainder of this thesis is organised in five major chapters.

In the following chapter 2 we will present the state of the art relevant for our project. We begin by focusing on the Android operating system. Characterising the main points associated of developing software solutions for this environment and how this operating system presents some limitations when one needs to use the OpenGL API. Then, we present the specific concepts connected with virtual globe applications, also identifying and characterising existing solutions.

The analysis of the problem associated with the development of an Android virtual globe solution is presented in chapter 3. In this chapter we also analyse and design our approach to solve the enunciated problem.

In chapter 4 the major steps in the development of our solution are presented. We begin by the characterisation of the porting process, focusing on the specific modifications required by the OpenGL API. We also detail the steps and difficulties overcome in the extension of osgEarth in order to consume a W3DS.

We present the results obtained in this project, in chapter 5. Finally, in chapter 6, where we present the conclusions reached in the execution of this project and identify areas of possible future development.

Chapter 2

State of the Art

The implementation of a virtual globe application like the one we propose in this thesis implies the resolution of several problems relating with the rendering of 3D graphics and to the computation of geographic transformations and models. In addition to this particular challenge we propose to make this implementation natively in the mobile environment, specifically for the Android operating system, fact that will also provide additional constraints and challenges that need to be addressed. In this chapter we will present the major technologies and concepts associated with a virtual globe application for the mobile environment. We begin by introducing the Android operating system and what are the main characteristics associated with programming for this environment. Then we present OpenGL as the technology for the rendering of 3D graphics. This technology implies the most significant challenges and particularities in our implementation. Then the focus is given to the specific concepts associated with a virtual globe application. A brief history of the development of the concept of virtual globe is presented along with an analysis of several existing solutions.

2.1 Mobile Devices and Android

In the last few years we have witnessed a great proliferation of mobile devices. This kind of devices are now common among the general population, representing a research area and a market slice impossible to ignore. Consequently all solutions

implemented nowadays strive to be present in some way in these platforms.

Along side with the great increase in presence among the common user, mobile devices have been experiencing a constant increase in processing, storage, and graphic capabilities, making it's differences to a desktop platform progressively less significant. This opens the possibility for the inclusion of functionalities that have, so far, been impossible to implement in mobile devices.

With this elimination of previous limitations, the associated solutions based on removed functionalities, simplifications, and server-side processing are no longer an imposition. Obviously some limitations when compared to desktop platforms are still present, even if to a smaller degree, and will most probably always be a concern to the developer. In this particular case of interactive 3D rendering, the necessity to increase the complexity or the number of entities rendered will always be limited by the capabilities of the mobile device.

2.1.1 Android Operating System

With the proliferation of mobile devices there was also a great variety of new operating systems that were developed for those devices. In this initial approach each new series of devices would have a specific operating system. Consequently hardware and software portability were a serious and expensive problem. In order to address this problem a group of companies, led by Google, formed a consortium named Open Handset Alliance (OHA).

The OHA was established in 2007 with the primary objective of developing a standard platform for mobile devices [6]. For this objective they choose the operating system Android as the common platform for mobile devices.

Android is an open source Linux-based operating system developed primarily for mobile devices. Android was created by the company Android, Inc, later bought by Google in 2005. As was previously mentioned its main goal was to provide a common platform so that hardware and software developers could limit its efforts and expenses in achieving portability between devices. This simultaneous decrease in expenses and increase in portability of developed solutions allows for a greater competition between manufacturers bringing lower prices to the consumers.

Operating System	Unit Shipments	Market Share
Android	**159.8**	**70.1%**
iOS	47.8	21.0%
BlackBerry	7.4	3.2%
Windows Phone/Windows Mobile	6.0	2.6%
Linux	3.8	1.7%
Others	3.0	1.3%
Total	227.8	100.0%

Table 2.1: Top smartphone operating systems: shipments (in millions) and market share. Source: IDC Worldwide Mobile Phone Tracker, February 14, 2013.

Software applications will also benefit greatly by a common platform, limiting compatibility issues to the existing differences in hardware and capabilities of each device.

Android is clearly the most widespread mobile operating system, according to IDC data from the fourth quarter of 2012 (see table 2.1).

2.1.2 Developing in Android

The development of software solutions for Android is based on the Dalvik virtual machine. This virtual machine uses a register-based architecture and is optimised for systems that are constrained in terms of memory and processor.

One of its main features is the implementation of a garbage collection system that executes its function of freeing memory that is not currently being used while trying to maintain as much used memory as possible. This behaviour is responsible for the fact that Android applications are generally not destroyed or removed from memory, being only passed to the background [1]. Only when the user explicitly calls for a termination or when the system needs memory for a higher priority process, the application is finished and removed from memory. This behaviour allows the system to take advantage of the fact that the set of applications regularly used in a mobile device is relatively small, and more important, there is a couple of applications, like the calling and messaging applications, that are almost in constant use.

An application in Android is composed by a set of essential components. These components provide different points of communication and interaction with the

system and with the user. Despite the fact that each component exists as a self-contained entity, its through their collaboration that the desired behaviour of the application is implemented. In order to declare the existing components of an application, enabling the Android system to start each component, a manifest file is provided along with the application. This file is responsible for the definition of all the components that make up the application.

The components that have the greatest influence on the functionality of an application are the activities. An activity is represented by a single screen with an user interface, and a standard execution of an application normally implies a flow through several activities. As is the case with all application components, each individual activity of a given application is independent of the others. As such, any application can start the activities from other applications, given explicit authorization from the activity's original application.

This independence in addition to the previously referred garbage collection method, to which the activities are also subjected, results in a life cycle for Android activities that is a little unique and different from what is normally present in a desktop environment.

An Android activity is composed of four main stages:

Running if the activity is currently on the screen (at the top of the stack);

Paused if the activity is still visible but as lost focus (for example when a new non-full size or transparent activity is launched). A paused activity is completely alive retaining all state information, however it can be killed by the system in case of extreme low memory;

Stopped if the activity is completely obscured by other activity. A Stopped activity retains all state information, however, it is often killed by the system if memory is needed elsewhere;

Killed when the activity was dropped from memory, when it is launched again by the user it must be completely restarted.

These stages are further detailed in image 2.1 along with the methods called for each stage transition.

2.1.3 Android Native Development

Shortly after the popularisation of software development for Android, users started to question whether it made sense for this development to stay exclusively on the Java layer. Significant interest, and possible advantages, were identified in the use of `C`/`C++` code in developed applications. Users wanted to be able to reuse existing code developed in these languages, as well as take advantage of the greater efficiency they allow [7]. To address this interest Google introduced the Native Development Kit (NDK). This toolset comes to address this exact need, allowing developers to implement parts of their application using native languages as `C` and `C++` [8].

With the NDK, native classes and methods can be called from Java code running under the Dalvik virtual machine. To make this possible Android compatible libraries have to be built from the native code. The tool `ndk-build` enables the developer to execute this process, originating libraries that can then be made accessible to the Java code with the system call `System.loadLibrary`. The inclusion of native code in the application does not mean that it undergoes fundamental changes to its basic structure. The application is still packed in a texttt.apk file and will still be executed inside the Dalvik virtual machine on the device, without introducing major operating issues. To call a native method from the Java code after the inclusion of the relevant compatible library, the developer must use the Java Native Interface (JNI).

Before the decision to include native code, through the use of the NDK, several factors have to be considered. While it is true that certain performance critical portions of the code can benefit from the efficiency associated with `C` and `C++` code, the use of native libraries and the NDK also implies a significant increase in the complexity of the application. Only in cases where this increase in performance is indispensable, or where the reuse of existing code libraries written in these languages is a major factor, does the greater complexity associated with programming with the NDK become justifiable.

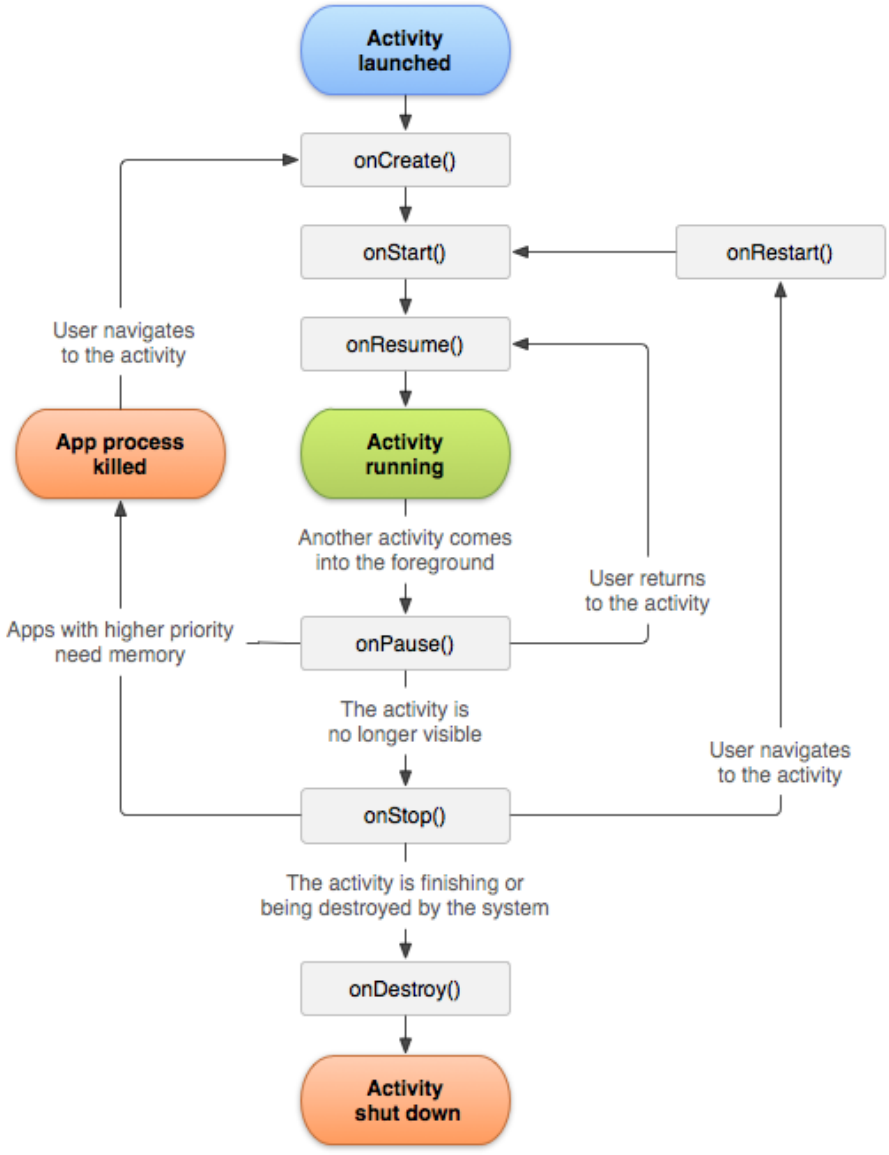


Figure 2.1: Diagram displaying the possible states and respective transitions in an Android activity (source [1]).

2.2 OpenGL

OpenGL is a multi-platform graphic rendering API developed by Kronos. At the moment, it is one of the most important standards for the render of 3D graphics.

With an implementation based on a system of extensions, that allows for the introduction of new functionalities in a flexible and efficient way, this API is mainly characterized by its direct communication with the graphic rendering hardware bypassing the operating system.

The process involved in rendering a given geometry with OpenGL can be summarized in the passing of the geometry's defining vertices through the pipeline of the graphic rendering hardware. In this pipeline several methods and transformations can be applied and the end result is then passed on to the display.

Initially this graphic pipeline presented a fixed nature, the developer was not able to define the operations and transformations performed to the geometry. However, with the introduction of OpenGL 2.0, the possibility to program some of the operations performed in the graphic hardware was introduced.

This programmable pipeline presents certain points where the developer can provide custom programs that will define the transformations applied to the geometry rendered.

These programs, referred to as shaders and implemented in the programming language GLSL [9], are responsible for a huge gain in the quantity and flexibility of methods that the developer is able to implement, opening the possibility for the application of several techniques and effects that were impossible with the fixed-function pipeline.

2.2.1 OpenGL ES

With the proliferation of mobile platforms among the common user, an implementation of this API specific for these platforms became increasingly necessary. OpenGL ES is then introduced with the objective of providing a graphic rendering API optimized for devices with limited resources. The specific constraints that OpenGL ES addresses are, limited processing capabilities and memory availabil-

ity, low memory bandwidth, great sensitivity to power consumption, and lack of floating-point hardware.

To accomplished this goal several functionalities and attributes present in OpenGL, whose purpose was in some way replicated in the API, were removed. A good example of this is in the specification of geometry. While in OpenGL the rendering primitives were originally described by issuing a begin command for a set of primitives, and then updating the current vertex positions, normal vectors, colors, or texture coordinates in an arbitrary order, and finally ending the primitive. This creates a very complicated state machine that does not run at an optimal speed. In current OpenGL versions, the vertex data is provided through vertex arrays and is rendered using calls to `glDrawElements` or `glDrawArrays`. OpenGL ES adopted only these simpler and more efficient approaches.

OpenGL ES 1.x

This first version of the OpenGL ES specification is based on the desktop version OpenGL 1.3. Besides the previously referred removal of several redundant features this specification is characterized by the extinction of per vertex operations, privileging the use of vertex arrays, the more efficient alternative. OpenGL ES 1.1 was developed based on OpenGL 1.5 and is completely backwards compatible with OpenGL ES 1.0. Several optimizations to memory and power usage were introduced in this version as well as considerable improvements to image quality. Previous restrictions on texture dimensions (in OpenGL ES 1.0 textures had to be square and with dimensions multiple of 2) were also removed, even though these restrictions are still recommended for efficiency motives.

OpenGL ES 2.0

Defined relative to OpenGL 2.0 this version of OpenGL ES has as its main feature the possibility to access the programmable rendering pipeline. With this API the user is capable of defining and using shader programs written in the OpenGL ES Shading Language [10]. Relative to the desktop version of OpenGL there are some limitations in the use of the programmable pipeline, seeing as the user can only define vertex and fragment shaders.

The use of the programmable pipeline is actually an imposition in this version, seeing as the fixed-function pipeline is no longer available. Consequently, unlike OpenGL 2.0, which implements a programmable pipeline but also provides full backward compatibility to older versions of OpenGL that implement a fixed function pipeline this specification is not backwards compatible with other versions of OpenGL ES. This drop of backwards compatibility is explained by the fact that, once you have a programmable pipeline, there is no reason to use the fixed function version, as you can directly program the effects you want to render with a lot more liberty and flexibility. In addiction, the OpenGL ES 2.0 driver's memory footprint would be much larger if it had to support both the fixed function and programmable pipelines. Taking into consideration the kind of devices targeted by OpenGL ES that fact is of great importance, outweighing the disadvantages of removing that feature.

OpenGL ES 3.0

This version of OpenGL ES was introduced in August 2012 and is backwards compatible with OpenGL ES 2.0. Several improvements were introduced to the rendering pipeline in this version, emphasizing the implementation of a new version of the OpenGL ES Shading Language with full support for integer and 32-bit floating point operations.

WebGL

WebGL is an API developed in JavaScript for the rendering of 2D and 3D graphics in compatible web browsers. This API provides a communication interface between the JavaScript code and the OpenGL ES 2.0 specification, whose implementation is a responsibility of all compatible browsers. Thus, any web page running in a WebGL compatible browser is capable of rendering 3D content, making use of the programmable rendering pipeline and directly injecting user code in the GPU. This issue is often times raised as a problem for the integrity and security of client applications and is the reason that several browser implementations shy away from providing WebGL compatibility [11] [12].

2.3 Virtual Globes

Since a long time ago, the visual representation that is most intuitive and descriptive of the planet Earth is the globe. Early terrestrial globes emerged following the establishment of the sphericity of the planet in the middle of the second century BC, being the Erdapfel [13], created in 1492, the oldest surviving exemplar at the present day.

The interest in transferring this physical representation of the Earth to a virtual environment arises with the development of the modeling language Virtual Reality Modeling Language (VRML) [14]. This language was introduced in 1994 and had as its main objective the representation of 3D animated worlds via the Internet. With the introduction of GeoVRML in 1998, the language begins to be extended to allow the representation of georeferenced information, thereby enabling the use of this language for the development of virtual globes. However the great effort required in the programming of these globes, together with a growing evolution of 3D graphics that this language failed to follow, led to the extinction of VRML, and to the relatively small popularity of the virtual globes it allowed to develop [2]. This modeling language was eventually replaced by the X3D standard. An example of a globe implemented with the aid of this modeling language can be seen in image 2.2.

In 1996 starts the EarthBrowser project, developed as an application for the rendering of the planet based on ray-tracing algorithms. This application, while prior to the current concept of virtual globe, displayed some characteristics and features that would than be part of the definition of this concept. This application begins with the aim of simulating the formation and motion of clouds. Several features were subsequently added, such as high-resolution maps, possibility to drag and rotate the planet, and the capability of zoom in a specific area.

The concept of virtual globe as we know it today has its origin in the Digital Earth (DE) project, introduced in 1998 by the Vice President of the United States at the time, Al Gore. He proposes a virtual version of the globe characterized as a three-dimensional and multi-resolution representation of the planet, where there can be included large amounts of georeferenced information [15].

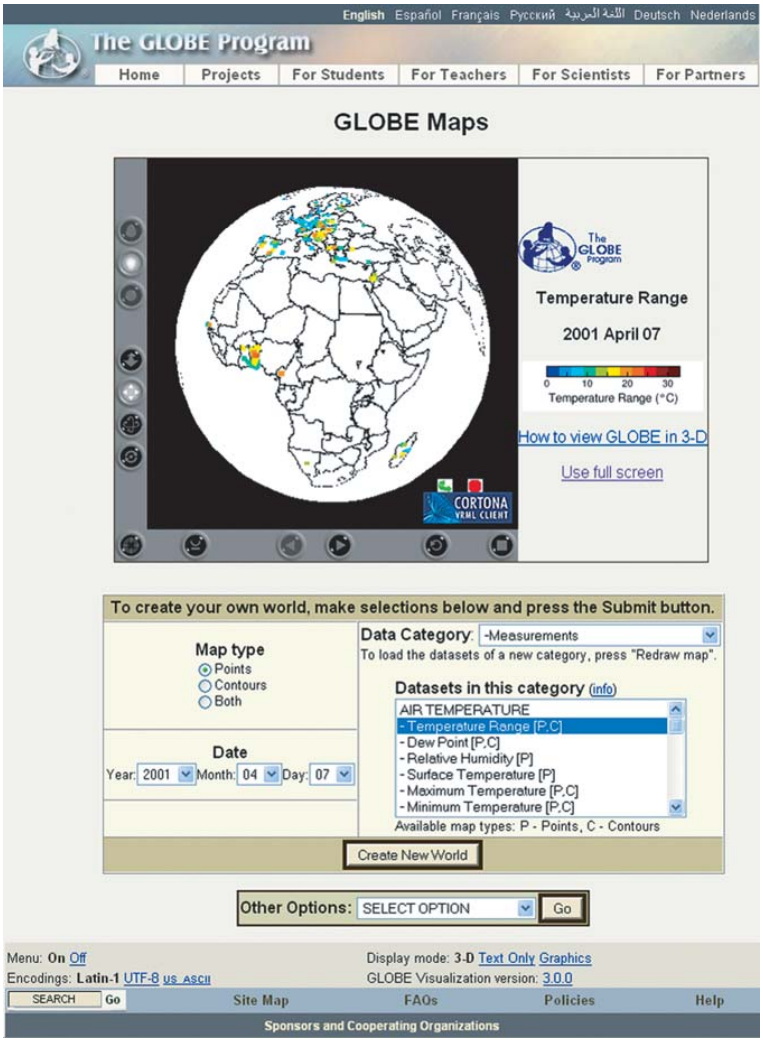


Figure 2.2: Example of a virtual globe implemented using the VRML language (source [2]).

The Digital Earth is then introduced with the purpose of assuming the role of point of connection between producers and consumers of this type of information. A virtual globe application, according to the concepts introduced, is composed by two main components. A navigable 3D viewer of the planet, available at various levels of resolution, and the mechanisms needed to integrate and present spatial information from various sources.

The concept of Digital Earth is, from its conception, idealized as a global project comparable to the Web. This comparison is established in the sense that

this project can not be of the responsibility of any organization or government, benefiting from the contribution of thousands of people from different places, cultures, and academic training. Thus enabling many possibilities for the community to contribute freely with geographic information through interaction with the virtual globe, giving rise to a knowledge base that can be used in a large number of scenarios. For example, giving aid to urban planning decisions, tracking the incidence of a virus, or even as a support tool in the response to natural disasters. Thereby, it becomes apparent the dual nature of this type of applications. One can use this type of solution for a casual querying of the topology of an area previous to an intended visit, and, at the same time, there is also the possibility to make requests of a highly technical and scientific nature, thus making it a powerful academic tool.

The first examples of an implementation of a virtual globe, according to the principles outlined in the Digital Earth project, were introduced by the encyclopaedia Microsoft Encarta Virtual Globe 1998 Edition [16] and by Cosmi's 3D World Atlas, released in 1999. In these applications the user was presented with a 3D model of the planet in which users can navigate the maps of various cities and view a set of multimedia content associated with certain locations. The image 2.3 displays the initial screen of the virtual globe provided by Microsoft Encarta.

In these early iterations of virtual globes implemented according to the concept of Digital Earth, developers proceeded to the implementation of local applications, both in its execution and in relation to the data consumed. However, these days the term virtual globe is most commonly associated with client applications that implement their consumption and rendering functions of large amounts of geographic information through interactions with various services through the Web. This type of applications can also be referred to as geobrowsers [17].

This change started taking place with the development of the virtual globe Earth 3D Viewer by Keyhole Inc. Introduced in 2001, during the 5th African GIS Conference in Nairobi, Kenya, it was the first virtual globe based on the consumption of information provided by a set of servers distributed globally. The visual effect produced by the presentation of a 3D model of the Earth composed from a

Figure 2.3: Initial screen of Microsoft Encarta's virtual globe solution (source [3]).

set of global coverage satellite images, combined with the possibility to apply the overlay of several types of relevant data representations to the display, originated an immediate impact with several governments representatives and journalists [4]. An image of this presentation is available in figure 2.4. Despite the significant popularity that this virtual globe benefited among organizations, especially among journalists, famously being used in the coverage of the invasion of Iraq in 2003, its acceptance by the general public never reached very significant levels.

The first online virtual globe to collect significant popularity among the general public was NASA World Wind, released in 2004. This virtual globe project is being developed by NASA in conjunction with the open source community and provides large amounts of spatial information for various planets and celestial bodies. Thus, in addition to fulfilling the role of geobrowser for the general public, presenting information in the public domain, it is also being used in scientific missions on land, sea and space. The open source nature of World Wind, allowing for the expansion and customization of the geobrowser through the development of custom plugins, led to a high growth of functionality and data availability. This

Figure 2.4: Presentation of EarthViewer, the first 3D geobrowser (source [4]).

characteristic openness to the community is responsible for the unique potential of this virtual globe to aggregate a multitude of geographic information, public and private, providing access to information from government institutions, industries, and the general public [18].

In 2005, after acquiring Keyhole Inc in the previous year, Google launches Google Earth, an updated version of the virtual globe application Earth 3D Viewer. This implementation is responsible for the extreme popularization of virtual globes, causing an increase to 10 times the previous level in the media coverage regarding to these type of application [19]. Several factors can be considered to explain this success such as the fame and proliferation of the Google search engine and also of the web mapping software Google Maps, where it was added an image layer from the database of Google Earth [2].

Since the introduction of Google Earth several other similar iterations of virtual globe applications were implemented. Some of the major examples are Bing Maps 3D, Marble and ESRI ArcGIS Explorer, virtual globe client of the ArcGIS

Server and leader in the professional Geographic Information Systems market.

The implementation challenges associated with the virtual globe concept are mainly associated with the large amount of information that its definition requires to process. Its characterization as a multi-resolution representation of planet Earth implies the possibility to view the world in a extremely large number of contexts. We can, for example, begin to see the planet from a point in space proceeding to seamlessly travel until we can view the street of our university. The processing and representation of all relevant information between these moments of visualization would demand unaffordable computing and storage requirements. Thus causing the need to define services that provide access and efficient handling of the necessary information to different viewing contexts.

2.3.1 Existing open source solutions

In our analysis of the existing open source virtual globes we choose to focus in a set of important factors. Primarily, the support given by said solutions to a set of relevant features, the programming languages and frameworks used, and the flexibility and expandability of the implementation.

In table 2.2 it is possible to see a comparison between the support to a set of relevant technologies offered by some major open source virtual globes.

Table 2.2: Level of support of each virtual globe application for a set of major features.

	WMS	**Raster**	**Vector**	**Elevation**	**3D models**
NASA World Wind Java SDK	full	full	full	partial	full
ossimPlanet	full	full	full	partial	missing
gvSIG 3D	full	full	full	full	full
osgEarth	full	full	full	full	full
Earth3D	missing	partial	missing	partial	missing
Google Earth	full	full	full	missing	full

As one can see, there is little difference between NASA World Wind Java SDK, gvSIG 3D, osgEarth, and Google Earth in terms of support to these fundamental features. Despite the fact that feature support is in fact the main point of consideration, when deciding which virtual globe to work with, often times, another important factor to consider is the APIs, programming languages and frameworks used in the implementation of each of these globes. Table 2.3 presents the technologies that support them.

Table 2.3: Technologies used for implementation, rendering, and GIS support by each virtual globe application.

	Language	**Rendering**	**GIS**
NASA World Wind Java SDK	Java	JOGL (OpenGL)	
ossimPlanet	C++	OSG (OpenGL)	ossim GDAL/OGR
gvSIG 3D	C++ Java	OSG/JOGL (OpenGL)	gvSIG GDAL/OGR
osgEarth	C++	OSG (OpenGL)	GDAL/OGR
Earth3D	C++/Java	OpenGL/ JOGL	
Google Earth	C++	OpenGL DirectX	

Considering the information presented in table 2.2 and 2.3 two main baselines were identified for our project. These baselines were NASA World Wind Java SDK and osgEarth.

NASA World Wind

Released in 2004, NASA World Wind was the first online virtual globe to collect significant popularity. This project [20] is currently developed by NASA in conjunction with the open source community and offers large amounts of spatial information for various planets and celestial bodies. Thus, in addition to fulfilling the role of geobrowser for the general public, representing information in the public

domain, it is also used in scientific missions on land, sea, and space.

NASA World Wind is available as an SDK implemented in Java. The rendering engine provided with this virtual globe is based on OpenGL using the wrapper library JOGL. The expansion of the solution is possible through a system of custom plugins.

osgEarth

Open-source virtual globe solution supported by Pelican Mapping. osgEarth [21] is implemented in C++ and is supported by OpenSceneGraph [22] for all its rendering needs.

This SDK provides a different way to display the terrain, in the sense that it does not require the user to build a 3D terrain model before he can display it. Instead, it will access provided raw data sources at application run time and composite them into a 3D map on the fly.

2.4 OGC Services

In order to satisfy the primary requirement of an efficient generation and access to geo-referenced data, a virtual globe client application normally resorts to the inclusion of a set of services specified by the Open Geospatial Consortium (OGC).

This organisation has as its main objective the development of standard specifications for the representation, access, and processing of geographic information and services [5].

The effort made in the specification and establishment of these standards is not made with the single goal of providing easy and efficient methods to manage and access geographic data. All standards specified and accept are also responsible in ensuring the interoperability between the various existing geographic information systems. Thus, the services specified by OGC assume a great importance and are in fact a central point for the functioning of these systems [17]. The standards specified by OGC that will assume greater importance in the context of the development of a virtual globe solution are, the Web Map Service (WMS) and the Web 3D Service (W3DS).

2.4.1 Web Map Service

The WMS specification characterises a service for the construction of maps of georeferenced information using as source a database of geographic data [23]. This specification is also responsible for the definition of the methods associated with the management of and access to these geographical maps. The service is implemented using an interface based on HTTP requests. These requests are made over a specific geographic area and layer of interest. The response to these requests is provided on the form of an image file that can be represented in a format based on pixels or vectors.

2.4.2 Web 3D Service

The recent specification proposal for the W3DS [24] characterises a set of methods and procedures to build, access, and collect georeferenced 3D scenes containing information about the terrain, textures and infrastructures. The geographic information produced by the service is delivered in the form of scenes composed of several graphic elements. The W3DS also strives for the optimisation of all scenes delivered, with the purpose of making possible the rendering of these scenes in real time with interactive frame rates.

The goals and premises defined in the design of this service imply that any implementation of this specification must be able to handle scenes with varying degrees of complexity without significant change in efficiency or response delay. From a full representation of the Earth to the individual lamps in a particular street, all requests must be handled without introducing too big of a delay in the system and all produced scenes must allow for an efficient rendering of its composing elements. This feature represents an interesting challenge, requiring the inclusion of information to multiple levels of detail for each object in order to maintain the possibility for an efficient rendering, without sacrificing quality.

The construction, handling, and delivery of these replies of higher complexity and size will demand the implementation of several optimised algorithms for the construction and delivery of each requested scene. These optimisations are mostly based in the restriction of the data handled and delivered in the reply ensuring that only the geographical area and level of detail specified in the request are con-

sidered [25].

The W3DS will then assume a similar role to WMS, despite the increased degree of complexity in the data delivered, in that it does not directly provide spatial information relevant to the application's request but a visual representation thereof. This representation is devoid of any information regarding the semantics or possible existing attributes associated with the geographic data delivered.

2.4.3 OGC Client Applications

There are several different premises and criteria to consider before initiating the design and implementation of a client application for OGC services. In figure 2.5 we can see the major types of client-server pairs that one can consider before deciding which solution better applies to the problem at hand.

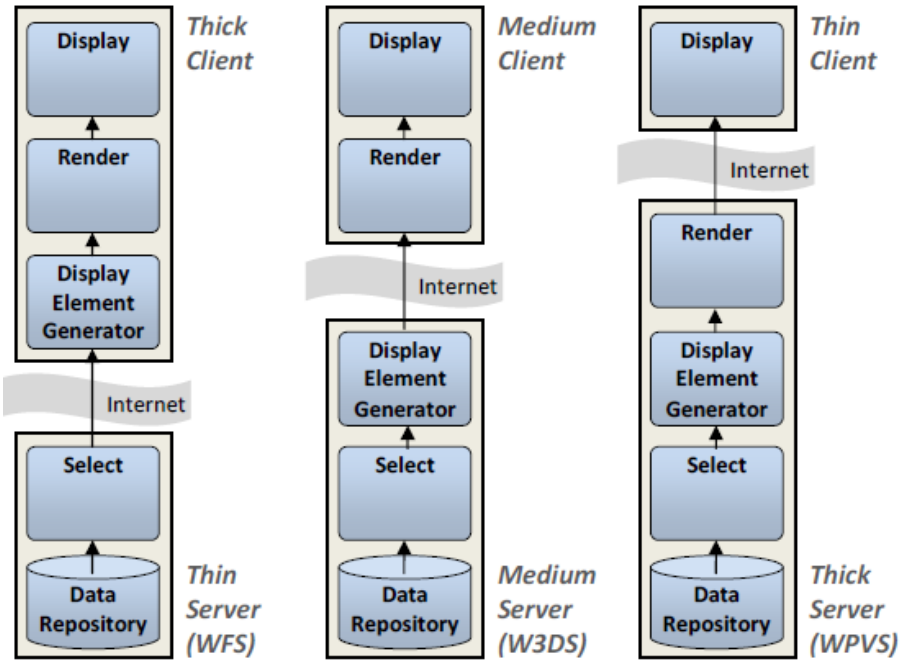


Figure 2.5: Different types of clients and servers when using OGC services (source [5]).

As we can see in the previous image, these clients can adopt a simpler configuration, leaving the vast majority of computational logic to the servers or to additional

layers of middle ware. Clients that assume this configuration are usually referred to as thin clients. An example of a client that usually follows this philosophy can be found in web mapping applications. These applications are characterised by the simple presentation of images, providing limited interaction with the elements present in the system [5]. The inclusion of a higher level of interactivity with existing features, and enabling the execution of complex operations on them, such as requesting that a given geometry be clamped to the ground, already implies a greater level of complexity to be present in the client application, giving origin to an intermediate solution.

In the particular case of applications such as a virtual globe, the high degree of complexity inherent to the data processed serves as a limiting factor in the choice of type of client to implement. The most pressing case of this complexity relates to the render of and interaction with the data provided by the W3DS, originating the need for much of the computational logic of the system to be present on the client side. Consequently, the client application ends up being responsible for the majority of the logic of the rendering system, as well as for all the methods necessary for any possible interaction with the rendered elements. In this iteration of the client-server pair, the primary role of the servers is then restricted to providing the answer to queries for information access.

Chapter 3

Architecture

In this chapter we will present the architecture of the propose solution for our project. We will begin by outlining the factors that will influence the design of any possible solution. The requirements enunciated in the genesis of the project are then presented. These requirements, as well as the general nature of the project, entail several specific challenges and difficulties that are than presented. Taking into account both the requirements and the main difficulties identified we present our approach to solving our problem as well as the main components of our solution.

3.1 Requirements

As was previously stated we propose the development of a virtual globe application capable of consuming the geographic services implemented in our system and capable of running natively in the Android operative system. In order to achieve this objective an analysis of the relevant existing solutions, presented in the previous section, was made. This analysis allowed us to identify and outline several specific requirements for the design of a possible solution, in addition to the general requirements related with the nature of this project (such as the requirement for the application to run in Android).

These requirements are as follows:

- The application implemented must run natively in the Android Operative

System;

- Common functionalities characteristic of virtual globes should be present (such as zoom, pan, drag, ...);
- The gestures associated with virtual globe solutions in mobile environments should be supported (for example: pinch to zoom, two finger drag to tilt, etc..);
- The implemented solution must be compatible with the geographic services previously developed, in particular the newly specified W3DS;
- Existing graphic models for 3D geometry must be supported. In alternative a method for an acceptable conversion to a supported model must be provided;
- The implemented solution must allow the exchange of the geographic services consumed in a simple and quick manner;
- The application developed must be flexible in terms of the geometry represented, allowing the user to enable and disable specific sets of entities when needed;
- The application should allow the selection of a particular model geometry presenting the relevant information inherent to the selected entity;
- The application must support the inclusion and presentation of models provided locally by the user.
- The solution developed in this project must be made available to the open source community, allowing its expandability in a simple manner and providing adequate documentation for the setup of the application.

3.2 Main difficulties

Along with the identification of the specific requirements that a virtual globe application implies, the analysis presented in the previous chapter permitted the identification of a set of major challenges associated with the proposed project.

Most of the difficulties identified for the implementation of this project and fulfilment of the requisites identified arise from the complexity associated with virtual globe solutions and from the development of this application for the mobile environment.

3.2.1 Data volume

The first major challenge associated with the implementation of a virtual globe application is the great amount of information that this type of solution handles at any given time. The fact that a virtual globe must provide an accurate representation of the Earth from a multitude of points of view, being it from space or at street level, implies the need for the efficient management of and access to significant quantities of data. Normally, this problem is addressed by the recourse to several standard services, being the applications responsible for the implementation of the necessary methods for the consumption of these services. Any implemented solution will also have to handle the high and constant throughput of information that the consumption and representation of the data provided by these services entails. The data that characterises the scene rendered is constantly changing putting a strain in the rendering engine and in the network.

3.2.2 Rendering challenges

The unusually large spectrum for which a virtual globe solution is suppose to provide an accurate visual representation of the Earth raises several problems that must be addressed in the implementation of the rendering engine of the application.

These challenges are mainly present in terms of:

Precision: the mere size of the Earth in addition with the possibility for the user to seamlessly zoom between various levels of detail may result in the temptation to use an unreasonably large coordinate space. However the use of large, single-precision, floating-point coordinates leads to z-fighting and jittering.

Accuracy: the common approximation of assuming the representation of the Earth as a perfect sphere enables several simplifications, however this representation

does not correspond to the truth, since in fact, the Earth has a bigger diameter across the equator than between the poles. Virtual Globe applications can not resort to this approximation, seeing as it introduces errors when positioning assets on the globe. Mathematical models to accurately model the Earth are than an important necessity.

Curvature: the curvature of the Earth also represents an issue to take into account, seeing as lines in a planar world are represented as curves on the Earth, possibly giving rise to oversampling near the poles.

Datasets: the high amount of data that each entity represented by the rendering engine can imply, is not compatible with the storage present in the GPU memory or even in a local hard drive. This entails the recourse to huge server-side datasets through which the data is accessed and paged during runtime, taking into consideration a set of view-parameters.

3.2.3 Mobile Environment

Perhaps the major challenge of this project is the fact that we are proposing the development of an application with the complexity of a virtual globe in the mobile environment. These type of devices are normally connected with several restrictions in terms of its processing and graphic power, available storage and network, and limited power supply. In fact, only very recently did the development of a virtual globe application in a mobile device became a possibility. All the issues so far identified in the process of implementing a virtual globe application become much more pronounced in the mobile environment and will thus imply further approximations and optimisations to produce an acceptable result.

The major areas of concern when implementing a virtual globe natively for a mobile device relate to:

Data storage: mobile devices are usually greatly limited in relation to the amount of storage available for use by an application. Be it physical or virtual memory the developer will have to adapt the application and algorithms to these constraints. The implementation of caching algorithms, for example, will

have to take this factor in consideration, introducing a new concern in the management of assets and entities.

Network: mobile devices are also limited in terms of the quality and availability of a network connection. Transfers of high amounts of data can introduce a great delay in the response of the system and are preferably avoided. This factor implies the inclusion of several parameters to filter the requests made to the network in order to ensure that the necessary relevant information is transferred while each request results in reasonably sized answers.

Rendering: the render of 3D graphics in mobile devices is limited by the fact that in these devices the support to common rendering methods is limited. In Android the standard API for graphic rendering, OpenGL, is only present as its subset OpenGL ES. This implies several constraints and limitations that will result in several adaptations to existing rendering methods.

3.3 Solution

The first step taken in order to solve the problem enunciated in chapter 1, and address the challenges and difficulties presented in this section, was a wide research of the open-source virtual globe solutions that exist currently for the desktop environment. This research was presented in section 2.3.1 and enabled us to further identify and understand the necessary steps and algorithms for the development of a virtual globe application.

We testified the high degree of complexity associated with the rendering engine for a virtual globe, also noted were all the geographic models and transformations that this application implies. Several dependencies to auxiliary libraries for coordinate transformation and geographic model computation were identified. Taking into account this high level of complexity and interdependence, starting an implementation of a virtual globe solution for the mobile environment from scratch would be a superfluous effort. Consequently we choose to take a desktop implementation of an open source virtual globe application and use it as basis for our project, proceeding to the execution of the porting process of said solution to the Android operative system.

3.3.1 Choice and implications

Taking into account the support offered by each solution, in addition to the technologies present in each virtual globe identified, we chose osgEarth as the base project for the implementation of our virtual globe solution. This choice is related to the greater efficiency associated with its implementation and with the flexibility it offers in the definition and consumption of new data sources and data types.

The recourse to osgEarth means that the implementation of the methods and algorithms necessary for the rendering engine and also for all the geographic transformations and models is no longer the major concern of the project. The main challenge is now the adaptation of these methods available in the desktop version of osgEarth to the mobile environment.

In order to achieve this porting process we begin with the cross compilation of the osgEarth framework and all relevant auxiliary libraries to the Android operative system. This cross compilation entails an initial preparation step where the constraints and limitations of the Android developing environment have to be met. As was previously stated, one of these limitations, and the one that will result in the major changes to the existing framework, is the obligation to use OpenGL ES for the rendering engine, instead of its desktop counterpart. Another factor to consider is the fact that Android NDK resorts to a non-standard `C` library, requiring some caution when compiling code that was not developed with this library in mind. For example, when compiling for Android a library that uses the method `isnan()`, one has to take into consideration that in Android's `C` library `isnan()` is still provided as a macro, as was the case in `C`'s standard version C99. This fact creates a conflict when the libraries intend to use `isnan()` as a method originating a compilation error. An appropriate possible solution is to alter the `c++config.h` file in the Android NDK include directory adding the definition `#define _GLIBCXX_USE_C99_MATH 1`, in order not to use this macro and similar others.

3.3.2 Integrating the osgEarth framework

Before initializing the process of porting osgEarth to Android it is necessary to analyse and identify the modules that make up this framework. This analysis is

performed with the objective of identifying the modules that will need to be altered in order to be able to execute the cross compilation to Android and to support all the functionalities needed to fulfil the requirements presented in section 3.1.

In figure 3.1 it is possible to identify the major components and modules that compose the existing desktop solution as well as the relations between each module. Highlighted with different background colours are the components and modules that were altered for this project.

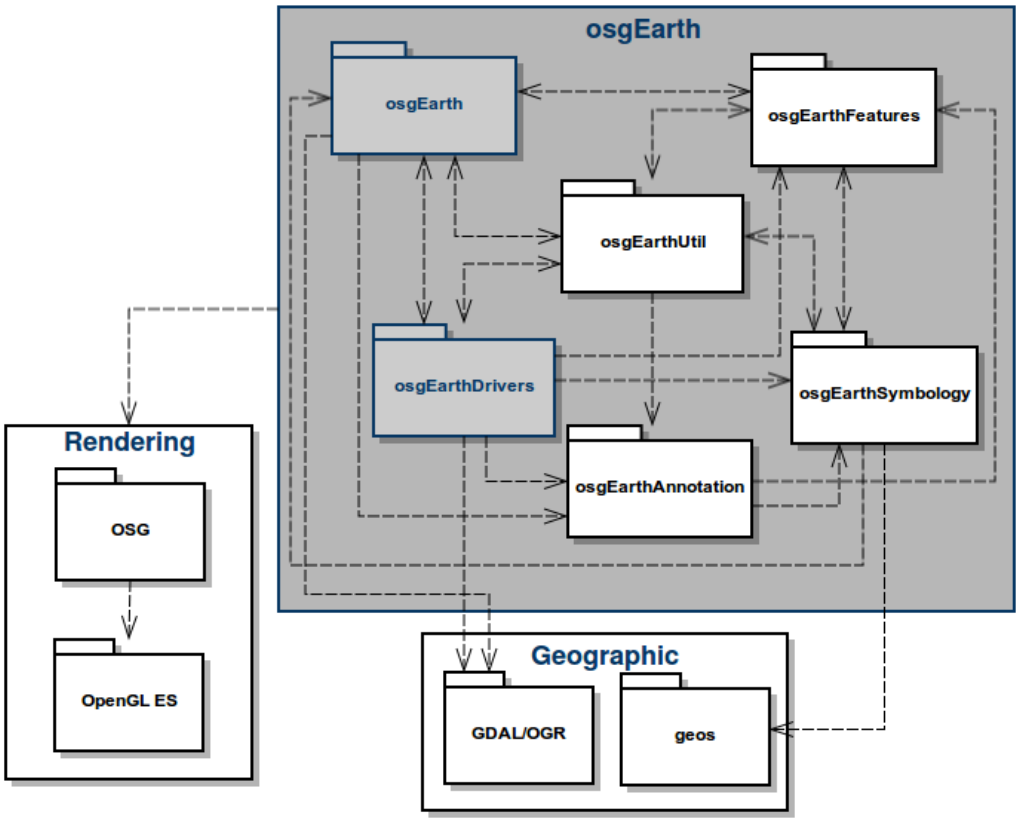


Figure 3.1: Components that compose the osgEarth framework. Components altered for our solution are highlighted with different background colours (original image).

As one can see in the presented diagram, we can divide the osgEarth framework in three major components. A primary component with all the modules that make up osgEarth, responsible for originating the globe and also for all the necessary methods to place additional graphic elements and information in spe-

cific locations on the globe. To achieve this general functionality these modules resort to two auxiliary components, the OpenSceneGraph (OSG) library, and a geographic component composed by the libraries GDAL/OGR and geos.

The component responsible for the rendering engine is then the OSG library. The rendering engine implemented by this library is based on the scene graph data structure and uses the OpenGL API. Seeing as there is already an OSG Android version, no alterations are necessary prior to compiling this library to Android.

The osgEarth framework calls upon GDAL/OGR in order to interpret and convert geographic data, producing a visual representation of the information contained. When there is a need to calculate the extent occupied by some specific data or to determine possible intersections or inclusions between data sets, osgEarth uses the geos library. The process of cross compiling these libraries is straightforward, seeing as they have no incompatibilities with the Android compilation environment. The cross compilation process used in this project is explained in subsection 4.1.1.

The major changes needed in order to enable the porting of the osgEarth framework to Android have to be made in the modules that compose osgEarth's core, specifically the package `osgEarth`. This package is responsible for setting up the globe model, handling the generation of the map, and managing image textures and terrain models. In this process several graphic nodes are created and added to the scene graph. This raises several possible incompatibilities in the cross compilation of this package for Android, seeing as the shader code that will then be responsible for the representation of these nodes is also composed in this package. Now, as was previously stated, in Android we only have access to the subset library OpenGL ES, the reduced support and functionalities of this library will then imply several necessary changes prior to the successful porting of this package. Subsection 4.1.2 presents the limitations of the OpenGL ES API and the necessary changes for a successful porting process.

The package `osgEarthDrivers` is responsible for the support to several different data types and data sources. Seeing as one of the main requirements of this project is the consumption of the W3DS service, a plugin for this service was developed and integrated into this package. The steps and concerns taken in the implementation of this plugin are explained in section 4.2.

With these steps we are able to port the osgEarth framework to Android and extend it in order to fulfil all the requirements enunciated for our project. Our Android virtual globe application implemented using the resulting extended ported framework is then presented in chapter 5.

Chapter 4

Development

In this chapter we will present the steps that were necessary to take in order to obtain a framework that would be able to solve the problem that was enunciated for this project.

With our choice to use an existing open source virtual globe solution as base for our project it became necessary to make sure that this solution would be available in our developing environment, the Android operative system. We will then present the necessary steps that need to be executed in order to set up the compilation process to build each necessary component into a version that is compatible with Android. We will also explain the changes that were made in order to overcome the limitations introduced by the Android environment, specifically in terms of the usage of the embedded systems specification of OpenGL.

Seeing as one of the major requirements of this project is the ability to consume a W3DS, feature not supported in the base framework chosen, a plugin for this data source is presented. This plugin takes into consideration several specific challenges associated with this service. The steps taken in order to overcome these challenges are also explained in this chapter.

4.1 Porting to Android

In order to be able to execute the porting process of the osgEarth framework to Android one has to overcome some constraints and limitations of the Android

build environment. As was previously stated, the major limitation in terms of support offered by this environment is related to the absence of the OpenGL graphic rendering API. The need to resort to the embedded systems specification of OpenGL will induce several incompatibilities. Only after these incompatibilities are resolved does it become possible to carry out the cross compilation process that will generate each Android compatible library.

4.1.1 Cross Compiling

The process of porting the osgEarth framework to Android also implicates the need to cross compile to this environment several auxiliary libraries to which the framework refers. Among the most significant cases, one can refer to the libraries geos, proj, and, most important, the library responsible for the rendering engine, OpenSceneGraph.

It is exactly through the an analysis of the OpenSceneGraph library that this process is initiated, given that Android versions of this library are already available. Thus, the method we followed for the porting of the necessary additional libraries was based on this existing porting of OpenSceneGraph.

This porting process is characterised by the definition of a set of `.mk` files, integrating part of the compilation for Android systems, that will be introduced in the `cmake` build environment. Through the use of a cross compiling process, made possible by the use of Android's `ndk-build` tool, libraries compatible with the Android system are then compiled.

Regarding the `.mk` files, the ones that arise with the utmost importance are the `Application.mk`, and the `Android.mk` file. The first is responsible for the definition of which modules will be included in the compilation. It also sets up the Android platform version for which the compilation will take place. In the listing 4.1 an example of one such file is present. This file is used in this project in the building process of Android compatible versions of the libraries geos and proj.

```
1 #ANDROID APPLICATION MAKEFILE
2 APP_BUILD_SCRIPT := $(call my-dir)/Android.mk
3 APP_PROJECT_PATH := $(call my-dir)
4
5 APP_OPTIM := release
```

```
6
7  APP_PLATFORM := 8
8  APP_STL := gnustl_static
9  APP_CPPFLAGS := -fexceptions -frtti
10 APP_CPPFLAGS := -Os -mthumb-interwork -fno-short-enums
11 APP_CPPFLAGS := -Wl,--no-undefined
12
13 APP_ABI := armeabi armeabi-v7a
14
15 APP_MODULES :=  geos proj sqlite3
```

Listing 4.1: Sample `Application.mk` file, used in this project to compile a set of auxiliary libraries.

Normally, the `Android.mk` file is used in order to set some global variables and include several auxiliary files corresponding to each of the modules that one wishes to build. In each such file there is present a listing of all source code files that define each compiled module. A portion of the file used in this project to build the Android compatible version of the library geos is presented in listing 4.2.

```
1  include $(CLEAR_VARS)
2
3  LOCAL_MODULE := geos
4  LOCAL_C_INCLUDES := \
5      $(GEOS_PATH)/src \
6      $(GEOS_PATH)/include
7  LOCAL_CFLAGS := \
8    $(LOCAL_C_INCLUDES:%=-I%) \
9    -DHAVE_LONG_LONG_INT_64
10 LOCAL_SRC_FILES := \
11     $(GEOS_PATH)/capi/geos_c.cpp \
12     $(GEOS_PATH)/capi/geos_ts_c.cpp \
13     $(GEOS_PATH)/src/algorithm/Angle.cpp \
14 (...)
15     $(GEOS_PATH)/src/simplify/TaggedLinesSimplifier.cpp \
16     $(GEOS_PATH)/src/simplify/TopologyPreservingSimplifier.cpp \
17     $(GEOS_PATH)/src/util/Assert.cpp \
18     $(GEOS_PATH)/src/util/GeometricShapeFactory.cpp \
19     $(GEOS_PATH)/src/util/Profiler.cpp \
```

```
20     $(GEOS_PATH)/src/util/math.cpp
21 include $(BUILD_STATIC_LIBRARY)
```

Listing 4.2: Portion of the file `geos.mk`, used in the project to build an Android version of the geos library.

Now, in order for the modules defined in these files to compile successfully it is necessary to take into consideration the existing differences and limitations of the Android build environment, making all the necessary alterations. These limitations are most common and more severe in the libraries that depend on OpenGL, which, as previously stated, is not present in Android. The changes that have to be made in order to adapt existing code to the subset library OpenGL ES, only version present in Android, are explained in the following subsection.

4.1.2 OpenGL ES support

As was previously stated, one of the main challenges in the process of porting the osgEarth framework to Android is related to the graphic rendering of 3D scenes. In the desktop version of osgEarth this rendering is done using the API for graphic rendering, OpenGL. However the Android operating system only has available the specifications of this API developed for embedded systems. Given the fact that these embedded specifications correspond only to a subset of the features available in OpenGL, some limitations and constraints that need to be addressed will become apparent.

For this work we chose to focus on the specification OpenGL ES 2.0. The reasons that led us to this choice are mainly related with the greater flexibility and efficiency ensured in this version, mainly by the access to the programmable rendering pipeline.

In order to be able to successfully compile all libraries with this version of OpenGL it becomes necessary to first identify which existing differences and limitations will involve making changes in our application, making then all the necessary alterations to the code.

Limitations and necessary changes

The main changes to take into account in our case are mostly related to the lack of features offered by the fixed function pipeline of OpenGL. In addition to this, there are several differences in methods and data types available to perform the rendering of geometric primitives.

Although OpenGL ES 2.0 has been chosen by us for the fact that it allows the access to a programmable rendering pipeline, through the use of vertex and fragment shaders, this specification has the problem of not being backward-compatible with the previous versions of OpenGL ES. With the introduction of the possibility to use these shader programs, the responsibility to implement the functionalities previously performed in the fixed-function pipeline is now completely delegated to the shader code. Consequently, the functions and data structures present in the fixed-function pipeline were totally removed in this version of the specification. Thus, fixed functions previously used to perform coordinate transforms, materials or lighting calls are no longer supported. This change triggered the application of very different shaders compared to the desktop version of osgEarth. In listings 4.3 through 4.6, it is possible to compare the code of the shaders needed for a simple application, which will apply the colours present on a texture to geometry, in each of these cases.

```
1 void main()
2 {
3     gl_Position = ftransform();
4     gl_TexCoord[0] = gl_MultiTexCoord0;
5     gl_FrontColor = gl_Color;
6 }
```

Listing 4.3: Vertex Shader code for application with access to the fixed-function pipeline

```
1 uniform sampler2D diffuseMap;
2
3 void main()
4 {
5     vec4 base = texture2D(diffuseMap, gl_TexCoord[0].st);
6     gl_FragColor = base * gl_Color;
```

```
}
```

Listing 4.4: Fragment Shader code for application with access to the fixed-function pipeline

```
varying mediump vec4 texCoord0;
varying mediump vec4 vColor;

attribute vec4 osg_Vertex;
attribute vec4 osg_Color;
attribute vec4 osg_MultiTexCoord0;

uniform mat4 osg_ModelViewProjectionMatrix;

void main()
{
    gl_Position = osg_ModelViewProjectionMatrix * osg_Vertex;
    texCoord0 = osg_MultiTexCoord0;
    vColor = osg_Color;
}
```

Listing 4.5: Vertex Shader code for application without access to the fixed-function pipeline

```
varying mediump vec4 texCoord0;
varying mediump vec4 vColor;

uniform sampler2D diffuseMap;

void main()
{
    mediump vec4 base = texture2D(diffuseMap, texCoord0.st);
    gl_FragColor = base * vColor;
}
```

Listing 4.6: Fragment Shader code for application without access to the fixed-function pipeline

It becomes apparent that the lack of methods and data structures available in the fixed-function pipeline will force the necessary information to be explicitly

stated in the shaders, being than passed through the application code to attributes and uniforms. The same precautions must be taken in order to perform the rendering of different materials and light effects, given that functions and data structures like `gl_LightSource`, `gl_FrontLightProduct`, `gl_FrontMaterial`, etc., are no longer available.

Another important change is related to the fact that in OpenGL ES it is not possible to group vertices using the nomenclature `Begin/End`, or the associated methods to specify individual information for each vertex. This mechanism must be replaced by passing a pointer to a buffer array with all vertices that will be represented. This pointer is then used in a function call to `DrawArrays` or in the case of specifying individual elements of the array, the function `DrawElements`. The fact that the function `DrawElements` in OpenGL ES is limited to the data types `unsigned byte` or `unsigned short` can also originate some problems. In our case it forced the modification of the 3D models serviced by our W3DS. These models, which were previously defined using the data type `unsigned int`, then had to be converted to a compatible version. Seeing as this W3DS service continues to serve clients in which this limitation is not present we decided not to change the data served to the other service consumers. In this sense the graphical models converted for use with OpenGL ES were made available by defining a new style on the server.

4.2 Extensions to osgEarth

In the design of the solution for our project we identified as a primary requirement the capability of our virtual globe application to consume data provided by a W3DS. The importance of this objective is related to the fact that it is only through the use of this service that it becomes possible to obtain a dynamic behaviour for our application in relation to the geographic areas represented. It is by its use that we will be able to render 3D data relevant to a particular geographical area without the need for any preparation steps specific for each location.

This requirement to consume a W3DS entails a well thought out solution, given that despite the already mentioned similarities with the WMS, the complexity of

the data processed by W3DS implies a set of more complicated challenges.

The amount of information that the representation of a standard graphic scene requires to transfer and render emerges as a major obstacle to overcome in an application that consumes the W3DS. Requests to the service have to be restricted only to the most relevant geographical areas. These requests also have to be managed in a way that the transfer of data packets becomes compatible with real-time rendering, without introducing large delays between the navigation to a relevant area and the representation of 3D scenes, thus keeping this process transparent to the user.

The complexity inherent to render these elements will require careful management in relation to the graphic elements to keep in scene. In the course of using the application several 3D scenes will be transferred and represented, the permanence of all these scenes in the rendering graph is not compatible with interactive frame rates. Consequently, it has to be found a balance between a realistic representation of the entire field of view presented in the virtual globe and the need for maintaining interactive frame rates.

4.2.1 W3DS plugin

To implement all the logic associated with the construction and render of the requests to the W3DS service we chose to develop an extension to the WMS plugin already present in the osgEarth framework. In order to support the increased complexity inherent to W3DS, special attention was given to the amount of data processed in each request.

For each layer of data, information is kept on which areas are already in memory, with the assistance of a auxiliary data structure. Thus, requests over areas already represented in, at least almost, its entirety are not carried out immediately. The level of detail in which each layer is active also deserved special attention so as to filter requests that would implicate an unbearable amount of data. A different solution for this problem would be to change the response logic of the W3DS server so that the level of detail would have an influence in the graphic scenes generated, filtering the entities included in each scene according to their relevance at that level of detail. This way the quantity of data passed in a response would

be filtered in the server according to the level of detail requested. Even if a request were made for an area expanse otherwise unbearable to the application only information corresponding to some, more important structures, would be sent.

In order to perform the management of the number of components present in the rendering graph, we resort to the auxiliary data structure referred to previously. This structure, in addition to indicate the geographical areas of each layer that are already represented in the scene, also stores a reference to the "father" node in the graph of the group of graphical components corresponding to that area. The management of the components in the scene thus consists in computing the geographical distance between the current view position and the centre of the bounding box that includes all the elements of a particular group. If this distance exceeds a certain threshold then the group is marked for removal, logging this operation in a changes queue.

With this plugin, any W3DS data source that we want to include in our application is easily integrated, being it during the implementation of the application or, preferably, in a `.earth` configuration file. Listing 4.7 displays an example `.earth` file where the layer `pt_postes`, with a W3DS data source, is added to a virtual globe application.

```
<map name="readymap.org" type="geocentric" version="2">

    <image name="readymap_imagery" driver="tms">
        <url>http://readymap.org/readymap/tiles/1.0.0/7/</url>
    </image>

    <image name="readymap_streets" driver="tms">
        <url>http://readymap.org/readymap/tiles/1.0.0/35/</url>
    </image>

    <image name="pt_postes" driver="w3ds">
        <url>http://webgis.di.uminho.pt:8080/geoserver2/sig3d/wms
           ?&styles=osg_poste</url>
      <layers>sig3d:SIG3D_POSTES</layers>
      <format>kml</format>
    </image>
</map>
```

Listing 4.7: Example of a W3DS data source being defined in a `.earth` file.

4.2.2 Caching 3D entities

Seeing as the consumption of a service like W3DS requires a significant effort in terms of network and processing, the use of a caching data structure assumes great importance. Especially in cases where several city infrastructures are represented. Structures like light and energy poles are spread in great number throughout a specific area, however, every instance of each of this structures is normally represented by the same 3D model. This implies some possible gains in the process of building the representing 3D scene in the application, since the graphic model for each type of structure only as to be transferred and processed once. Thus, despite the fact that each instance has different attributes and a different location, all the iterations of a model previously treated make use of a cached instance of the same 3D model. The transformations needed in order to place the model in each location are obtained by attaching a matrix node with all geographical transformations. The individual information and attributes necessary to correctly represent the new instance are also associated by attaching a node with the relevant specific information.

4.2.3 Changes to the scene graph

Another challenge that the consumption of the W3DS introduces and that our solution will have to address is the fact that said consumption implies constant changes to the set of graphical components represented in the scene. Now, since the rendering environment in this application is based on the scene graph data structure, special care is needed when applying any changes to it. It is necessary to ensure that the changes arising from the consumption of scenes served by the W3DS never occur during a rendering pass of the graph. This precaution is extremely important since changes to the structure of the graph during a rendering pass will most likely break said pass and lead to an application freeze or even to an actual crash of the application. To avoid this issue we implemented a queue

system to store records of impending changes to the scene graph.

In order to assist the decision making analysis performed in the changes queue, two flags were defined as attributes of each graphic node. These flags serve the purpose of indicating if said node is currently part of the rendering graph, and if its marked for removal or not. The state of these flags is altered when a new scene is consumed in the W3DS service, and when the node's distance to the viewing position is greater, or smaller, than a certain threshold. Based on these flags, a method was implemented to manage the state of the rendering graph. A node is added to the rendering graph if it is currently not part of it and is not marked for removal. The removal of a node occurs if it is marked for removal, and is currently part of the rendering graph. The method responsible for analysing the changes queue and make the implied changes to the rendering graph is then registered in a callback system that calls the function between rendering passes, ensuring the safety of each operation.

Thus, with the help of the developed attributes and methods, and using a callback mechanism, we are able to ensure that these changes are never made during a pass over the rendering graph.

Chapter 5

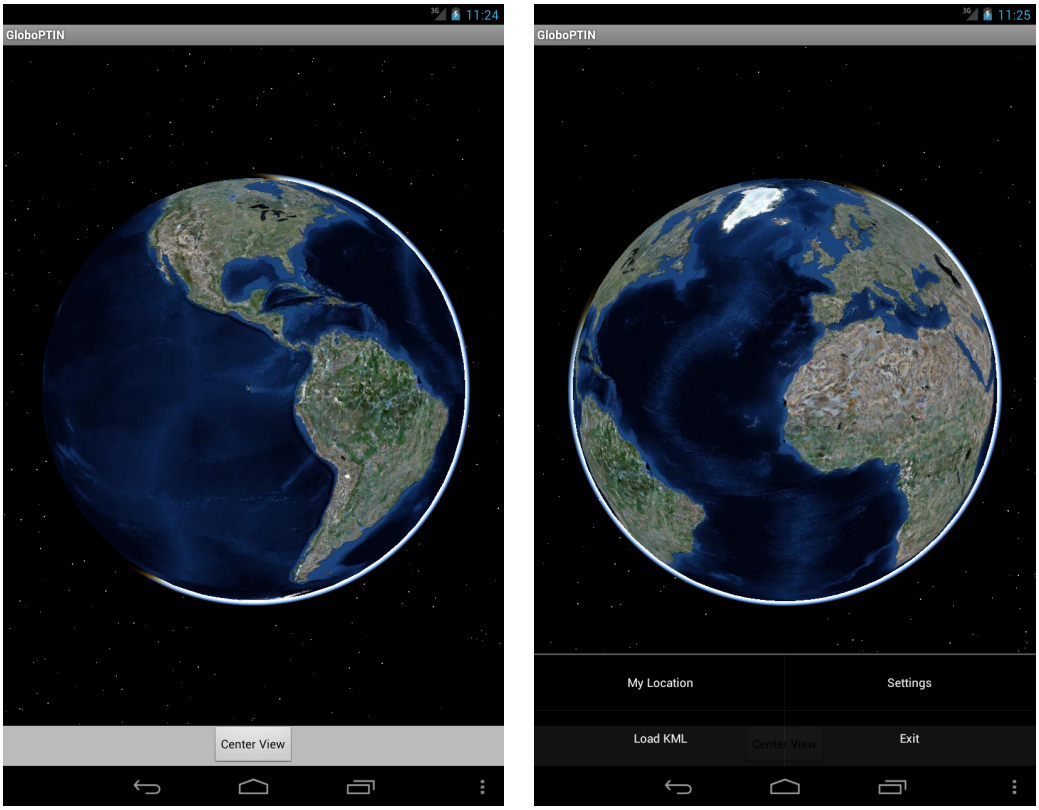
Results

In the previous chapter we presented the steps taken in order to port and extend the osgEarth framework. At the end of this process we obtained an Android compatible framework that provides all the necessary features and functionalities in order to develop a virtual globe application as the one we proposed in the goals for this project.

In this chapter we will then present the virtual globe application that was implemented with the aid of the developed framework.

The developed application initialises, as is common for virtual globe applications, with a view of the Earth from space. The user is than able to navigate the view with the set of typical gestures in the Android environment. The user is then able of zooming in or out, through the *pinch open* and *pinch close* gestures, move along the view, through the *swipe* gesture, and tilting the view, through a *two finger swipe*. The initial screen of our application is displayed in figure 5.1(a). In figure 5.1(b) its possible to see the main option menu made available to the user.

As is possible to see in the option menu, the user is capable of loading local `kml` files. The application will then process, place, and represent all the supported elements present in this file, moving the view in a fly-through animation to a view point relevant for the file. Another possible source of georeferenced graphical information is through W3DS. These services can be added to the application via source code or by editing a `.earth` configuration file.

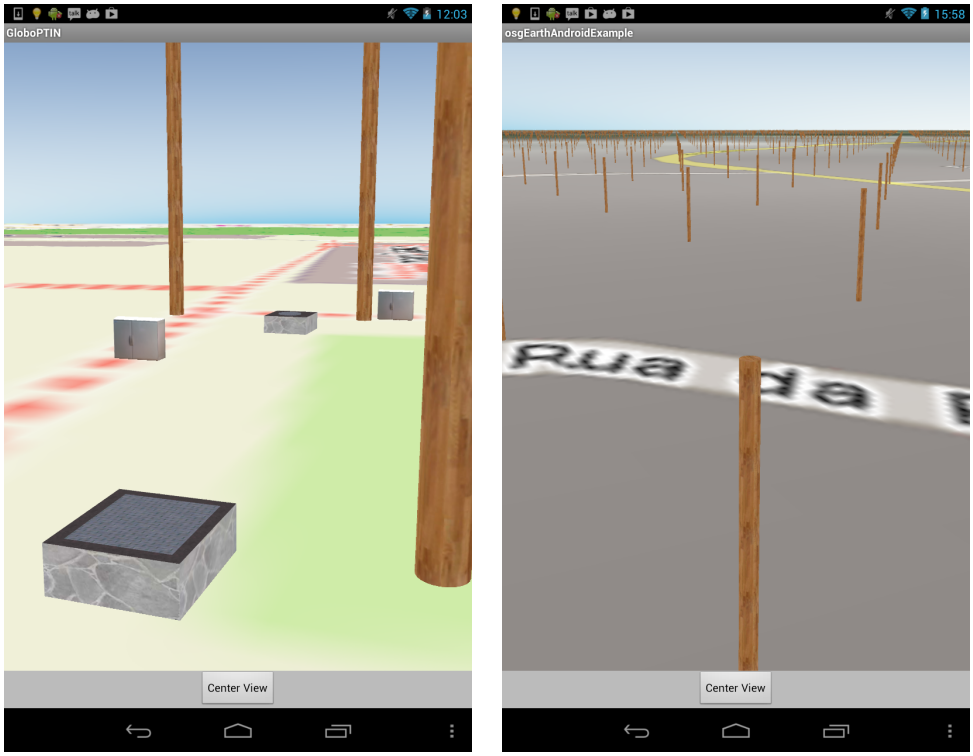


(a) Initial screen (b) Main option menu

Figure 5.1: Initial screen and option menu of our application.

In figure 5.2 we present a graphic scene containing several models of city infrastructure consumed from a W3DS. We also display a load test, loaded from a local `.kml` file, where several entities represented by the same graphic model rendered. With our caching system all these entities share the same graphic node while maintaining unique attributes.

The virtual globe developed also allows the user to select each individual represented model. This functionality can be used for several features. In the implementation of our application we used this functionality in order to apply different effects to the selected model or display individual attributes and information of the model. In figure 5.3 both these features are displayed.

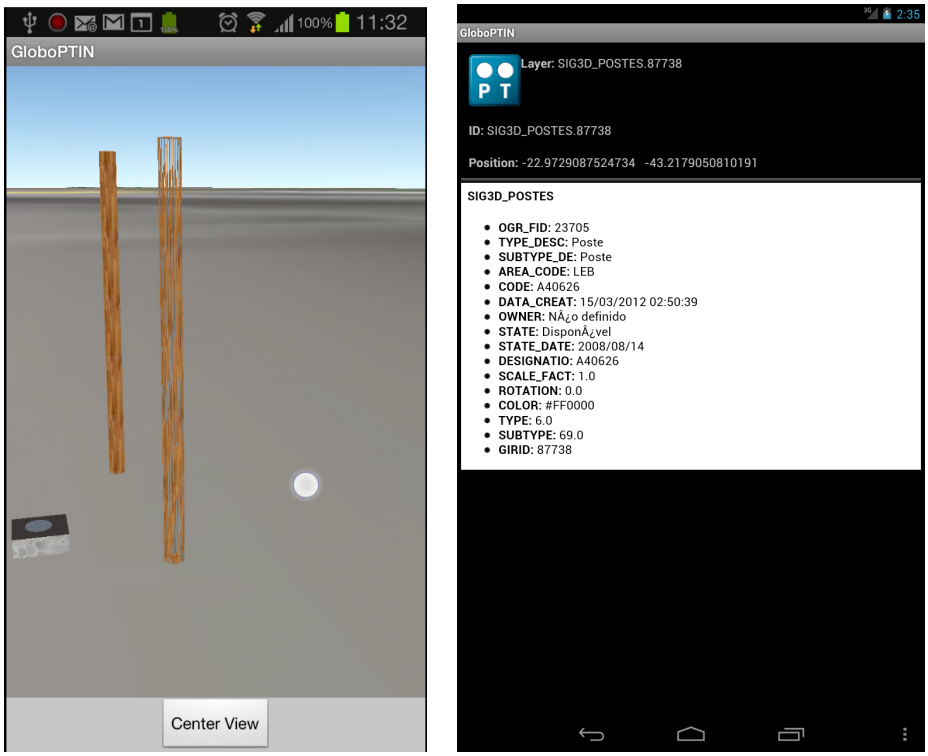


(a) Representation of city infrastructure (b) Load test from a local `kml` file

Figure 5.2: Example representation of several `kml` files served by a W3DS, and load test with several identical entities.

In order to provide greater control to the user in relation to the information represented in the application we implemented the possibility to change in runtime all the data sources present in the application. This way the user is capable of changing both the map image provider and to decide which layer of pre-defined W3DS data sources are visible in the viewer. Figure 5.4(a) displays the settings menu through which the user can access these options. In figure 5.4(b) the available pre-defined map image providers are presented.

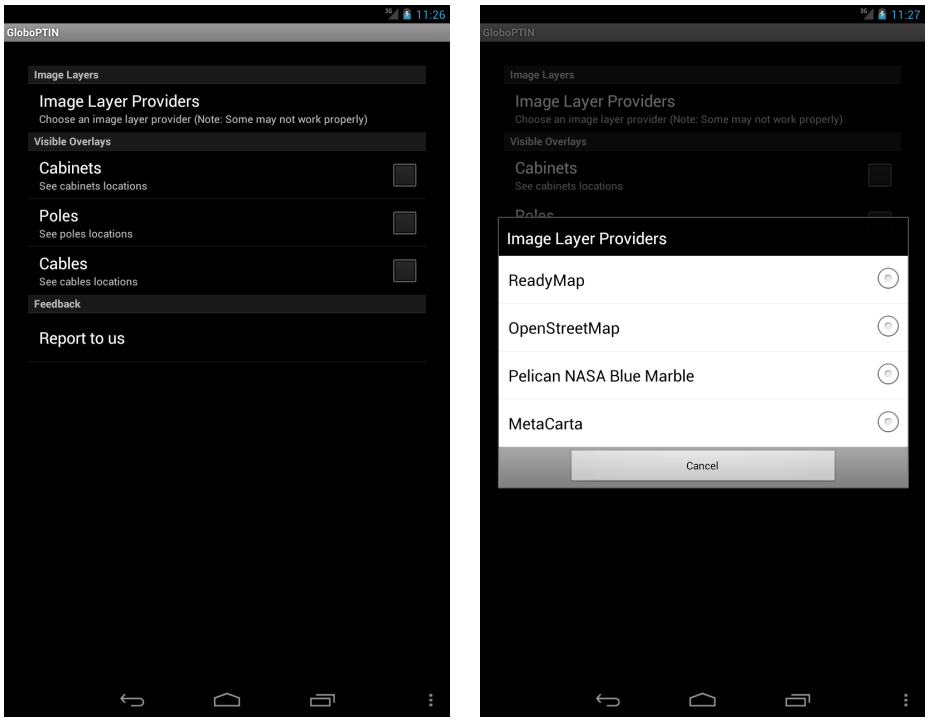
All the developed code is available in a BitBuckect repository. A manual explaining all necessary dependencies and build steps, through which the end results of this project can be replicated, is available in appendix A.



(a) Model with an edge effect obtained through the use of `GL_LINES`

(b) Information associated with a selected graphic model

Figure 5.3: Representation of a model with a different visual effect after selection and a view of the screen with the specific attributes of a selected model.



(a) Settings menu (b) Pre-defined image providers

Figure 5.4: Set of menus through which the user can personalise the graphic information displayed in the application.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

The overall objective of this thesis was the development of a powerful and expansible client for several geospatial data types and services. This client would have to consume the newly specified W3DS and be compatible with the Android operating system.

In the analysis of existing solutions relevant to our project the concept of virtual globe was quickly identified as the one that would imply the most attractive set of advantages in the resolution of our problem. However, further research resulted in the conclusion that the number of virtual globe implementations for the mobile environment was very limited. Being identified a complete absence of open source implementations, a necessary characteristic in order to develop new custom functionalities, compatible with our preferred mobile development environment.

We then set out to provide an open source solution for the development of virtual globe applications in the Android build environment. We choose to make use of an existing open source solution in the desktop environment as base for our project, making all the necessary changes in order to port all necessary components to Android. osgEarth was identified as the open source solution that best met the needs of our project.

We claim that we were able to port osgEarth to Android, making and documenting the necessary steps and alterations that this process entails. We identified

the recourse to the OpenGL API as the major obstacle to the porting process given the limited support that is present in the embedded specifications of this API. Because of this limited support, changes to the shader programs responsible for the 3D rendering where made thus obtaining an Android compatible rendering engine.

We also describe the advantages and difficulties inherent to the consumption of the W3DS. The process of expanding osgEarth in order to support this new data source and the steps taken to implement a new plugin to consume this service while overcoming said difficulties is also presented.

We were then able to obtain an open source framework for the development of Android compatible virtual globes. The process of porting existing similar libraries to the Android operative system, making all necessary preparations and alterations was also analysed and explained in this work.

With the developed framework we implemented a virtual globe application for the Android operating system. In addition to the most common features, our virtual globe offers the user an above average control over the data sources represented at all times and is able to integrate the newly specified W3DS.

The result product of this project is highly flexible and expandable and is available as open source, from a public BitBucket repository.

6.2 Future Work

In terms of the osgEarth framework ported in this project to the Android operative system, most of the possible future development is related with expanding the set of data types and data sources supported. In particular the collada `.dae` type has significant interest seeing as it is broadly used through several systems as the standard for the representation of graphic models. A detailed analysis of the performance given by the framework for various mobile device configurations identifying existing stress points and possible optimisations is also an area where further work should be made.

In terms of the virtual globe application developed, there is significant interest in implementing an offline mode, where the application would consume packages of pre-prepared areas giving to the user the possibility of using the application in locations where there is limited or non-existent network connections. In addition

to the offline mode, the possibility of editing the data served by the W3DS is also of great interest. Functionalities such as the creation of new entities or elimination of existing ones, and consequent alteration to the service database, and the possibility of editing the attributes associated with each entity, would further contribute to the usability of this solution as a major tool in geographical information systems.

Bibliography

[1] Google. (Accessed in October 1 2012) Activities. https://developer.android.com/guide/components/activities.html.

[2] B. T. Tuttle, S. Anderson, and R. Huff, "Virtual Globes: An Overview of Their History, Uses, and Future Challenges," *Geography Compass*, vol. 2, no. 5, pp. 1478–1505, Sep. 2008.

[3] DinSide. (2000, (Accessed in November 18 2012)) Encarta world atlas. http://www.dinside.no/18813/encarta-interactive-world-atlas.

[4] T. Foresman, "Digital Earth visualization and web-interface capabilities utilizing 3D geo-browser technology," *Proceedings of the 20th ISPRS Congress*, 2004.

[5] G. Percivall, C. Reed, L. Leinenweber, C. Tucker, and T. Cary, "OGC Reference Model," 2011.

[6] O. H. Alliance. (Accessed in August 18 2013) Alliance overview. http://www.openhandsetalliance.com/oha_overview.html.

[7] L. Batyuk, A.-D. Schmidt, H.-G. Schmidt, A. Camtepe, and S. Albayrak, "Developing and benchmarking native linux applications on android," in *MobileWireless Middleware, Operating Systems, and Applications*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, J.-M. Bonnin, C. Giannelli, and T. Magedanz, Eds. Springer Berlin Heidelberg, 2009, vol. 7, pp. 381–392. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01802-2_28

[8] Google. (2009, (Accessed in October 7 2012)) Introducing android 1.5 ndk. http://android-developers.blogspot.pt/2009/06/introducing-android-15-ndk-release-1.html.

[9] D. Wolff, *OpenGL 4.0 Shading Language Cookbook*. Packt Publishing Ltd., 2011.

[10] A. Munshi, D. Ginsburg, and D. Shreiner, *OpenGL ES 2.0 programming guide*. Pearson Education, Inc., 2008.

[11] J. Forshaw. (2011, (Accessed in March 19 2013)) Webgl - a new dimension for browser exploitation. http://www.contextis.com/research/blog/webgl-new-dimension-browser-exploitation.

[12] ——. (2011, (Accessed in March 19 2013)) Webgl – more webgl security flaws. http://www.contextis.com/research/blog/webgl-more-webgl-security-flaws.

[13] cartographic images.net. (Accessed in November 18 2012) Behaim globe. http://cartographic-images.net/Cartographic_Images/258_Behaim_Globe.html.

[14] M. Pesce, *VRML: Browsing and Building Cyberspace*, 1995.

[15] A. Gore, "The digital earth," *Australian surveyor*, no. February, pp. 41–44, 1998.

[16] M. N. Center. (1997 (Accessed in February 5 2013)) Now a virtual globe, not just a world atlas. https://www.microsoft.com/en-us/news/press/1997/nov97/vglobepr.aspx?navIndex=1.

[17] M. Gould and M. Craglia, "Next-generation digital earth: A position paper from the vespucci initiative for the advancement of geographic information science," *International Journal of Spatial Data Infrastructures Research*, vol. 3, pp. 146–167, 2008.

[18] D. Bell, F. Kuehnel, and C. Maxwell, "NASA World Wind: Opensource GIS for mission operations," *Aerospace Conference, 2007 IEEE*, 2007.

[19] T. G. Web. (2007 (Accessed in February 5 2013)) Media coverage of geospatial platforms. http://www.geospatialweb.com/figure-4.

[20] NASA. (2007, (Accessed in October 2 2012), Jul.) World wind java sdk. http://worldwind.arc.nasa.gov/java/.

[21] P. Mapping. (Accessed in October 5 2012) osgearth. http://osgearth.org/.

[22] R. Wang and X. Qian, *OpenSceneGraph 3 Cookbook*. Packt Publishing Ltd., 2012.

[23] J. L. Beaujardiere, "OpenGIS Web Map Server Implementation Specification," 2006.

[24] A. Schilling and T. H. Kolbe, "Draft for Candidate OpenGIS® Web 3D Service Interface Standard," 2010.

[25] G. Misund and M. Granlund, "Global models and the w3ds specification - challenges and solutions," in *Accepted to First International Workshop on Next Generation 3D City Models*, 2005.

Appendix A

Build Manual

In this appendix we present a manual portraying all the steps necessary in order to obtain and build the solution developed in this project, in order to replicate the obtained results or to further expand them.

Virtual Globe compilation for Android

May 2013

Contents

Introduction

The virtual globe developed in this project is implemented as an Android application (.apk). This application is written mostly in C++. From this C++ code a library, libosgNativeLib.so, is built and than utilized in the generation of the final .apk.

Requirements

Besides the Android SDK, the setup of this project needs the Android NDK framework. In this project the version r8b of this framework was used. To install the Android NDK its necessary to execute the following steps:

```
cd Android
wget https://dl.google.com/android/ndk/android-ndk-r8b-linux-x86.tar.bz2
```

After this installation, its necessary to define two environmental variables:

```
export ANDROID_NDK=/home/jgr/Android/android-ndk-r8b
export ANDROID_SDK=/home/jgr/Android/android-sdk-linux
```

The Android folder used is then where both the SDK and the NDK frameworks are installed.

The compilation of this project is supported by the cmake tool, used to generate the necessary Makefiles. To compile this project the version 2.6.4, or higher, of cmake is necessary.

Obtaining the source code

The necessary source code can be obtained from following repository:

https://bitbucket.org/jgrocha/osgearthandroid

To download the code the following steps should be taken:

```
cd
git clone git@bitbucket.org:jgrocha/osgearthandroid.git
cd osgearthandroid/
```

In this manner, all the code will be in the folder osgearthandroid.

Compilation of OpenSceneGraph

The compilation of OpenSceneGraph depends on several auxiliary libraries. So, before the compilation process one should download these libraries into the appropriate folder.

```
cd OpenSceneGraph
wget http://www2.ai2.upv.es/difusion/osgAndroid/3rdpartyAndroid.zip
unzip 3rdpartyAndroid.zip

cmake . -DOSG_BUILD_PLATFORM_ANDROID=ON -DDYNAMIC_OPENTHREADS=OFF -DDYNAMIC_OPENSCENEGRAPH=OFF
-DOSG_GL1_AVAILABLE=OFF -DOSG_GL2_AVAILABLE=OFF -DOSG_GL3_AVAILABLE=OFF -DOSG_GLES1_AVAILABLE=OFF
-DOSG_GLES2_AVAILABLE=ON -DOSG_GL_LIBRARY_STATIC=OFF -DOSG_GL_DISPLAYLISTS_AVAILABLE=OFF
-DOSG_GL_MATRICES_AVAILABLE=OFF -DOSG_GL_VERTEX_FUNCS_AVAILABLE=OFF
-DOSG_GL_VERTEX_ARRAY_FUNCS_AVAILABLE=OFF -DOSG_GL_FIXED_FUNCTION_AVAILABLE=OFF
-DANDROID_ABI="armeabi armeabi-v7a" -DANDROID_PLATFORM=8 -DANDROID_STL="gnustl_static" -DJ=4

make
```

Compilation of osgEarth

This compilation is executed in two steps.

Compilation of all dependences to Android

```
# Move into osgearthandroid/3rdparty/jni
cd ../3rdparty/jni
~/Android/android-ndk-r8b/ndk-build
```

Compilation of osgEarth to Android

```
# Move into osgearthandroid
cd ../..
cmake . -DOSG_BUILD_PLATFORM_ANDROID=ON -DJ=4 -DOSG_DIR="./OpenSceneGraph" -DDYNAMIC_OSGEARTH=OFF
-DOPENTHREADS_LIBRARY="./OpenSceneGraph/obj/local/armeabi/libOpenThreads.a"
-DCURL_LIBRARY="./OpenSceneGraph/3rdparty/build/curl/obj/local/armeabi/libcurl.a"
-DGDAL_LIBRARY="./osgearthandroid/OpenSceneGraph/3rdparty/build/gdal/obj/local/armeabi/libgdal.a"
-DGEOS_LIBRARY="./3rdparty/obj/local/armeabi/libgeos.a"
-DSQLITE3_INCLUDE_DIR="./3rdparty/jni/sqlite-autoconf-3071401"
-DSQLITE3_LIBRARY="./3rdparty/obj/local/armeabi/libsqlite3.a"

make
```

Generation of osgViewer

The generation of the Android application is made in two steps. First we build the library libosgNativeLib.so and than the osgViewer.apk.

Generation of the library libosgNativeLib.so

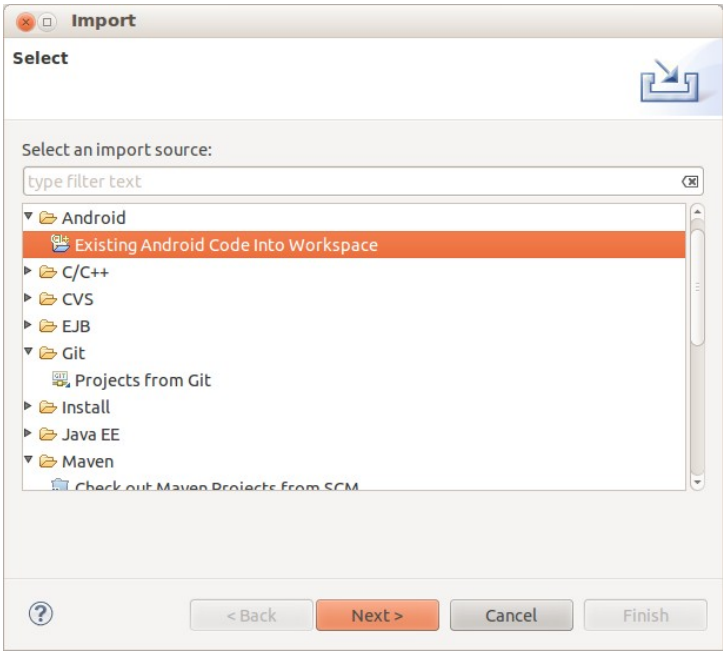
```
cd osgViewer/jni
~/Android/android-ndk-r8b/ndk-build
```

The makefile includes two debug instructions that present the computed PATH. The output of the execution of this makefile (via ndk-build) should be similar to the following:

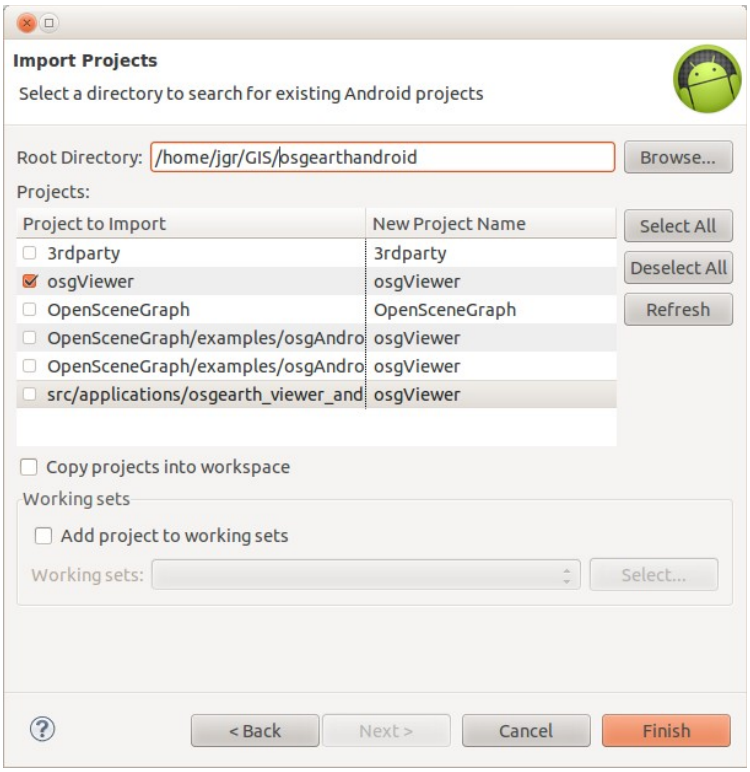
```
value of LOCAL_PATH is: /home/jgr/GIS/osgearthandroid/osgViewer/jni
value of OSGEARTH_ANDROID_DIR is: /home/jgr/GIS/osgearthandroid/osgViewer/jni/../..
Gdbserver      : [arm-linux-androideabi-4.6] libs/armeabi/gdbserver
Gdbsetup       : libs/armeabi/gdb.setup
Install        : libosgNativeLib.so => libs/armeabi/libosgNativeLib.so
```

Generation of osgViewer.apk

After opening Eclipse IDE, go to File → Import..., choose the option “Existing Android Code Into Workspace”, as the following image illustrates.



In the following dialog, its necessary to point out the folder with the source code (osgearthandroid) and select the project osgViewer, as illustrated.



After the project is imported into Eclipse we should be able to execute it. The generated apk is localized in osgViewer/bin/osgViewer.apk.