An Open Source Virtual Globe Implementation for Android

Paulo Dias Almeida
Universidade do Minho
Braga, Portugal
diasalmeida.paulo@gmail.com

Jorge Gustavo Rocha
Universidade do Minho
Braga, Portugal
jgr@di.uminho.pt

ABSTRACT

A virtual globe is a 3D software representation of the Earth, providing an intuitive interaction. The user can freely move around by changing the angle, the position and the distance to the ground. The interface also provides the ability to choose what to see on the globe surface (or near the surface). The user can switch between geographic features, natural or human-made features, abstract features like administrative boundaries, or any other data that can be represented on the earth's surface. This data can be download and rendered on top of the globe at run time.

Virtual globes have a number of key benefits as a platform for communicating and visualizing geospatial data over traditional mapping technologies, since they provide as true a representation of the world as possible, without the usual limitations and deformations of projections.

Despite these advantages, a complete, well supported, open source virtual globe solution is still missing for mobile platforms.

In this paper we discuss the implementation of a virtual globe for mobile platforms. After an initial survey to identify the best possible virtual globe technology, we selected an existing desktop project to port to the mobile platform. We discuss the requirements and challenges of this virtual globe implementation. The architecture and the implementation decisions are presented.

Since one of the major features of a virtual globe is the ability to get data at run time from different services, we also show how this virtual globe can be extended to consume data from a Web 3D Service. In the same way, this globe can be extended to consume other services.

1. INTRODUCTION

There is growing interest in the visualization of and interaction with geographic information. Recent years have seen the appearance and popularization of a large number of solutions where geospatial and georeferenced data plays a key role. Consequently, several efforts have been made to develop 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 visualization and interaction of this data became an obvious priority.

The appearance of virtual globe applications greatly contributed to the increased popularity of geographic solutions and systems among the general public. 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 geographic information on top of a virtual globe that mimics the earth surface. Topography information and terrain elevation, or 3D buildings and city infrastructures 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 combined with weather data. All this information can be explored in a uniform manner.

This growing interest on geographic applications resulted in many applications for all platforms, but in particular, for mobile devices. Smartphones also have native devices for positioning, so even more applications were built for these devices. There are many applications exploring geographic information and its positioning capabilities, but few virtual globes.

Mostly because until a few years ago this kind of application was almost impossible to provide a good user experience. The limited capabilities associated with these devices meant that applications such as a virtual globe would have an implementation in mobile devices that was either impossible or extremely limited.

However, in recent years we have witnessed a great increase in processing power, storage, and graphic capabilities of mobile devices, making the difference from a desktop plat-

form progressively less significant. This opens the possibility for the development of applications and functionalities that have, so far, been impossible to implement in mobile devices. With these increased hardware capabilities, it became feasible to develop a virtual globe with a good performance, without removing functionalities and aggressive simplifications, and thus providing a good user experience.

In this paper we focus on the development of an open source virtual globe solution for the Android operating system. It is not an open source project in the limited sense that the source code is available. It is an open source project where others can participate taking advantage of the git features. Public git based open source projects really changed the way we interact with open source projects. We don't need to write a private email to the author to request a feature or to report a bug. Every interaction is visible to all (no need to report the same bug twice) and users can actively provide new features, code improvements, etc. by issuing pull requests in the git terminology.

This papers is structured as follows. We start by the oldest known globes to the very best state of the art virtual globes. From the state of the art open source solutions, we selected an existing project that meets all functional requirements. Since we were lucky to find an exiting open source solution for the desktop to start with, our methodology focused on the porting process. We describe in detail the challenges we faced by porting a large application made for the desktop to the mobile platform. Finally, we provide a real example of how to extend the mobile virtual globe to consume data from a new service. This exercise is presented to help others to enhance the virtual globe with new features. We close the paper with our closing remarks and outline for future work.

2. VIRTUAL GLOBES

The most intuitive and descriptive representation of Earth is the globe. Early terrestrial globes emerged following the establishment of the sphericity of the planet. Martin Behaim's Erdapfel [3], created in 1492, is the oldest surviving example to modern times.

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) [9]. This language was introduced in 1994 and had as its main goal the representation of 3D animated worlds via the Internet. With the introduction of GeoVRML in 1998, the language is 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 programming these globes, together with a growing evolution of 3D graphics that this language failed to follow, led to the decline of VRML, and to the low popularity of the virtual globes it allowed to develop [11].

The concept of virtual globe as we know it today has its origin in the Digital Earth (DE) project, introduced in 1998 by Al Gore, Vice President of the United States. He proposes a virtual version of the globe characterized as a three-dimensional and multi-resolution representation of the planet, where large amounts of georeferenced information can be included [5]. The DE is then introduced with the purpose of assuming the role of being a bridge between producers and consumers of this type of information. A virtual globe application, according to the concepts introduced, is composed of 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.

Early examples of virtual globes following these concepts, were introduced by the Microsoft Encarta Virtual Globe encyclopaedia 1998 Edition [4] 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 he could navigate the maps of various cities and view a set of multimedia content associated with some locations. In these early iterations implemented according to the concept of DE, developers proceeded to the implementation of local applications, both in execution and in relation to the data consumed. However, nowadays, the term virtual globe is most commonly associated with client applications that consume large amounts of geographic information through interactions with various Web services. This type of application is also referred to as geobrowser [6].

This change started taking place with the development of the virtual globe Earth 3D Viewer by Keyhole Inc. Introduced in 2001, it was the first virtual globe based on the consumption of information provided by a set of servers distributed globally.

Despite the significant popularity of this virtual globe 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 gain significant popularity among the general public was NASA World Wind, released in 2004. This 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, 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 sharp growth of functionality and data availability. This 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 [2].

In 2005, after acquiring Keyhole Inc, Google launched Google Earth, an updated version of Earth 3D Viewer. This implementation is responsible for the extreme popularization of virtual globes, causing a 10-fold increase from the previous level of media coverage for this type of application [12]. Several factors can be considered to explain this success, such as the fame and proliferation of the Google search engine and the web mapping software Google Maps [11].

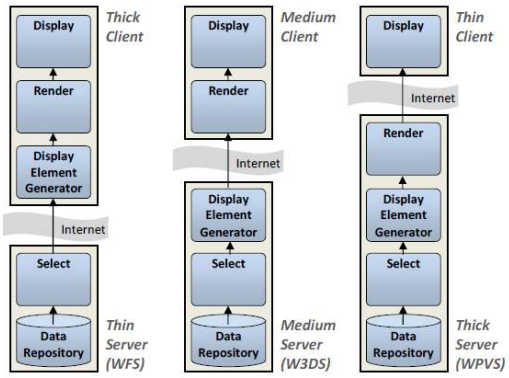

Figure 1: Thin vs. thick clients for portraying features over the Internet.

3. CHALLENGES AND OPPORTUNITIES

The development of a virtual globe solution capable of consuming a diversity of geographic services and data, running natively on the mobile phone presents some software engineering challenges.

3.1 Thick clients for portraying features

A virtual globe is considered as a thick client for portraying features, according to the OGC reference model [8].

Client models are depicted in Fig. 1. Thick clients must handle not only the display, but also the rendering and styling of 3D features.

For example, a virtual globe is expected to: (i) retrieve some 2D GML features from a server; (ii) assign the correct height to the features relative to the ground; (iii) render them using some style and (iv) portray then according to the visibility rules, depending on the users view point. This illustrates how thick clients are far more complex than web mapping applications.

3.2 Virtual globe implementation challenges

Two main challenges are specific to the implementation of virtual globes: data management and rendering issues.

3.2.1 Data formats and volume

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, where the application is responsible for the implementation of the necessary methods for the consumption of these services. Any implemented solution will then have to handle the high and constant throughput of information that the consumption and representation of the data provided by these services entails.

3.2.2 Rendering

The unusually large spectrum for which a virtual globe solution is supposed 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. Several issues related to the precision and the accuracy of what is rendered result from this broad scale of representation. The mathematical and geographical models needed in the representation of the planet and geolocation of assets is also a major concern for the rendering engine.

3.3 Mobile Environment

These specific challenges related with data and rendering are even more important in the mobile world. Mobile devices are associated with many constraints in terms of processing and graphics power, available storage and network bandwidth, mostly due to battery constraints.

Data transfer and management in mobiles devices is even harder. But perhaps the major challenge in this environment is related to the rendering engine. Rendering 3D graphics in mobile devices is limited by the fact that they have limited support to common computer graphics rendering application programming interfaces (API), like OpenGL.

In Android for example, only a subset of OpenGL is available. This smaller API is called OpenGL for Embedded Systems (OpenGL ES). With less and more primitive routines, the rendering is harder to develop.

3.4 Open source solutions

Before taking any decision on how to develop an open source virtual globe, we decided to do a survey of existing solutions for other platforms, to identify the best possible virtual globe technology.

In the table 1 we present the solutions surveyed. For each solution, we indicate whether each feature is supported or not.

The table 1 shows small differences between World Wind, gvSIG 3D, and osgEarth in terms of functionalities. This is a good sign and an indicator that all of them are mature solutions.

With so many virtual globe's key functionalities already implemented in different open source projects, we decided not to develop a mobile solution from scratch.

In order to select one of these projects to use we also compared other important factors besides its features, like APIs, programming languages, and frameworks used in the implementation of each of these globes. Table 2 presents the technologies that support them.

3.5 Our Approach

The previous sections allowed us to identify a significant set of challenges associated with the implementation of a virtual globe. Some of these challenges result in the use of auxiliary libraries for rendering, coordinate transformation, and geographic model computation. 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 chose to take an existing desktop open source virtual globe solution and use it as basis for our project, proceeding to the execution of the porting process.

Table 1: Level of support of each virtual globe solution for a set of major features.

	WMS	Raster	Vector	Elevation	3D models
NASA World Wind Java	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

Table 2: Technologies used for implementation, rendering, and GIS support by each relevant virtual globe application.

	Language	Rendering	GIS
NASA World Wind Java	Java	JOGL (OpenGL)	
gvSIG 3D	C++ Java	OSG/JOGL (OpenGL)	gvSIG GDAL/OGR
osgEarth	C++	OSG (OpenGL)	GDAL/OGR

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 basis for the implementation of our 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.

We also chose the Android operating system to develop our virtual globe, instead of other closed source environments, since it is the most adopted OS in the mobile world. According to IDC[1], Android market share was around 80% by the end of 2013.

4. DEVELOPMENT

Considering our choice to use an existing desktop solution as the basis for our project, it became necessary to make sure that this solution would be available in our developing environment, the Android operating system. We 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.

4.1 Porting Process

In order to execute the porting process we begin with the cross compilation of the osgEarth framework and all relevant auxiliary libraries to the Android operating 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 width 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` standard version C99. This fact creates a conflict when the libraries intend to use `isnan()` as a method, originating a compilation error. An appropriate 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 other similar ones.

The porting process is characterized by the definition of a set of `.mk` files, defining the compilation rules 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. The specific steps taken for this process are further explained in [1], and can be seen in the project repository.

4.2 OpenGL

OpenGL's limited usage in Android demanded the most significant changes in the porting process.

Given the fact that the embedded specification, available in Android, corresponds only to a subset of the features available in OpenGL, some limitations and constraints must be overcome. The main changes to take into account 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 allows access to the programmable rendering pipeline, through the use of vertex and fragment

[1] `http://www.idc.com/getdoc.jsp?containerId=prUS24442013`, consulted in Jan, 2014.

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 of implementing 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.

Another important change is related to the fact that in OpenGL ES there isn't the possibility of grouping 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` this can also originate some problems. In our case it forced the modification of the 3D models used in our data services. These models, which were previously defined using the data type `unsigned int`, then had to be converted to a compatible version.

5. RESULTS

We were able to implement a virtual globe on Android, based on osgEarth, as described.

The application starts with a view of the Earth from space, as usual in virtual globe applications. The user is then able to navigate using the typical gestures in touch based interfaces.

The user can zoom in or out, using the *pinch open* and *pinch close* gestures, move along the view, with the *swipe* gesture, and tilting the view, by a *two finger swipe*.

The initial screen of our application is displayed in Fig. 2. In Fig. 3 its possible to see the main option menu made available to the user.

In Fig. 4 we display the representation of a set of graphic models defined in a local `kml` file. We also display, in Fig. 5, a load test where several entities represented by the same graphic model are rendered.

The developed virtual globe also allows the user to select each individual feature represented. This functionality can be used for several purposes. This allows different effects to be applied to the selected feature: to highlight it, for example, or to display the feature's attributes and related information.

The image and data sources for the application can be defined in a `.earth` file similar to the one presented in the listing 1. In order to provide greater control over the information presented in the application, we added the possibility of changing all the data sources in runtime. The user is capable of changing between several pre-defined data sources in runtime.

Figure 2: Initial screen of our application.

5.1 Expanding our solution to consume W3DS

Our greatest motivation to develop a virtual globe for mobile devices was the ability to extend it. Closed source API like Google Earth do not allow us to add new features.

With an open source implementation any new functionality, data type, or data source can be added. And if someone adds some sort of improvement, all other users will benefit from that.

In the following section we show how to add a new service provider to the virtual globe. We will use the newly specified Web 3D Service (W3DS) to show how W3DS 3D scenes can be rendered in the virtual globe.

5.1.1 W3DS

This proposed W3DS specification [10] defines 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 optimization of all scenes delivered, with the purpose of making the rendering of these scenes in real time possible with interactive frame rates.

Making our application capable of consuming the W3DS poses new challenges and we need to change the code to

Figure 3: Option menu of our application.

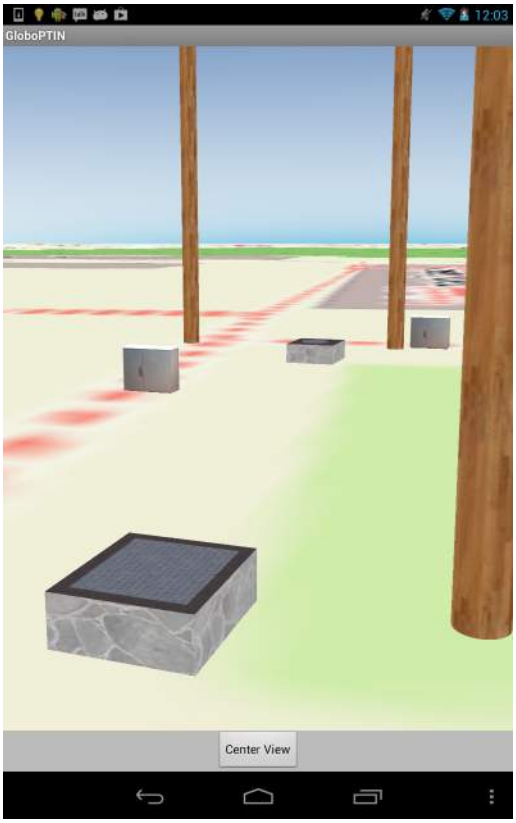

Figure 4: Representation of city infrastructure.

accomplish this new requirement. We used the W3DS implementation described in [7], a Geoserver module.

Transfer and render the large amount of information of a standard 3D scene is the major difficulty to overcome in an application that consumes the W3DS.

The complexity inherent to render these elements will also require careful management in relation to the graphic elements to keep in scene. During the application usage several 3D scenes will be transferred and represented as the users changes his view point. The performance needed to get and render all of these scenes is hardly compatible with interactive frame rates.

Consuming W3DS scenes implies constant changes to the set of graphical components represented in the scene. Since the rendering environment in this application is based on a graph based data structure, special care is needed when applying any changes to it. It is necessary to ensure that the changes arising from new scenes served by the W3DS never occur during the 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 that rendering and lead to unpredicted results.

5.1.2 Expanding the virtual globe

To address the presented issues auxiliary methods and data structures were used. For each layer request from a W3DS server, information is kept on which areas are already in memory. Thus, requests for areas already loaded in, are handled differently.

The level of detail in which each layer is active also deserved special attention, in order to to filter requests that would return an unbearable amount of data. This must be managed on the client side. A different solution for this problem would be to change the response logic of our W3DS server so that the level of detail would have an influence on 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 with a high level of detail, for a large area, only information corresponding to some, more important structures would be sent.

In order to perform the management of the number of features in the rendering graph we compute the geographical distance between the current viewing position and the centre of the bounding box that includes the elements of each particular group of assets. If this distance exceeds a certain threshold then the group is marked for removal, logging this operation in a changes queue.

In order to assist the decision performed in the changes queue, two flags were defined as attributes for each graphic node. These flags serve the purpose of indicating if that node is currently part of the rendering graph, and if its marked for removal or not. The state of these flags is altered when

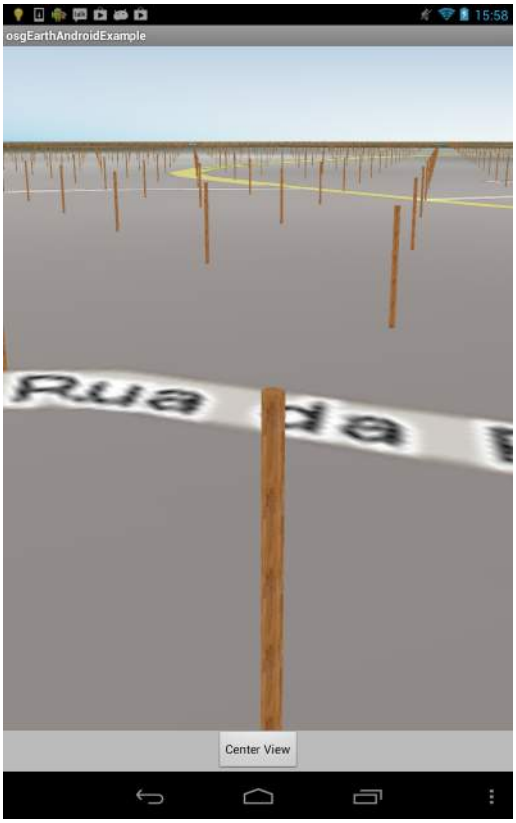

Figure 5: Load test from a local `kml` file.

a new scene is consumed from 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.

The method responsible for tracking the changes queue and making the necessary changes to the rendering graph is then registered in a callback system that calls the function between rendering cycles, ensuring the safety of each operation. Thus, with the implemented attributes and methods, using a callback mechanism we are able to ensure that these changes are never made in the middle of a cycle over the rendering graph.

Listing 1 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 1: Example of a W3DS data source being defined in a `.earth` file.

6. CONCLUSIONS AND FUTURE WORK

The overall objective of this project was the development of a powerful and expansible client for several geospatial data types and services, running on mobile platforms.

In the analysis of existing relevant solutions the concept of virtual globe was quickly identified as the most attractive solution. However, further research resulted in the conclusion that the number of virtual globe implementations for the mobile environment was very limited.

6.1 Conclusions

We choose to make use of an existing open source solution in the desktop environment as the basis of our project, making the necessary changes in order to port all necessary components. osgEarth was identified as the open source solution that best met the needs of our project. We chose to use Android, since it is widely adopted and offers an open source development kit.

We claim that we were able to port osgEarth to Android, and make it available as an open source framework for creating virtual globes in Android. We demonstrate the simple expandability of this framework by describing the process of making our application capable of consuming a W3DS service.

In addition to the most common features, our virtual globe offers real time control over the data sources to be displayed on top of the globe.

The resulting product is highly flexible and expandable and is available as open source, from the *git* based repository `https://bitbucket.org/jgrocha/osgearthandroid`.

6.2 Future Work

In the short term, the application can be improved in terms of performance and portability. To improve its performance, a detailed profiling and analysis is necessary to identify performance bottlenecks and possible optimizations. In terms of portability, several hardware configurations should be tested and evaluated, since Android is running in many device configurations, ranging from very basic devices to high end configurations with fast and versatile GPUs.

We already noticed interest in implementing an offline mode, where the application would consume local data from pre-prepared locations. This allows the application usage in

scenarios where network connections are limited or non-existent. In addition to the offline mode, the possibility of editing the data is also of great interest, further contributing to the usability of this solution as a major tool in geographical information systems. The editing capabilities pose several challenges, like synchronization back to the servers providing the data.

Data types and data source support can be extended by the community, to encompass common data type and services not yet covered. In particular the COLLADA `.dae` type has significant interest, as it is broadly used through several systems to represent graphic models and it has been adopted by ISO as a publicly available specification, ISO/PAS 17506.

7. REFERENCES

[1] Paulo Adelino Dias Almeida. An open source virtual globe for android. Master's thesis, Universidade do Minho, 2013.

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

[3] cartographic images.net. Behaim globe. `http://cartographic-images.net/Cartographic_Images/258_Behaim_Globe.html`, Accessed in November 18 2012.

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

[5] AI Gore. The digital earth. *Australian surveyor*, (February):41–44, 1998.

[6] 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*, 3:146–167, 2008.

[7] Nuno Oliveira and Jorge Gustavo Rocha. Web 3D Service Implementation. In Beniamino Murgante, Sanjay Misra, Maurizio Carlini, Carmelo M Torre, Hong-Quang Nguyen, David Taniar, Osvaldo Gervasi, and Bernady O Apduhan, editors, *Computational Science and Its Applications – ICCSA 2013*, volume 7973 of *Lecture Notes in Computer Science*, pages 538–549. Springer-Verlag Berlin, 2013.

[8] George Percivall. OGC Reference Model (OGC 08-062r4). Technical report, 2008.

[9] Mark Pesce. *VRML: Browsing and Building Cyberspace*. 1995.

[10] Arne Schilling and Thomas H. Kolbe. Draft for Candidate OpenGIS® Web 3D Service Interface Standard. 2010.

[11] Benjamin T. Tuttle, Sharolyn Anderson, and Russell Huff. Virtual Globes: An Overview of Their History, Uses, and Future Challenges. *Geography Compass*, 2(5):1478–1505, September 2008.

[12] The Geospatial Web. Media coverage of geospatial platforms. `http://www.geospatialweb.com/figure-4`, 2007 (Accessed in February 5 2013).