DEVELOPMENT OF TERRAIN SIMULATION APPLICATIONS WITH VRML-WEB3D AND GRAPHIC LIBRARIES

Emilio Jiménez[(1)], Eduardo Martínez[(2)], Mercedes Pérez[(2)], Félix Sanz[(2)]

University of La Rioja, ETSII, 26004 Logroño, La Rioja, Spain
[(1)] Department of Electrical Engineering
[(2)]Department of Mechanical Engineering

emilio.jimenez@unirioja.es *(Emilio Jiménez)*

Abstract

Terrain visualization applications are widely used in very diverse type of applications (geophysical systems, training simulators, etc.), with many different uses: development of roads, mines, dams, and general building works, establishment of water behavior in rivers or dams, visual impacts, real military strategy simulation, simulation of natural phenomena such as floods, volcanic eruptions, landslides and avalanches, etc.

These applications usually require a high computational power, and then must be executed in local computer systems; but they also require sometimes the capability of network utilization (for instance in distributed simulation, in public access terrain visualization, in distributed or multi-user simulation, etc.)

The recent developments of three-dimensional visualization systems can be naturally applied to terrain visualization targeted to the web, which traditionally had been implemented with 2D systems due to the huge data volume involved.

This work shows the steps and the methodology to follow in order to develop a virtual interactive terrain visualization system, taking into consideration the different uses that can be interesting to include in the practical applications for terrain visualization, and illustrating the explanations with some real applications.

The paper also includes the analysis of two of the possible technologies to achieve this objective: VRML as an exponent of Web3D technology, and open source three-dimensional graphic libraries. Their fundamental characteristics are considered, and their methodologies, pros and cons are illustrated, based on the different applications that are presented.

Keywords: Web3D based simulation, virtual reality (VR), terrain visualization, graphic libraries (GL), Virtual reality modeling language (VRML).

Presenting Author's biography

Dr. Emilio Jiménez Macías. PhD in Electrical Engineering (with *Computer science, electronics and automation* specialty), from the Universities of Zaragoza and La Rioja, Professor at the *Electrical Engineering Department* of the University of La Rioja, where he is coordinator of the *System Engineering and Automation Group*, and main researcher of the *Modelling, simulation and optimisation of industrial automated and logistics systems* Group.

ISBN 978-3-901608-32-2

1 Introduction

There exist several developments all over the world and very recent semantics in 3D visualization, so it is necessary to make a special effort in generating surveys and standards. In this paper, we wish to contribute to clarify the process of development of a Terrain Visualization System (TVS) in real time by providing a guide through the issues previously commented and illustrating the stages with practical examples.

We explain the pros and cons of some of the different currently available options, offering criteria for an appropriate development. In order to overcome the limitations given by Web3D technologies in general, and virtual reality modeling language (VRML) in particular, a specific graphic engine developed with open source graphic libraries is shown (Figure 1). Some programs - used to rename the terrain textures according to general VRML structures - and small applets, as interaction tools between the user and the 3D scene, have been implemented in a virtual TVS of La Rioja (one of the 17 autonomous regions in Spain, with a surface of about 5000 Km^2). They are used to clarify and exemplify some issues throughout the paper.

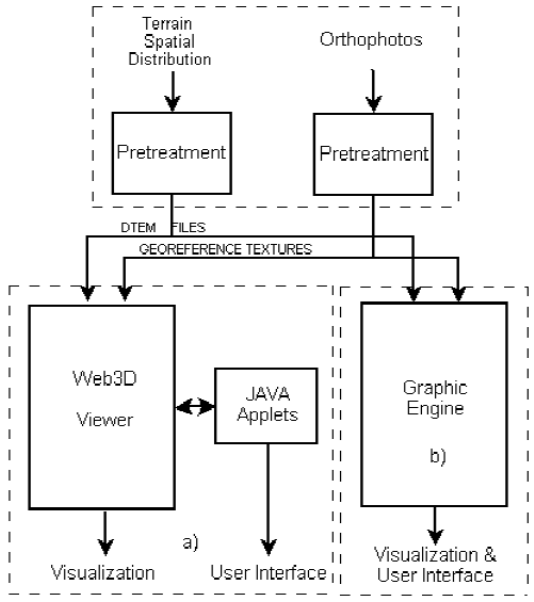


Figure 1. Virtual Terrain Visualization Systems: a) with Web3D viewers b) with graphic engine

The paper is organized as follows. First, the basic characteristics of a TVS will be briefly commented in Section 2. The Web3D-VRML technologies are introduced in Section 3 where their strong and weak points are shown. Section 4 is devoted to explore the VRML viewers and some tips to create the Digital Terrain Elevation Model (DTEM) and to endow the TVS with interactivity are provided. The development of the graphic engine, and its libraries, which present the 3D geometry of the scene, are discussed in Section 5. Finally, Section 6 concludes the paper and refers to future work.

2 Territory Visualization

The spatial distribution of the terrestrial surface is a continuous function, but on storing and representing these values digitally it is necessary to reduce the infinite number of points to a finite and manageable number, so that the surface can be represented by a series of discrete values [1] (surface discretization). For this purpose digital terrain models (DTM) and DTEM are used. DTM is a numeric data structure that represents spatial distribution in quantitative and continuous variables. These variables may be height, slope, contour, and orientation, as well as any other data applicable specifically to the terrain and its characteristics at any given point. DTEM is a numeric data structure that represents the height of the surface of the territory. By definition it can be seen that DTEM is a particular type of DTM. These DTEM are stored fundamentally in two digital formats: a) as a map of altitudes, that is, a two-dimensional matrix in which each quadrant represents the corresponding height of each point; b) by means of chromatic representation of the altitudes, that is, an image either in shades of gray or in color, where the color or shades of gray in the image depends on the specific height of each defined point. In general, for areas of lower height are assigned darker colors, and areas of higher altitudes have assigned lighter colors. The main problem with this second storing method resides in the color scaling assigned to the true terrain height.

Starting from this point and taking as reference any of the digital elevation models available nowadays on the market, we can begin to create our own 3D terrain model. For this purpose we must create a polygonal surface in which the vertexes agree with the coordinates taken from the appropriate DTEM.

The next step is to achieve that the 3D model that we have created present realistic appearance; that is to say, that it allows perceiving more details of any height relative of one given point with respect to any other. To achieve this objective we can think about the possibility of incorporating a specific model texture. The quickest method for achieving this realistic aspect is by using ortophotographs of the terrain. An ortophotograph is a digitally corrected photographic presentation that represents an orthogonal projection of an object, generated from real oblique photographs. Thus we can take measurements as if we had a map having the same values as on any map. Incorporating these corrected photographs of a terrain model, we can obtain a realistically acceptable representation (see Figure 2).

ISBN 978-3-901608-32-2

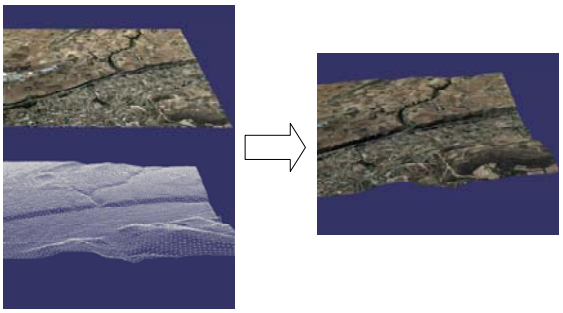


Figure 2: Creation of a 3D terrain model from DTEM and ortophotograph.

3 Web3D-VRML Technologies

One of the possible ways for implementing a territory visualization system consists on using Web3D technologies. The use of any of the Web3D technologies available nowadays, allows us to develop a 3D environment, capable to communicate on Internet and adaptable to the specific necessities that our system requires [2].

The term 'Web3D' [3] refers to any programming language, protocol, archive format or technology that may be used for creating or presenting interactive 3D universes through Internet. Among these languages for programming virtual universes, we can include as open standards: VRML (Virtual Reality Modeling Language), Java3D and X3D (Extensible 3D).

There are also a large number of solutions at proprietary level (an other ones still being developed) that satisfy the specific needs of the customers, generally aimed at electronic trade and entertainment purposes, such as Cult 3D, Pulse 3D and ViewPoint, etc.

In spite of these possible multiple solutions, to use an open standard presents important advantages; first the specifications and documentation are well known; and there are various applications of all kinds that support these standards. We shall briefly analyze the available standards, their main characteristics and their advantages and disadvantages.

3.1 VRML

VRML is an archive format that allows the creation of interactive 3D objects and worlds. The standard VRML was created and developed by the VRML Consortium, in principle a non-profit making organization exclusively aimed at the development and promotion of VRML as a standard 3D system on Internet. VRML appeared in 1994, the first officially recognized technology for the creation, distribution and representation of 3D elements on Internet by the ISO (International Standards Organization).

VRML is a hierarchic language of marks that uses Nodes, Events and Fields to model static or dynamic virtual realities. There are special Fields (EventIn and EventOut) that allow the sending and reception of events to other Fields.

With these special Fields and the command ROUTE, one can control the flow of Events, directing the effect of one action among other multiple objects to animate a scene or simply to pass information to any of these objects.

3.2 X3D

X3D is an open standard XML, a 3D archive format that permits the creation and transmission of 3D data between different applications, especially web applications.

Its principal characteristics are:

X3D is integrated in XML;
- X3D is modular;
- X3D is extensible;
- X3D is shaped;
- X3D is compatible with VRML.

X3D, instead of limiting itself to a single static wide specification - as in VRML that requires total adoption to achieve compatibility with X3D - has been designed with an architecture based on components that give support for the creation of different profiles, which can be individually used. These components can be independently extended or modified, adding new levels or new components with new characteristics. Using this architecture, these specification advances are faster and the development of one area does not delay the evolution of the global specification.

3.3 Java3D

Java3D™ API is a set of classes to create applications and applets with 3D elements [4]. It offers to developers the possibility of managing 3D complex geometries. The main advantage that this API 3D presents against other 3D programming environments is that it allows the creation of 3D graphic applications, independently of the type of system. It forms part of API JavaMedia. Therefore, it can use of the versatility of Java language, and it can support a great number of formats, including VRML, CAD, etc.

Java3D is a grouping of high class interfaces and libraries, which allows making good use of high graphic loading speed by hardware. The calls to Java3D methods are converted into Open GL or Direct 3D functions. Even though either conceptually or officially Java3D form part of API JMF – its libraries are installed independently of JMF. Java3D does not directly support each possible 3D necessity, but provides the capacity to implement it with Java code.

In other cases, VRML loaders are provided that translate files from this format to appropriate objects of Java3D. Browsers can visualize 3D environments by means of a plug-in.

Java3D provides a high-level programming interface based on the object-oriented paradigm. This fact

implies some advantages such as to obtain a more powerful, faster and simpler development of applications.

The programming of 3D applications is based on "scene graph models" which connect separated models with a tree-like structure, including geometric data, attributes and visualization information. These graphs give a global description of the scene, also known as 'virtual universe' [5]. This permits us to focus on geometric objects instead of the triangles existing in the scene.

3.4 Comparisons

X3D takes the work realized by VRML97 and it tackles matters that have not been specifically treated so far. From the VRML basis taken as premise, X3D provides more flexibility than VRML does. The main change is the total rewriting of the specifications in three different parts regarding: abstract concepts, file formats and ways to access to the programming language. Other modifications imply a greater precision in illumination and event models, and to rename some fields to obtain a solider standard.

The most important changes are:

- Graphic capacity expansion.
- A revised and unified applications programming model.
- Multiple file coding to describe the same abstract model, including XML.
- Modular architecture that permits ranges of adoption levels and support for the different kinds of market.
- Expansion of the specification structure.

The X3D scene graphics, the core of any X3D application, are identical to the VRML97 scene graphics. The original design of VRML graphic structure and its node types were based on already existing technology for interactive graphics. The changes to include the progresses in commercial hardware were carried out first in X3D graphics: the introduction of new nodes and types of fields for data.

X3D has a single unified programming application interface (API). This differs from VRML97, that has an internal and external scripting API. The X3D unified API simplifies and solves many of the problems that existed with VRML97 as the result of a more robust implementation.

X3D supports multiple codification archives, such as VRML97 and XML (Extensible Markup Language), or compressed binary, nowadays developing. It uses a modular architecture that provides greater extensibility and flexibility. The great majority of these applications do not need the full power of X3D, nr the support for all its platforms and defined functionalities in its specification. One of the advantages of X3D is that it is organized in components that can be used for the implementation of a defined platform or specific market.

X3D also includes the concept of profiles. They are a predefined collection of components generally used for certain applications and platforms, or in scenarios like the geometric interchange between design tools. Unlike VRML97, which requires total support from the implementation, X3D allows a support for each particular need. The mechanism of X3D components also permits the companies to implement their own extensions following a rigorous set of rules.

Furthermore, X3D specification has been restructured, allowing a greater flexibility in the life cycle of this standard, which adjusts itself to its own evolution. The standard X3D is divided into three different specifications that permit ISO to change the timing and the way to adopt the concrete parts of the specification.

One of the main differences between VRML/X3D and Java3D, at a conceptual level, is that Java3D is defined as a low-level 3D scenario programming language. This means that the creation of 3D objects and elements in Java3D does not only require the 3D element building, but also the definition of all the aspects related to the visualization and control of the scenario capabilities.

Another remarkable aspect is the loss of velocity and performance afforded by Java3D vs. other VRML/X3D viewers developed in C/C++ [6] and vs. viewers that directly use Direct 3D or OpengGL [7]. In spite of this, it is possible to use Java3D as a VRML/X3D file viewer. It is only necessary to use some of the VRML/X3D loaders developed for Java3D. At present, the Web3D Consortium is developing under GNU LGPL (Lesser General Public License), Xj3D as a tool to show VRML and X3D contents, completely written in Java. The main advantages in using Java3D as VRML/X3D visor is its execution capability in different platforms and the fact that the final user is released of installing specific plug-ins for the browser.

4 Use of VRML in the implementing of territory visualization

4.1 Selection of VRML viewer

The first problem to solve when we tried to implement the territory visualization system was to find a VRML viewer able to reasonably support and manage the great amount of data we wished to visualize. In the list of Web3D viewers available on the market nowadays, we can specially remark, among others, those represented in Table 1.

When carrying out a 3D visualization of an environment on Internet, it is necessary to bear in mind that the final representation on screen depends on the viewer chosen. Moreover, as it is to be

expected, neither all the viewers present the same behaviour, nor they are designed and programmed in the same way. So it is important to be very clear about which viewer is going to be used, in order to achieve the best possible results according to the needs of the application being developed.

Table 1: Web 3D Viewers

Web3D viewers	Java Applets	Proprietary Developments
CosmoPlayer	Xj3D	Axel
Cortona	WireFusion	ViewPoint Media Player
BS Contact	3DzzD	Adobe Atmosphere
Octaga	BS Contact J	Deep View
Flux	Shout 3D	Emma 3D
FreeWRL	blaxxun Contact3D	Cult 3D
OpenVRML		
Venues		
OpenWorlds		

Regarding the technological characteristics of these viewers, we can distinguish between those based on the use of a plug-in in the browser or those that use Java applets. Furthermore, we can find different proprietary solutions that use their own archive formats to store virtual scenes. Although these proprietary solutions can be better adjusted to the specific needs of a determined development at a given moment, they lack the advantage to work on an open standard recognised universally. So it is subjected to the decisions made by the proprietary company of that format and solution. However, the use of a system based on an open standard allows us to take our own virtual environment to the different developments that the standard.

For instance, Viewpoint Media Player uses a file format based on XML and includes the interaction capability through the use of scripting – continuous lines of interpreted commands –. Scripting vs. VRML presents a similar capability to interact directly with the environment in terms of execution time. When communicating with the Viewpoint Media Player plug-in from the HTML page, we can count on the possibility of using either JavaScript or Flash 5.

Adobe Atmosphere and Deep View are different applications mainly used by Adobe to give to its PDF documents the possibility to include 3D contents. Adobe stopped the development of Adobe Atmosphere in December 2004, and presently it uses the Deep View technology developed by HighHemisphere. In this case, Universal 3D (U3D) file format is used [8].

Emma 3D is an open-source development based on Ogre3D graphic motor and uses an archive format similar to VRML. Cult3D allows the visualization of models imported directly from 3D Studio and other formats, as well as basic animation and interaction with the scene. For example, if we use CosmoPlayer as viewer, we must take into account that it is old software. That implies it cannot make good use of the graphic capabilities of new 3D graphic cards, and it make mainly the rendering with software instead of with hardware.

On the other hand, if we use Xj3D, as well as any other viewer based on applets, we must remember that we use a viewer running in Java. Therefore, we must have the Java Virtual Machine (JVM) from Sun Microsystem installed and, according to the particular viewer, we may also need the Java3D library. In this case, to use the Java3D library allows us to accede to the graphic capabilities of nowadays-available 3D graphic cards. However, using JVM involves certain declines in the performance of the application, since it is an interpreted programming language [9], - or semi-interpreted language because a pre-compilation is carried out at the level of byte codes.

In Table2 [10], a comparison in the loss of performance and speed of Java3D against other VRML/X3D developed in C/C++ and directly using Direct 3D or OpengGL can be observed.

Table 2: Comparison of performances between Java and C++

Elements Used	Calculation	Comparison with C++
C++ (no 3D)	$0.9 \cdot 1 + 0.1 \cdot 1$	1.0 (0% slower)
Pure Java (no 3D)	$0.9 \cdot 1.35 + 0.1 \cdot 3.25$	1.54 (54% slower)
Mixed Java/C++ (no 3D)	$0.9 \cdot 1 + 0.1 \cdot 3.25$	1.225 (22.5% slower)
C++ (with 3D)	$0.4 \cdot 1 + 0.54 \cdot 1 + 0.06 \cdot 1$	1.0 (0% slower)
Pure Java (with 3D)	$0.4 \cdot 2.5 + 0.54 \cdot 1.35 + 0.006 \cdot 3.25$	1.924 (92.4% slower)
Mixed Java/C++ (with 3D)	$0.4 \cdot 1 + 0.54 \cdot 1.35 + 0.06 \cdot 3.25$	1.32 (32% slower)

Another important aspect for choosing a viewer is to know on which platforms it can run, and which is likely to be the possible range of users who will have access to the application. In principle, any viewer developed in Java has the advantages and disadvantages inherent in Java applications [11], that is, its capability for multiplatform execution and its dependence on the JVM of Sun Microsystem. Moreover, focusing on the developments of the VRML open standard, we can observe in Table 3 a summary of the different operative systems in which it is possible to execute each one of these viewers.

Table 3: Summary of the running capability in different platforms

VRML Viewer	Windows	Pocket PC	Linux	Mac OS
BS Contact	X	X	-	-
Cortona	X	X	-	X
Octaga	X	-	X	-
Flux	X	-	-	-
FreeWRL	-	-	X	X
CosmoPlayer	X	-	-	-
OpenVRML	X	-	X	X
Venues	X	-	-	-
OpenWorlds	X	-	-	-

In the decision-making process of choosing a viewer, the use of proprietary solutions was discarded in order to make good use of the advantages of an open standard such as VRML. As previously shown, VRML viewers can be sorted in two main groups

according to the technology employed: those which make use of JVM and those which incorporate plug-ins for the browser by means of ActiveX. The principal disadvantage the former group presents against the latter one is the loss of performance and velocity, unlike applications compiled at machine-code level [12]. This is the reason for discarding the use of any VRML viewer developed in Java; the specific needs of an application of territory visualization demands mainly high performance in refresh velocity of the visualization (Frames per second - FPS) and in memory use.

Finally, once the capability of the remaining viewers to execute the specific application developed has been tested, we decided to use the Bitmanagement Software viewer (BS Contact). At present, Bitmanagement Software and Octaga develop the leading viewers for the visualization of Web3D VRML/X3D technologies. The other viewers are a step behind as regards performance and updating to the development of new standards such as X3D.

4.2 Creating MDT

Once the different available Web3D technologies have been analysed and the one to be used has been chosen, as well as the necessary Web3D viewer, we have to determine the specific needs of the system we want to implement. The first step to develop this 3D territory visualization system is to have a terrain DTEM. In order to use this model, it is necessary to obtain the corresponding terrain height for each of the coordinates X and Y of the specific area that is to be visualized.

There is at present the possibility of knowing free the height of any point on Earth with a resolution of approximately 1 kilometre. This is possible thanks to files such as GTOPO30 (Global Topographic Data horizontal grid spacing of 30 arc seconds) of the U.S. Geological Survey's Center (USGS) for Earth Resources Observation and Science (EROS). Without any doubt this is a highly useful tool, but in our case they do not reach the desired precision, so it was necessary to resort to other greater resolution local databases. In this case as a starting point for the generation of DTEM, a database with a 5-metre spatial resolution was used. This database is in a dBASE format and occupies several Gigabytes. In order to work with it, it was necessary to create a program that allowed consulting automatically through coordinates X and Y represented in Universal Transverse Mercator projection (UTM). So it was possible to sequence terrain mesh generation process. For this purpose, a program using PERL (see Figure 2) was created which permitted covering the whole area, a total surface of more than 5.000 Km^2, and extracting the corresponding height coordinates.

Although the program worked correctly, the main problem found was slowness in the consulting process, as the different databases used were not correctly indexed. In order to solve this problem we had to resort to a program in C++ Builder that makes the automatization of the indexation of the different databases. Subsequently, another specific application was necessary to chose the area from where the data would be extracted and the required scan of the mesh (see Figure 3).

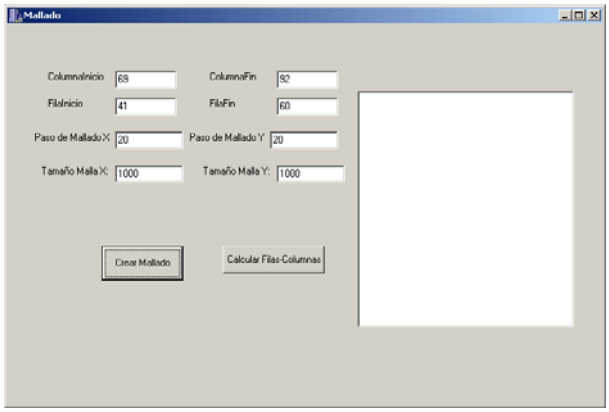


Figure 3: Application in C++ for the creation of DTEM

Once we are able to obtain the height (altitude) of each of the desired co-ordinates, the following step is to organize the resulting information into an appropriate form for its posterior treating and processing. In this case it was decided to build a structure in horizontal rows and vertical columns, which would be totally adaptable to the function of mesh, scan which could be chosen at any moment. Furthermore, it can be noted that in the above diagram, the visualized surface was divided into different Zones (areas) with the aim of being able to facilitate dynamic up-loading and down-loading in attention to the relative position of the observer/user (see Figure 4).

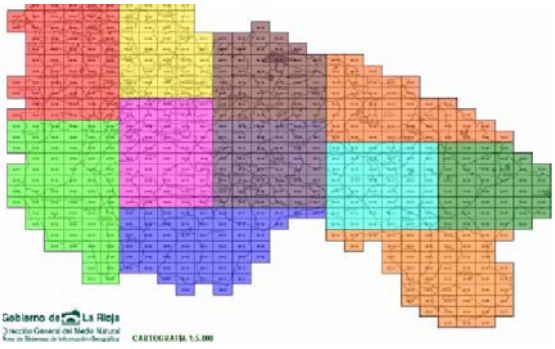


Figure 4: Division in Zones (Areas) of the surface to be visualized (116x75 Km^2). Each colour represents a zone (an area)

From this point different tests were carried out to adjust the size of the mesh scan to the specific needs of each application. In this case we are concerned with a territory visualization application in which the user is going to make a flight over the terrain at a determined height, such that it is not necessary to have to rely on an excessive mesh precision, as it is not going to afford anything visually important for the

observer [13]. So in the end a compromise decision was taken between resolution fidelity and required detail as against visualization refreshing velocity (the previously mentioned FPS), opting for the value of 100 metres mesh scan. This value is more than sufficient to maintain an acceptable realism in the environment topography as well as also allowing fluid navigation.

4.3 Incorporation of textures/ortophotographs

The following step to achieve a realistic visualization of the territory is to incorporate a series of textures into the terrain DTEM which we have generated, such that the textures from the aerial photographs of the terrain affords sufficient realism in the final visualization.

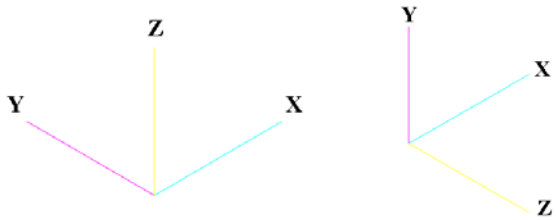
This is achieved starting from the corresponding terrain ortophotographs. In this case the format of the ortophotographs is JPG over an extension of 4x2.5 Kms, which constitutes 10 Km^2 surface area. On carrying out a simulation of the VRML environment, it is necessary that the model files and texture files are not excessively unwieldy. This presupposes the need to subject the texture/ortophotographs to a partition process according to the dimensions and scene structure previously explained. To carry out this partition a specific C++ program was employed (see Figure 7).

As well as this it is necessary to create the maximum precision in the optimum resolution of textures for its posterior resolution during simulation. In order to find this maximum resolution tests were carried out from 0,5 m/pixels up to 4.0 m/pixels. In the end it was found that using resolutions lower than 2 m/pixels did not really afford much improvement in scene realism, due to the treatment the different viewers offered in these textures. As well as this, using such high-resolution textures in the different viewers obliged a notable increase in the size of these viewers and in the loading of work made by the viewers, together with the corresponding loss of visualization refreshing (the FPS). Therefore it has been determined that the texture resolution in our application will be 2 m/pixels.

Another important aspect in the speeding up of simulation was the inclusion of detail levels [14], such that it was possible to lighten the viewer load without losing realism or quality for the observer. In order to achieve this detail levels through the use of different texture levels was established, depending on the distance of the observer from them.

- From 0 to 1,500 metres: 2 m/pixels resolution.
- From 1,500 to 5,000 metres: 4 m/pixels resolution.
- From 5,000 metres to eye-sight reach: without texture and only the net-meshing is observed.

Another noteworthy treatment of the ortophotographs are the reference co-ordinates. The ortophotographs are represented in UTM (see Figure 5a) and which have to be adjusted to the co-ordinate system used in VRML simulation (see Figure 5b), for which it is necessary to make a vertical inversion of the ortophotographs through an informatic application (for instance, it is sufficient to use any Photo-Editor program). In this way ortophotograph UTM co-ordinate systems and VRML environment systems become pefectly harmonized.



(a) (b)

Figure 5: Ortophotograph Reference System

4.4 User Interaction Tools

The method which allows VRML to increase (improve) user interaction is through JavaScript Code (usually denominated VRML Script) in Java. The writing of VRML Script codes presupposes the incorporation of different interaction methods within VRML virtual scene, whilst through the use of Java codes it is possible to interract with the scene from a series of external applets. Thus, the programmer is completely free to create his own user interface through Java libraries [15] and then through External Authoring Interface (EAI), and be able to connect with virtual scene. For the use of VRML Script or Java it is necessary to resort to VRML package libraries: whilst VRMRL Script uses the VRML, VRML.node and VRML.field, on using Java applets we have to fall back on VRML.external.

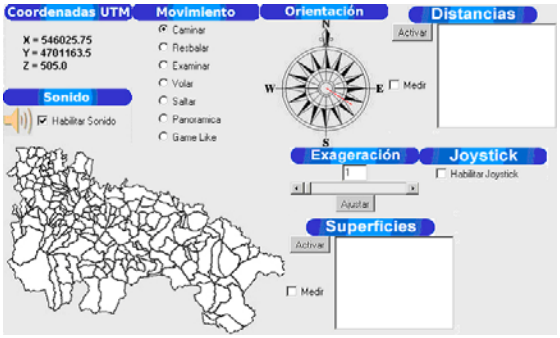


Figure 6: External Applets User Interraction

Concentrating on the use of applets to connect with VRML scenes through EAI library, we must count on the following elements:

- HTML Archives: in the same HTML page we must include references to VRML files and applet.
- Applets: Applets are presented in the usual generic coding of any applet, but it must also include the necessary coding to communicate with VRML Scene. They allow to accede to VRML secenes and control them.

- Node references: in order to read certain information in the VRML scene or to control certain parametres, it is necessary to make reference to a specific scene-node which contains this information or parametres.
- Teading and writing VRML scene: Once a specific scene-node is referenced its fields can be acceeded through these functions. Their use is limited to access to those fields defined as eventOut and exposedField.
- Receiving VRML scene events: In the case of needing to receive events produced be the scene we must implement in our applet EventOutObserver: "public class compass extends Applet implements EventOutObserver" interface.

4.5 Pros and cons in using VRML for territory visualization.

One of the advantages of implementing the VRML visualization system is having to recur to the back-up of an open standard, with all that this implies, from available documentation to the possibility of using different design tools which support this standard.

However this presents the great disadvantage or limitation of end dependence on an external viewer on which there is no real control. Although using of EAI library allows certain control and interaction with the VRML viewer and VRML scene, in effect we do not have access to low-level viewer methods and configuration, as is the case with algorhythm renderings.

5 Development of a graphic motor for territory visualization

5.1 Open-coded 3D visualization

In order to resolve the problems expressed in the previous paragraph, on generating an appropriate system for territory visualization the first step is to know what graphic libraries are at present available which meet our needs. Torque Game Engine (TGE), TV3D SDK , 3D GameStudio or Reality Engine are different systems for 3D viewing and they all belong to different companies under different kinds of licencing. Although any of these developments could meet our specific needs, today various open-coded projects exist which offer comparable technical performances to the above-mentioned. Within these open-coded projects can be specially mentioned: Crystal Space, OGRE, Irrlicht, Nebula Device 2, RealmForge GDK, OpenSceneGraph, Axiom.

In Table 1 we can see the main characteristics of the various open-code developments. As can be seen, some of these projects are still under initial developmente and thus do not yet present a sufficient degree of reliability in some specific aspects of their functions.

Table 4: Synopsis of different open-coded graphic library characteristics

Graphic Motor	Program Language	API Graphics	Operative Systems	State of Development
OpenScene Graph	C++	OpenGL	Windows, Linux, MacOS, Solaris, SunOS, FreeBSD, Irix, Playstation	Stable
OGRE	C++	OpenGL/DirectX	Windows, Linux, MacOS	Stable
Irrlicht	C++	OpenGL/DirectX	Windows, Linux, MacOS	Alpha
The Nebula Device 2	C++	DirectX	Windows	Stable
RealmForge GDK	C#	OpenGL/DirectX	Windows, Linux, MacOS, Solaris, HP/UX, FreeBSD	Alpha
Crystal Space	C++	OpenGL	Windows, Linux, MacOS	Stable
Axiom	C#	OpenGL/DirectX	Windows	Alpha

Finally, after evaluating the possibilities of the different libraries presented, it was decided to use OpenSceneGraph, basically due to its independence of the platform used, and above all for its appropriate construction and its expansion possibilities. The main disadvantage was its lack of specific documentation, but this problem is minimized through a series of practical examples that afforded basic knowledge of the different capabilities and functioning of the library.

5.2 OpenSceneGraph

OpenSceneGraph [16] is a recently-developed graphic library which incorporates the different primitive basic concepts of OpenGL. This language uses C++ as a programming language as well as presenting independence of tha platform, besides being an open-coded development. Among the possible uses of this library we find scientific visualization, virtual engineering and game development.

OpenSceneGraph employs scene graph techniques to contain all the information relevant to the generated scene. A scene graph is a data-structure which allows the creating of a scene hierarchic structure, such that a father-son series is maintained among the different elements. For example, father-node position and variation positions affect son-nodes. In this way a various link robot-arm can be created, each one dependent on the previous, and simply applying an initial link-movement, the rest of the dependent links will automatically move according to the defined structure. Another important father-son relation exploited by the scene-graph techniques is the possibility of defining envolving volumes which group close elements, so that during element download processes which are to be represented on-screen, it is not necessary to fall back on sons of a father-node already discarded.

5.3 3D model creation

In order to achieve a correct and agile territory visualization, it is first necessary to carry out a correct modelation of the scene to be represented.

ISBN 978-3-901608-32-2

As is to be expected, due to the great quantity of information to be treated, it is necessary to establish a correct and ordered structuration to facilitate and flexibilize its management or handling. At the software level the solution is to maintain at all times a similar quantity of information (textures and DTM), which allows a fluid handling of information. To achieve this we resort to a dynamic up-load and down-load of the scene being treated according to the position of the user at each moment. This process of scene up-load and down-load is carried out through a data base which pages the different scene areas and allows us to decide which part or parts are necessary to bear in mind at each moment (see Figure 12). At the theoretical level the date base limits are not defined, but in practice the larger the size, the more the process performance of dynamic up-loading and down-loading is affected. Due to this it is very probable a data base re-estructurion may have to be carried out, according to communities and provinces, always searching for a way to limit the data-base size to the specific necessities at each moment according to the surface over which the user moves.

Apart from this division in zones (areas), a division in horizontal rows is also carried out. In this way the facilitation and optimization of the selection process of the different scene graphs is carried out, whether they should be rendered or not.

Besides relying on a correct scene structuring it is also necessary to have available a correct system of multiresolution textures to reduce the load during rendering [27]. In order to achieve this a PagedLOD node is available which allows the settings of several texture resolution levels, such that according to the distance between the observer view-point and the model can be visualized at each level.

The rendering task in OpenSceneGraph is divided into three stages. The first is Update, in which changes in executing time of the scene-graph are made; the second is the Cull discard in which the list of scene elements which will be rendered in the last stage is formulated; and lastly is the process of rendering in itself (Draw).

In order to use this tool, three resolution levels have been established (2 m/pixel, 4 m/pixel or mesh) according to the distance (under 1,000 metres, from 1,000 to 5,000 metres, or over 5,000 metres) as previously mentioned in detail levels.

Through this structure, with the appropriately chosen ranges of rendering for each of the levels, constant refreshing velocities from 20 to 30 FPS is achieved, even during the loading of the application (Figure 7).

On the other hand it is important, though not strictly necessary, to generate the entire geometry of the scene in the binary format provided by OpenSceneGraph's own graphic library. This binary format (IVE) facilitates the initial process of scene loading by the system, thus lessening wait-time for the user on loading the application.

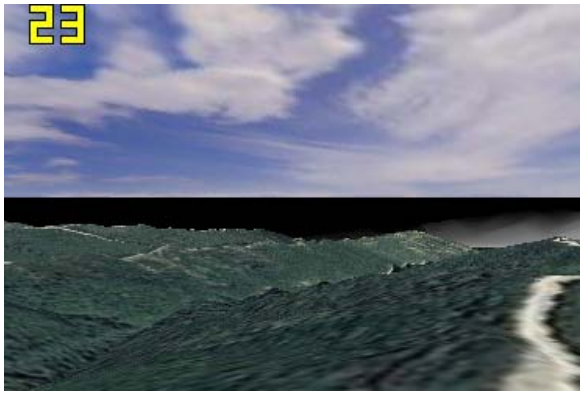


Figure 7: Application Capture during FPS at that moment

5.4 Incorporating independent tools to the visualization

One of the most important aspects in a territory visualization application is the ability of the user to navigate with ease and without environment problems. For this purpose usually user interaction by means of the mouse moving over the scene visualization on screen is employed. Thus it will be necessary to provide the application with the capability of receiving and responding to the events with the mouse (which are) being produced in such visualization. This can be achieved creating a new class from the program "Producer::KeyboardMouseCallback" and putting into use the appropriate behaviours. For example, in the case of territory visualization, the user must always be above ground level, and, generally, at a determined height (altitude). It is therefore necessary to employ a collision detection system, which prevents the user from penetrating through the terrain itself as he moves over the scene. For this purpose we can make use of a specifically designed visitor in OpenSceneGraph's "IntersectVisitor". Apart from facilitating 3D visualization of the scene, it is also necessary to create a user-friendly interface (see Figure 8), with which the user can interact simply and which affords him the capability to control or obtain information from the scene (for example, recuperating observer position at each moment to present it on screen).

6 Conclusions and future works

Throughout this article we have set out the basic steps to be followed for the development of territory visualization application. In first place we have presented the basic characteristics of a territory visualization system. Following that we have included a brief study of the different applicable technologies basing ourselves on VRML, X3D and Java3D. From this base we have proceeded to detailing the different necessary steps to put into practice a territory visualization system based on VRML, from the choice of viewer to the incorporation of Java-developed user-

ISBN 978-3-901608-32-2

interraction external tools. Also aspects such as DTEM creation and the incorporation of model photorealist have been explained.

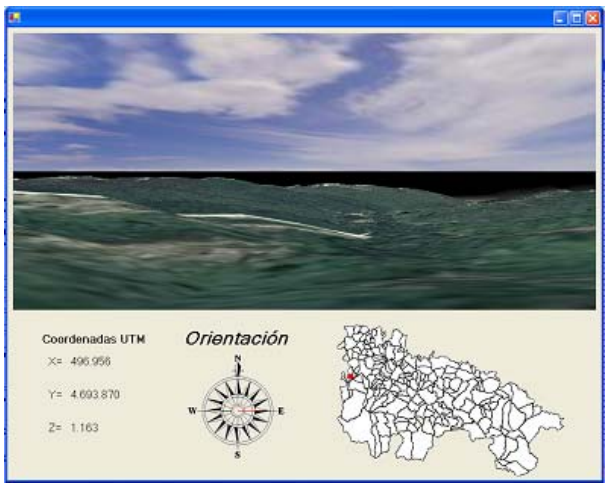


Figure 8: User Interface

From the aforementioned it can be deduced that the use of VRML for the creation of territory visualization is viable, but always at the behest of depending on an external element entrusted with scene visualization, over which one does not have real control, nor is its codification known, nor can its programming be modified. In order to overcome these short-comings the development of a specific graphic motor through the use of open-coded has been proposed. With this objective in mind, different existing open-coded graphic libraries and their basic functioning characteristics have been analyzed, to finally detail the steps followed in the implementation of territory visualization by means of OSG library.

One of the potential capabilities of the developed system which may be exploited in the future, is the different 3D geometries, with the end purpose being to facilitate aspects such as territory management, distribution and planning, as a further onward step in geographical information systems.

7 References

[1] Giger, C.: "Digital Elevation Models and Digital Terrain Models", Technical Presentation, GeoInformation Technologies Group, Swiss Federal Institute of Technology, Zurich, Switzerland. 2002.

[2] Lindstrom, P., and Pascucci, V. (2001). "Visualization of Large Terrains Made Easy", Proceedings of IEEE Visualization 2001, IEEE, Piscataway, NJ, 363-371.

[3] Web 3D Consortium - Open Standards for Real-Time 3D Communication. http://www.web3d.org/

[4] Sowizral H, Rushforth K, Deering M. The Java 3D API Specification. Addison-Wesley, 1998.

[5] Burrows, A. L.; England, D.: Java 3D, 3D graphical environments and behaviour, Software-Practice and Experience, 359-376, 2002.

[6] Burns, A., Wellings, A.J. Real-time Systems and Programming Languages. Addison Wesley, 2001

[7] Mason Woo and others: OpenGL programming guide, Addison-Wesley, 1999.

[8] Universal 3D Format. http://www.intel.com/technology/systems/u3d/

[9] Barr, R.; Haas, Z. J.; van Renesse, R.: JiST: an efficient approach to simulation using virtual machines, Software-Practice and Experience, 540-576, 2005

[10] Evaluating Java for Game Development. March 4th 2002. Jacob Marner

[11] Kilgore RA, Healy KJ, Kleindorfer GB. The future of Java-based simulation. Winter Simulation Conference, December 1998; 1707–1712.

[12] Wellings, A.J. Concurrent and Real-time Programming in Java. Wiley, 2004

[13] AYENI, O.O., 1982, Optimum Sampling for Digital Terrain Models: A Trend Towards Automation. Photogrammetric Engineering & Remote Sensing, 48(11), 1687-1694.

[14] Blow, J.: Terrain Rendering at High Levels of Detail. Proceedings of Games Developers Conferences, 2000.

[15] Gosling J, Joy B, Steele G. The Java Language Specification. Addison-Wesley, 1996.

[16] R. Osfield, D. Burns, OpenSceneGraph. http://www.openscenegraph.org. 2003.

[17] Guedes, L.C., Gattass, M., and Carvalho, P.C.P. (1997). "Real-Time Rendering of Photo-Textured Terrain Height Fields", Proceedings of SIBGRAPI 97, Campos de Jordao, SP, Brazil, 18-25.

