

# Landscape Visualization in High Resolution Stereoscopic Visualization Environments

Björn ZEHNER

## 1 Introduction

Large-scale projection-based visualization systems provide users with a unique feeling of immersion and the possibility of providing them with an impression of the real-world size of objects. This can be used to help the general public to participate in landscape planning issues. In one project, which was our initial motivation for looking into the matter of showing digital landscapes on such a system, the project partners want to investigate users' preferences regarding the types, size and distribution of wind turbines in onshore windfarms. However, trying to obtain users' opinions by presenting them simply with different choices will most probably be too abstract. Imagine, for example, people who are interviewed about their opinion of windfarms and asked to decide between larger or smaller wind turbines in relation to the minimum distance they should be allowed to have from the next village. In most cases the person interviewed will have no notion of what it means to see a 120m high wind turbine from a distance of one kilometer. Furthermore, by travelling interactively through a 3D representation of the potential future windfarm and its surrounding landscape, viewers would also become aware of effects such as the visual occlusion of the wind turbines by trees which might influence their view on the question of whether or not turbines should be placed in or near forests. Within a dense forest the turbines, even if they have to be taller than on an open field, might only be visible to someone strolling through the forest from a much shorter distance, due to this effect of occlusion.

To aid the viewer in imagining and understanding the scenery, we use a projection-based stereoscopic visualization environment with an approximately 6×3 meter large screen and corresponding projections on the floor and sidewalls. In order to achieve a high resolution of approximately 6400×1800 pixels, 13 SXGA+ projectors are used to run this system. Figure 1 shows a picture taken in our display system. The image is generated frame sequentially for the left and the right eye and users wear special glasses which separate these images, so that they get a real 3D stereoscopic view. Viewers are headtracked, so that they can turn their head and move while the image is always computed in a way that correctly maintains perspective, and they can move through the virtual landscape using a kind of pointer. The rendering is done using 13 normal workstations with high-end graphics boards. The use of such large-scale Virtual Reality systems is common in the oil and gas or automotive industries but rare in the environmental sector. The Macaulay Institute in Aberdeen uses a cylindrical 3-channel projection for public participation (MACAULAY 2008). The Collaborative for Advanced Landscape Planning at the University of British Columbia, Vancouver, uses its Landscape Immersion Lab, a 3-channel projection, for visualizing and investigating landscapes (CALP 2008).



**Fig. 1:** The visualization center

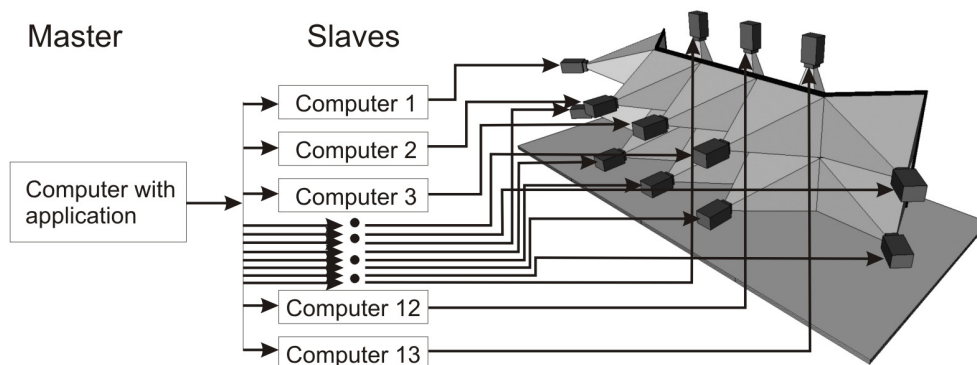
The need to generate the image distributed on 13 projectors creates difficulties and also places constraints on the software or graphics libraries that can be used and on the way the scene has to be organized. It should also have a high quality, both in the foreground and background. MEINEL & WALTER (2001), for example, describe the construction of landscape visualizations for the evaluation of potential windfarms by local authorities. They point out that the detail in the foreground is very low when standard tools like GIS are used, and that this prevents the use of 3D visualization for critical decision-making, such as when an important view from a monument has to be evaluated.

This paper first outlines the various technical methods that can be used to run such a display system for landscape visualization and then examines which one has been chosen. Then it describes the workflow we have established to generate a high quality digital landscape that can be shown on such a system and our experiences of the software and libraries used.

## 2 Multichannel Rendering

The hardware setup in our laboratory is sketched in figure 2 as an example of this technology. To run a multichannel system, specialized hardware and software is needed that can provide the necessary 13 stereographic synchronized outputs. A few years ago this was the domain of SGI (Silicon Graphics Inc.) computers with multiple processors, shared memory and multiple graphics units, such as the SGI Onyx or the SGI Prism. These came together with suitable software development kits, such as Iris Performer (ROHLF & HEHLMAN 1994), which made the development of the necessary software easier. These computers were very expensive and can currently not be bought in the EU because they are

not RoHS (Restriction of Hazardous Substances) compliant. SGI also had problems keeping up in terms of graphics performance with the mass-market-driven development of consumer graphics boards. Today, the state of the art to run such systems is the use of a cluster of normal workstations equipped with very good graphics boards that can be synchronized. However, this places constraints on the software as theoretically it must run in parallel on different computers and synchronize the graphics output. Several methods were established to achieve this. The framework Chromium (HUMPHREYS et al. 2002) intercepts the OpenGL stream from an application that is running as master, filters and packs the stream and sends it to slave computers which unpack it and perform the rendering. On the one hand this has the advantage of being transparent to the application, so that this does not need to be reimplemented and therefore existing applications for landscape visualization could be used. On the other hand, constantly sending the OpenGL stream from the master node to all slave nodes of the cluster generates a high demand on the network. Distributed scenegraphs like OpenSG (REINERS et al. 2002) keep a description of the model in a tree structure which is distributed from the master node to the slave nodes. Changes to the scenegraph on the master node are propagated to all slaves. The rendering itself is done entirely on the slave nodes. So, for a mostly static scene the network traffic is very low after the scenegraph has been transmitted to the slave nodes. However, the rendering backend of the software must be completely reimplemented. For a further discussion of the pros and cons of the different methods, see EILEMANN (2007) or the Chromium website (CHROMIUM 2008).



**Fig. 2:** The hardware configuration of our visualization center

Some commercial software is also available which specialises in the distributed rendering of static data-bases. But we want to be able later on to adapt the software to specific needs like giving users the option of manipulating the scenegraph (e.g. moving and resizing wind turbines) by providing them with our own user interface or linking different views like a 3D and a map view. We have therefore chosen to experiment with the open-source scenegraph OpenSG as the basis for our work. OpenSG is reasonably well documented for an open source project. It is available for multiple operating systems (UNIX, Windows, Macintosh) and comes with tutorials, which, besides showing other features, provide a valuable starting point for setting up a flat multi-channel projection on a computer cluster with one computer as master. OpenSG provides its own binary file format which includes

all data, such as shaders, images, geometry and structure in one file, which can be useful for presentations. It also separates the structure of the scenegraph from its content, such as materials and geometry, which facilitates the multiple use of repetitive objects like trees. The next part of this paper deals with the workflow we used to generate a digital landscape that can be loaded by the OpenSG file loader and the problems encountered and solutions devised.

### 3 Workflow

Implementing an application that generates the scenegraph sounds work-intensive. However, most of our objects are generated with existing software, such as ArcGIS for the terrain or Cinema4D for the wind turbines, and can therefore be easily loaded from files. So only the assembly process needs to be implemented. For our purpose we assumed that a landscape model was required with dimensions of just a few kilometres. The creation of a digital landscape necessitates a terrain with some form of texture to represent the ground cover and on top of that realistic-looking trees and further models, generated by modelling packages (e.g. the wind turbines or houses). The large number of trees in particular will result in a very complex scene that can only be rendered with sufficient speed if Level Of Detail (LOD) methods and spatial datastructures such as quadtrees are supported. Our initial hope was to use ArcScene and, incorporating the XFrog Plant-Libraries, to generate the whole landscape, then to export it as a VRML file and simply load it into OpenSG. However, using this simple approach did not achieve the visual quality, complexity and rendering performance required. So we had to use a combination of software tools (see Table 1) to pre-process our data step by step and assemble the scenegraph with a self-written program. The following sections discuss these different steps in more detail. A diagram of the overall workflow is shown in figure 3.

**Table 1:** Software used

Software/Library	Source and license conditions
OpenSG	Open source, <a href="http://opensg.vrsource.org/trac">http://opensg.vrsource.org/trac</a>
VRED	Commercial, for non-commercial use to operate on data/scenegraph (see license information), commercial version for clustering, <a href="http://www.vred.org">www.vred.org</a>
Cinema4D	Commercial, <a href="http://www.maxon.de">www.maxon.de</a>
The Gimp	Open source, <a href="http://www.gimp.org">www.gimp.org</a>
XfrogTune	Commercial, <a href="http://www.greenworks.de">www.greenworks.de</a>
XfrogPlants	Commercial, <a href="http://www.greenworks.de">www.greenworks.de</a>
Shapelib tools	Open source, <a href="http://shapelib.maptools.org">shapelib.maptools.org</a>
Nvidia Texture Tools	<a href="http://developer.nvidia.com/object/nv_texture_tools.html">http://developer.nvidia.com/object/nv_texture_tools.html</a>

#### 3.1 Creating the terrain

The input data for the terrain are in an ArcGIS database with the elevation in raster format, the land-use (encoding, for example, deciduous or coniferous forest) in shape format, and

in addition a referenced aerial image from Google Earth™. One problem encountered was that even with a very large texture of 2048x2048 pixels, each pixel represents more than one metre, resulting in a very blocky (low frequency) image near the user. Furthermore using an aerial image as texture would mean that the ground in the forest regions showed the foliage from the top. So a simple solution was required which would enable us to use parts of the areal image and simultaneously replace parts of it with higher resolution images that represent the forest floor or a meadow.

The terrain was created from the elevation grid as a TIN by using ArcGIS and exported as a VRML file. Then the shapes representing land use were classified in ArcGIS and converted to raster format. Pure red indicates deciduous forest, pure green coniferous forest, pure blue lawn and the rest is yellow. The raster data were exported to GIMP, the yellow region selected, its RGB values set to zero and its alpha value to one. The resulting 4 component image encodes land use and was mapped onto the TIN using Cinema4D to generate the texture coordinates. This could also be done using VRED by creating the texture coordinates automatically from the vertex position, and then applying a texture transformation.

Later in VRED several images (forest floor, lawn, aerial and encoding image) are mapped onto the terrain using multitexturing. During rendering, we apply a short shader, written in OpenGL Shading Language (ROST 2004). The fragment shader, a small program that is executed for each pixel before it is output to the framebuffer, uses the colour code to decide from which of these images to sample. Furthermore texture coordinates can be rescaled at this point. This technique allows us to combine periodic images for the forest or grassy floor (using texture tiling) with the aerial image and create a high-resolution textured foreground.

If the user is in the forest or a meadow, the ground texture must have a fairly high resolution which should also show some structure and so not be a pure noise image. Seen from far away, however, the periodicity should not be visible. So, for example, to render grass as ground cover, a mipmap starting with 1024x1024 pixels down to 1x1 pixel is used. The high-resolution image data are generated by using perlin-noise (PERLIN 1985), which is reproducible and periodic, to sample from a photo of a meadow. The low-resolution image data are generated from random numbers. We can do the same with an image of a forest floor. The ground generated this way still looks artificial but is visually more appealing than the blocky structure of the aerial image.

### 3.2 Creating single trees

The XfrogPlants libraries provide highly detailed geometric models of common trees with textures for leaves, trunk and branches, and for billboards. With the software XfrogTune the number of polygons can be reduced from initially more than 100,000 to a few hundreds and a sequence of models with different levels of detail (LODs) thus created. While generating these models and using them in our visualization, we encountered several problems:

1. The leaves of the XfrogPlants models are represented by rectangles with an image of a leaf as texture which is in part transparent. When the distance of the viewer to the leaves increases, the texture appears smaller and smaller and OpenGL must filter and resize it. To

avoid artefacts it is possible to generate a series of prefiltered textures which are called mipmaps (OPENGL 1999). The textures originally provided have no mipmaps, and so the mipmap levels were generated automatically. During this process, colour and transparency were interpolated and the leaves, after several interpolations, became more transparent, so that the tree changed its overall colour. To improve the quality of the visualization, different mipmap images had to be generated using GIMP and assembled to a Microsoft Direct Draw Surface file (.dds), containing all the mipmaps using the NVidia Texture Tools.

2. XfrogTune generates the lower resolution models by firstly simplifying trunk and branches and secondly reducing the number of leaves and making them larger, so that the tree still fills the same amount of space on the monitor. One problem with this approach is that the tree changes its overall shape and the later change to the representation with billboards becomes more noticeable. One possible solution is to show the billboard earlier together with the lower resolution trees. Then the change in overall shape is less obvious.

After exporting the trees from XfrogTune to VRED we assembled a small sub-scenegraph containing the LOD-node and the geometries with different resolutions and the processed textures. As neither VRED nor Cinema4D could read the VRML output from XfrogTune, the Wavefront (.obj) format was used and the materials later fixed by hand using VRED or Cinema4D.

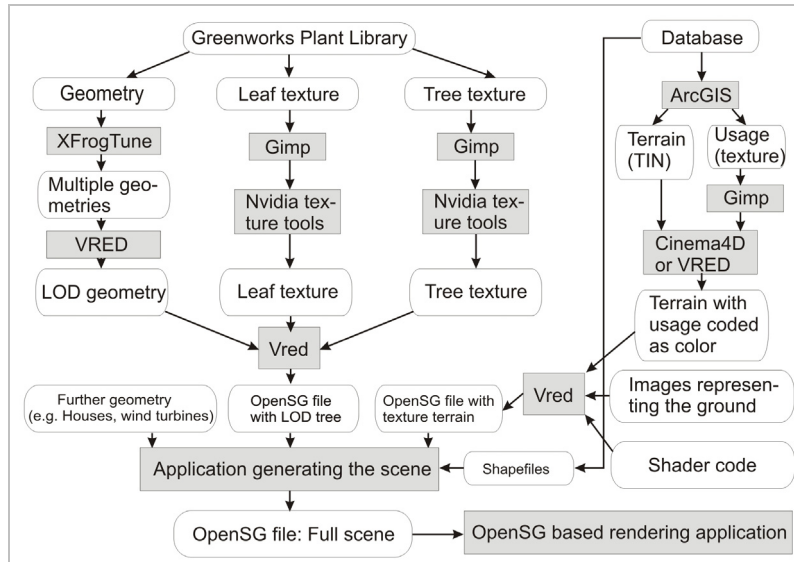
### 3.3 Creating a forest

The inputs for creating a forest are the terrain model and the polygon shapes from the GIS data that define the region it covers. First trees were represented using 6 LODs where the billboard represents the lowest level. As rendering performance is also limited by the number of objects (see CEBENOYAN 2004), this turned out to be too slow. The solution is to use geometric models with 5 LODs for rendering the near field and collect the billboards of one species in one object for rendering the far field as Point Sprites. To generate the forest procedurally several classes had to be implemented:

- CTreeFactory: Performs a so-called shallow copy of the LOD-tree-models. Only the structure of the scenegraph is copied, the geometry and materials stay the same.
- CBillboards: Billboard representation for the trees, renders them as Point Sprites.
- CTerrain: Represents the terrain model - here a TIN, including the textures like areal image and ground images. Has a function that returns the height, given the position.
- CPolygonStencil: Reads one polygon from a shape file, using the shapelib tools. Has a function that computes if a point is inside the polygon, using the ray shooting method as shown, for example, in LASZLO (1996)
- CForest: Stores the different species (a CTreeFactory object) and frequency of their occurrence. Computes a distribution of the species within a given rectangular area. In our experiment we simply use a rectangular grid with random deviation.
- CQuadTree: Implements a quadtree to store given OpenSG objects.

The overall process is simple: within the CForest class a distribution of trees is generated for the bounding box of the given object of type CPolygonStencil. If the generated position

is in the defining polygon, a tree is generated by the corresponding CTreeFactory object and added to the quadtree. Then the position is added to the CBillboards object of the same species. To allow for efficient culling and rendering, the geometric tree-models are organized in a quadtree that switches nodes on and off, depending on the user's distance from them.



**Fig. 3:** Workflow. Grey boxes represent software or libraries, white boxes represent data.

## 4 Conclusions and Outlook

We have experimented with using the open source scenegraph OpenSG for landscape visualization and have described the necessary steps to generate a fairly high quality landscape visualization using this scenegraph. The concepts presented are universal and can also be applied to shrubs, bushes or other objects. This approach is one way to do landscape visualization on a high-resolution tiled display, using a cluster of workstations. One disadvantage of using a scenegraph in a simple way is that the scene has to be kept in memory, even the parts and LODs that are currently not rendered. However, the example scene generated represents an area of approximately 2800 by 2800 metres and contains 22,000 trees which are represented by several species at different ages, the TIN and the necessary textures for the terrain. This scene uses approximately 1.2 GB of memory and could be rendered with more than 10 frames per second. The visualization quality is high and viewers visiting our centre were impressed. Given suitable optimization, even more detail is possible. For us, the use of OpenSG for landscape visualization on our display is therefore a very good solution.

## 5 Acknowledgements

I would like to thank Michael Vieweg for processing GIS and image data and for doing modelling work and Dr. Alison E. Martin for proofreading the manuscript. I also thank the members of the BMBF Project “Nachhaltige Landnutzung im Spannungsfeld umwelt-politisch konfligierender Zielsetzungen am Beispiel der Windenergie” for many interesting and inspirational discussions.

## References

- CALP (2008), *Collaborative for Advanced Landscape Planning*.  
<http://www.calp.forestry.ubc.ca/>.
- Cebenoyan, C. (2004), *Graphics Pipeline Performance*. – In: GPU Gems, Edited by Randima Fernando. – Addison Wesley, Boston.
- Chromium (2008), *Chromium Documentation Version 1.9 – An Overview of Chromium*.  
<http://chromium.sourceforge.net/doc/index.html>.
- Eilemann, S. (2007), *An Analysis of Parallel Rendering Systems*.  
<http://www.equalizergraphics.com/documents/ParallelRenderingSystems.pdf>.
- Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P. D. & Klosowski, J. T. (2002), *Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters*. – In: ACM Transactions on Graphics, 21(3): 693-702.
- Laszlo, M. J. (2004), *Computational Geometry and Computer Graphics in C++*. – Prentice Hall, Upper Saddle River.
- Macaulay (2008), *Virtual Landscape Theatre*.  
<http://www.macaulay.ac.uk/landscapes/>
- Meinel, G. & Walter, K. (2001), *Visualisierung geplanter Windkraftanlagen im Rahmen der Landschaftsbildbewertung – Möglichkeiten und Grenzen*. – In: Albertz, J. (Ed.), Rauminformationen für das 21. Jahrhundert. Vorträge, 20. Wissenschaftl.-Techn. Jahrestagung d. DGPF, 11.-13. Oktober 2000. – Berlin: 364-374 (= Publikationen der Deutschen Gesellschaft für Photogrammetrie, Fernerkundung und Geoinformation, 9).
- Perlin, K. (1985), *An Image Synthesizer*. – Computer Graphics (Siggraph 85 Proceedings, Association for Computing Machinery, 287-296.
- Reiners, D., Voß, G. & Behr, J. (2002), *OpenGL: Basic Concepts*. – OpenGL Symposium 2002  
<http://publica.fraunhofer.de/starweb/servlet.starweb?path=pub.web&search=N-9605>.
- Rohlf, J. & Helman, J. (1994), *Iris Performer: A High Performance Multiprocessing Toolkit for Real-Time 3-D Graphics*. – Proceedings of SIGGRAPH 94 Conference, Association for Computing Machinery (ACM). – New York: 381-395.
- Rost, R. J. (2004), *OpenGL Shading Language*. – Addison-Wesley, Boston.
- Woo, M., Neider, J., Davis, T. & Shreiner, D. (1999), *OpenGL Programming Guide (3. edition)*. – Addison-Wesley, Boston.

