

# Interactive demo: OpenGLfly, version 'Euskadi'

José Daniel Gómez de Segura    Borja Fernández    Rosa Peral    Susana López    Eduardo Ibáñez  
 jdgsegura@euve.org    bfernandez@euve.org    rperal@euve.org    slopez@euve.org    eibanez@euve.org

Virtual Reality Department  
 European Virtual Engineering (EUVE)  
 Vitoria - Spain

---

## Abstract

*This paper describes the demo associated to one of our projects, which is currently under development. The version presented can be considered as a large terrain simulation over the Basque Country area (North of Spain), with real imagery, georeferenced information and a user friendly and interactive interface.*

*The main goal is to demonstrate the applications of virtual reality techniques to fields such as virtual tourism, low cost simulators or edutainment. The requirements of the demo are low enough for it to be run in a home PC with a middle range graphics accelerator card. Currently, there are some OpenGL optimisations specific to NVIDIA chipsets and consequently this platform has a slight advantage in terms of performance.*

---

## 1. Introduction

The motivation for this project started two years ago. At that time we were involved in the development of a real time simulation of a civil engineering work over a large terrain areas for one of our customers.

The initial proposal included a simulation that used SGI OpenGL Performer libraries. Obviously, the hardware platform was from SGI. We own an Onyx2 with three graphics pipes and a Reality Centre for active stereoscopic visualisation.

The use of Performer has some key advantages, as it drives the Onyx2 graphics hardware to its maximum. Another reason is that it implements techniques for terrain visualisation<sup>1</sup>, such as:

- ASD: Active Surface Definition is a powerful real-time surface meshing and morphing library. It enables you to roam surfaces that are too large to hold in system memory very quickly. The surfaces, called meshes, are represented by triangles from more than one LOD. (SGI's Performer Guide)
- Cliptextures: patented algorithm to allow textures that are much bigger than will fit in texture memory, and even in system memory and exploits and extends properties of MIP-maps (<http://www.vterrain.org>)

Although the best quality in the simulation could have been achieved with these features, some major drawbacks were identified.

The main one was that the purpose of our customer with this simulation was the presentation of their project to different social agents, which implies mobility. Unfortunately, SGI's high-end hardware is not very common. This prevented us from delivering the simulation as code to be installed in other locations. Consequently, it was clear that the software and hardware together formed part of the final product.

From our experience, customers usually want to show this kind of simulations at their own sites. Our facilities could have been used, but this was unpractical in this case and moving the Onyx2 is fairly complicated. This does not mean that it cannot be done at all, but only for special events.

Thus, we reached the conclusion that it was necessary to provide a PC version of the simulation. PCs are way cheaper than Onyxes, so the customer could acquire a suitable platform for every location where the simulation was to be shown, or even use some of the available computers if these met the requirements.

In order to get the greatest compatibility between both platforms and reduce development time, some

potentialities were not implemented (ASD and cliptexturing, while true multipipe capabilities run only on Onyx). The PC code was based on OpenGL libraries and basically runs with the same database as in the Onyx.

With the knowledge gained from that project, we decided to develop a new application for large terrain visualisation, removing the sensible materials from our customer that could not be shown, and with these requirements:

- use of standard DEM and orthophotography available from the Internet or from public institutions
- fast enough on a standard PC so that most users can run the software at home
- optimised for vast terrain areas with high visual fidelity
- easy to fly and visually attractive

The current state of this development is the demo that is being presented in this paper.

## 2. Description of the demo

The application openGLfly, version 'Euskadi', is a 3D graphic engine for high quality and visual fidelity terrain visualisation, in this case for the Basque Country region. The terrain area is not limited by the graphic capabilities of a PC, but for the amount of secondary storage space. It can also show georeferenced information about singular points, such as cities, villages and mounts.

The user can move, or better fly, over the terrain database using the mouse, which is represented by a paper airplane, and the mouse buttons:

- left button: accelerate
- right button: decelerate
- middle button: stop and look around

The database has been created from digital elevation models (DEM) that can be obtained from the Basque Government. They have been processed with specific algorithms, which will be discussed later, in order to get and adaptive triangle mesh. This reduces the number of elements without introducing too much error with the original DEM. The terrain area has been divided into 3000 x 3000 metre tiles.

Each tile, of square shape, has an associated texture created with orthoimagery from the same source as the DEMs. The quality of these photographs has been reduced to meet the requirements of memory of a PC graphics card.

The application defines a grid of 11 x 11 tiles and textures around the observer position in the virtual database. When the observer moves, tiles and textures outside that grid are unloaded from memory and replaced with new ones, so that there is always geometry in front of the camera.


There is a touristic mode, which can be activated with the button  on the lower right corner of the screen. In this mode, a window opens on the right hand side with information about the nearest visible singular point in the database.



Fig. 1 The application showing additional information for the nearest singular point.

Clicking with the left button of the mouse on the map moves the camera position to that coordinates. This action implies the unloading of all tiles and textures and the loading of 121 (11 x 11) new ones. Therefore, the process is hidden in order to avoid terrain popping.

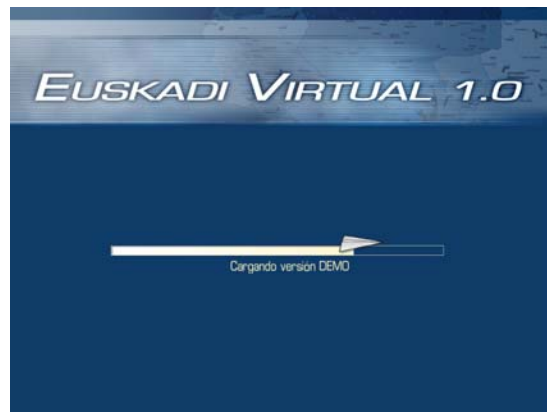



Fig. 2 OpenGLfly while loading terrain data

The help button  or the F1 key, show a window on the left hand side with the actions that can be performed pressing keys or with the mouse.

The ESC key quits the application.



Fig. 3 Quit confirmation dialog

OpenGLfly uses multithreading, so that certain critical parts of code are not blocked by I/O operations. A multiscreen version has also been developed (however, it is not presented here). Such version can run distributed over several computers synchronised over TCP/IP links and can be used for larger displays or passive stereoscopic visualisation.

This features have been implemented in this demo:

- OpenGL coding
- Culling with Bounding Boxes
- Anisotropic texture filtering
- Circular fog
- Textures in R5G6B5 format to reduce memory utilisation
- Vertex Array Range extension for OpenGL for a faster vertex transfer to the graphics card memory.
- Localised for Spanish, English and Basque Language.

### 3. Design considerations

#### 3.1 TIN

For simulations where the user 'flies' over the terrain, which is composed of DEM and orthophotographs, a TIN is usually a good compromise between accuracy and geometric complexity (which can be great enough to slow down the drawing of the scene)<sup>2,3</sup>.

A TIN, or triangulated irregular network, is composed of several triangles in different shape. They form a mesh that minimises the differences with the original DEM up to a desired error measure. It is pre-computed and it does not implement LODs<sup>4</sup>.

One reason to use TINs is that in this scenario the textures lose quality when the camera is too close to the terrain. The observer does not get any visual

improvement in that situation. Besides, the information contained in aerial photographs is 2D. Therefore, they are only useful for views from above. If a surface view is needed, manual modelling can hardly be avoided in order to recreate the features in the terrain, such as buildings, roads and even terrain materials.

Consequently, the approach used in the demo is adequate for simulations where the camera remains over a certain altitude.

An advantage of this method is that is very easy to position or reference objects on the terrain surface, as this does not change. If several LODs<sup>4</sup> had been used, the differences in the rendered terrain make necessary to track the relative position between objects and the actual surface.

So as to make terrain tiling easier (a must to hold huge terrains), the terrain has been splitted in more manageable pieces, with accordingly photograph divisions and geometry divisions. Finally triangles have been generated using Delaunay and a variant of data dependent triangulation algorithm. We introduced a little difference. Although the general algorithm remains the same, the error measure between the real data and the approximated mesh was calculated with volume differences between the planes we were generating and the ones given by the "ideal mesh", this is, the mesh we would have if we used all the points.

In our implementation, terrain vertices have 3 components (position, the normal vector and texture coordinates).

Position is quite trivial to obtain from the usually initial reference UTM coordinates and the metre resolution of the files.

So are the texture coordinates if the projection of the points is linear and tiled pieces are therefore rectangles. If generated tiles aren't rectangles, barycentric coordinates should be used to achieve accurate texture mapping, which should be done off-line to speed up the whole process, as the real-time coordinate generation is slower.

#### 3.2 Efficient paging of terrains

In the terrain generation process we have defeated the RAM memory size limitation, and it's even more important to do so in real-time applications.

With this idea in mind we've implemented an algorithm that loads/unloads terrain in system memory as they are needed by proximity to the camera. The terrain pieces to be loaded in system memory can be

selected in several ways: a rectangular matrix around the camera, the nearest ones inside the frustum and others<sup>5</sup>.

We've chosen to load a rectangular matrix of terrain around the view point, because used memory size isn't as important as avoiding temporally missing terrain when turning the camera.

The paging algorithm flow diagram is described in Appendix A.

Implementing the terrain pagination in a separate thread is a must in order to avoid the drawing thread being blocked by I/O operations, which is just unacceptable in any commercial graphic application<sup>6</sup>.

#### 4. Results

We measured performance (frames per second) with different graphic devices and the results obtained are the following ones:

Graphic device	Frames per second
nVidia GeForce 2	~50
nVidia GeForce 3	~65
nVidia GeForce 4	85
ATI Radeon 7200	~10
ATI Radeon 9700	~40 (normal PC) ~65 (very fast PC)

As we can see, even the fastest ATI card hardly equals the speed achieved with a non top-edge nVidia card, even though it's supposed to be even faster than a GeForce4, but this is because ATI doesn't support the nVidia's OpenGL extension. They have made another similar extension and we have not supported it in the engine because it's a bit different and programming would get a bit complex<sup>7,8,9,10</sup>.

With a GeForce4 (not nVidia's latest card, which is GeForce FX, not available here at the time we were writing this paper), we achieved a frame rate equal to the used refresh frequency.

Regarding the visual quality, it has to be noticed that some terrain artifacts can be found. These appear as pyramids, or as points of unusual height. It has been probed that they also exist in the original data, which is the case for the artificial cliffs that are located in the political border of the region and the unreal hill near Hondarribia.

#### 5. Requirements

This demo requires a PC with a 32 Mbytes graphics accelerator card compatible with OpenGL. It has been tested under Windows 2000 and XP, but it should also run with Windows 98 & Me.

At least DirectX 8 has to be installed, as it is used for input devices interaction.

The simulation has an acceptable frame rate with processors better or equal to a Pentium III @ 350 MHz, with 128 Mbytes RAM.

To copy and decompress all the terrain database, 900 Mbytes are needed in the hard disk.

#### 6. Future work and improvements

Although this demo runs quite smoothly and can be considered a standalone application for virtual tourism, it is not the final stage of our project. There are many improvements that can lead to a better efficiency and visual quality.

We plan to research and develop in the following areas:

- better paging algorithm, in terms of memory usage and speed: the rectangular tile grid is easy to manage, but leaves most of the data behind the observer point of view. In our case, this means a waste of texture memory.
- higher resolution textures, or mipmaps: in order to improve the appearance, instead of using more polygons, it is best to use higher resolution textures.
- cinematic effects.
- modelisation of Nature.

#### 7. References

1. SGI OpenGL Performer Programmer's Guide.
2. Michael Garland and Paul S. Heckbert, "Fast Polygonal Approximation of Terrains and Height Fields", Technical Report CMU-CS-95-181 Computer Science Dept., Carnegie Mellon University.
3. Paul Bourke, "An Algorithm for Interpolating Irregularly-Spaced Data with Applications in Terrain Modelling", Swinburne Centre for Astrophysics and Supercomputing, Australia.
4. Hugues Hoppe, "Smooth View-Dependent Level-of-Detail Control and its Application to Terrain Rendering", Microsoft Research.

5. R. Pajarola & P. Widmayer, “Virtual Geoexploration: concepts and design choices”, International Journal of Computational Geometry & Applications.
6. Barry Wilkinson, Michael Allen, “Parallel Programming”, Chap. 11, Image Processing, Ed. Prentice Hall.
7. NVIDIA OpenGL Extension Specifications, [http://developer.nvidia.com/view.asp?IO=nvidia\\_opengl\\_specs](http://developer.nvidia.com/view.asp?IO=nvidia_opengl_specs), NVIDIA Corporation.
8. AGP technology technical information, <http://www.intel.com/technology/agp/info.htm>, Intel Corporation.
9. Optimizing memory bandwidth, [http://cdrom.amd.com/devconn/events/gdc\\_2002\\_amd.pdf](http://cdrom.amd.com/devconn/events/gdc_2002_amd.pdf), AMD Corporation.
10. IA-32 system programming, <http://developer.intel.com/design/Pentium4/manuals/>, Intel Corporation.

Appendix A

