Scalable visualization of massive point clouds

G. de Haan
TU Delft, Computer Graphics & CAD/CAM group, the Netherlands

Abstract

Graphical renderings of "raw" point clouds can be visually attractive. When combined, the many individual points and their attributes convey spatial structures, especially when viewed in a fluent 3D fly-through and on a stereoscopic display. For such interactive visualization, a sufficiently responsive graphics update is essential. Where many software packages support simple rendering of smaller (subsets of) point clouds, the size of point clouds currently acquired easily surpasses their rendering capabilities, even on a modern graphics card. We addressed this issue in recent experiments on the visual exploration of LiDAR (Light Detection And Ranging) datasets in our Virtual Reality systems. Out-of-core terrain rendering techniques are applied, which originate from graphic rendering engines used in games and flight-simulators. In this document some of these techniques and results are highlighted, and challenges in balancing rendering performance versus visual quality are discussed.

1. Introduction

Acquisition of real-world scenes using LiDAR (Light Detection And Ranging) or other 3D scanning technologies produces enormous quantities of 3D data, typically in the form of large, unstructured point clouds. The 3D visualization of this raw data and its derivative features is often necessary to perform visual (quality-) inspection and to perform interactive selection and manipulation operations in 3D. But foremost, a 3D visualization empowers the human visual perception; it provides a recognizable and intuitive spatial context and is essential for the presentation of data to a non-expert audience.

Datasets of sizes up to some hundred megabytes (several millions of points) can be readily visualized using standard tools on commodity hardware. However, even scans of rather small environments can easily lead to large data files consisting of billions of individual points. An attempt to render these datasets with standard tools often result in low, non-interactive frame-rate or simply exceed the machine's limits in terms of main memory and graphics memory. In addition, it is in many cases necessary to have the data reside on a central server and only allow visualization from remote machines.

For these reasons, a *scalable* approach for visualization should be able to optimize the visual quality and interactivity given the limitations of the available rendering and network resources. To achieve this, powerful *out-of-core* data structures and rendering algorithms are needed. We consider the following three main scalability challenges of an interactive point cloud visualization system: *rendering, visualization and interaction*.

For the *rendering* part, a special data structure has to be designed that not only allows for high-quality real-time rendering but also supports editing operations. The rendering subsystem has to be scalable in terms of available processing power and main memory and the data structure should support methods for streaming the content to remote clients. For providing insight in all the properties and characteristics of the data, *visualization* techniques need to be developed that are able to visually represent the various features present in the data, e.g. comparison of two datasets in time. Finally, human-machine *interaction* issues need to be addressed. That is, the user has to be able to intuitively navigate, select, annotate and manipulate parts of the 3D data.

The current work in progress on these challenges is discussed in the remainder of this document. First, the architecture of our current prototype visualization system is described in Section 2 and 3. Here, some basic methods and data structures for scalability in terrain rendering are explained. Sections 4, 5 and 6, describe the integration of point clouds in our visualization system and the visual properties. We demonstrate some of the current results on visual quality and on performance in various point cloud datasets in Section 7. In Section 8, we highlight some related work and systems in the area of scalable rendering and point cloud visualization. Finally, in Section 9, we discuss current limitations and new challenges of current systems and indicate possible future directions of research and development.

2. Current visualization system

A schematic overview of the flow of data and its processes is shown in Figure 1.

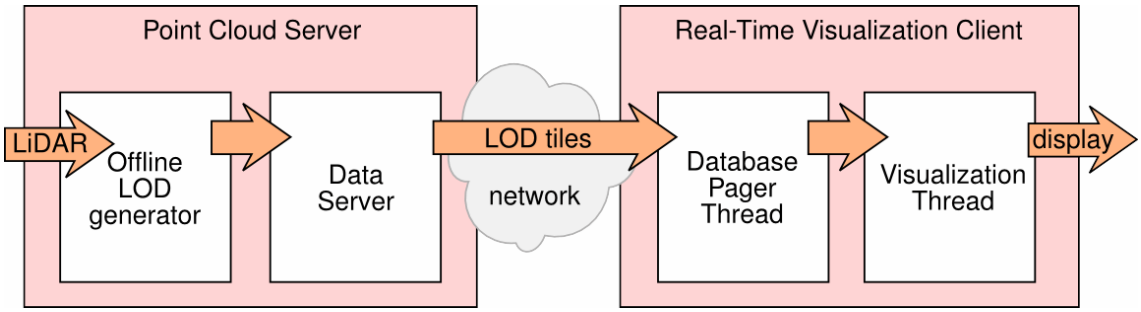

Figure 1: Point cloud visualization pipeline overview. The LiDAR files are first processed into LOD (Level-of-Detail) tiles before they can be used by the system.

For visualization we use "VRMeer", an open source software toolkit developed at the TU Delft Computer Graphics Group. It is used to prototype and to develop interactive Virtual Reality applications. It builds on features of the OpenSceneGraph (OSG) 3D graphics rendering toolkit. This is a modern *scene graph* engine and accompanying 3D toolkit that is widely used, under active development and open-source. The VRMeer toolkit adds support for various 3D display systems and 3D input devices and basic 3D interaction techniques. Although the base programming language is C++ for both OSG and VRMeer, we constructed wrapping layers for the Python programming language for both. This allows for a more high-level prototyping of new functionality and applications. As many other software toolkits and libraries for GIS data, such as the Geospatial Data Abstraction Library (GDAL), come with Python interfaces, this facilitates fast integration and linking of general 3D visualization applications with GIS specific data formats.

3. From point clouds to terrain rendering

Techniques for point cloud visualization have many parallels to terrain rendering techniques which are used in modern games and high-performance flight-simulators (e.g. SGI OpenGL Performer). OpenSceneGraph also provides the functionality for rendering large terrain models. These real-time, out-of-core terrain rendering techniques are created by combining level-of-detail (LOD) scene graph nodes and the paging of data in- and out of memory from disk or network. In this section a brief outline is given of the functionality of this mechanism.

In an OSG-based visualization client, the *visualization thread* determines which parts of the scene graph is visible, and enables for each node the suitable LOD level for this viewing distance. If this LOD level is not yet available in memory (e.g. it resides on disk), the *Database pager thread* is requested to load this data. Without visibly pausing the application, it will start loading the data from directly disk or from network in the background and insert the graphical

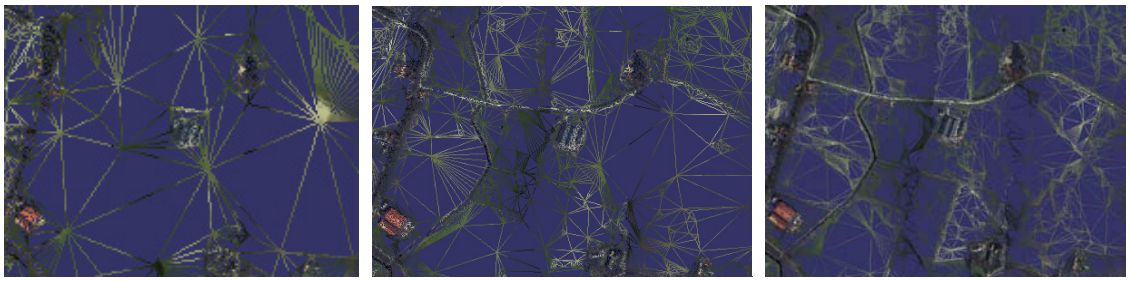

Figure 2: Screenshots in wireframe mode of the same area in the AHN2 terrain as seen from a large viewing distance (left, scaled) through to a small distance (right). As the viewing distance decreases, a different level-of-detail is visible with more accurate triangulation and higher resolution texture information.

node in the scene graph when it becomes available. At the same time, those nodes that are not visible in the scene are scheduled for removal from system memory.

OSG is accompanied by VirtualPlanetBuilder (VPB), a software tool to generate these terrain models. This program intelligently combines (several tiles of) Digital Elevation Maps (DEMs) and textures into a Triangulated Irregular Network (TIN) model at several levels of detail in a pre-processing phase. The final model contains scene graph nodes, each of which in turn contains terrain nodes with discrete level-of-detail tiles as its child-nodes. We used VPB to process AHN grid data (generated from filtered point clouds) which is available from the AHN test set. This set consists of several tiles in the ESRI ASC format (e.g. gridded DEM with 0.5m sized grid cells, height value in meters for each cell) and raw RGB data acquired by the LiDAR scanner in a geo-referenced GeoTIFF format. This data can be fed directly to VPB, whereas the necessary format conversion and coordinate system transitions are performed by the GDAL toolkit. As a result, VPB generates binary OSG files (IVE format). Several quality and terrain settings can be selected. Depending on the amount of tiles and required quality of the results, generation may take between several minutes to several days.

The resulting terrain model can be loaded directly in any OSG model viewer or our specialized, VRMeer based terrain viewer, see Figure 2. The visual quality differs with the distance the viewer takes to the landscape, and the spatial frequency of landscape features. A consistent frame rate of 60 frames per second is achievable during 3D exploration, without disturbing artefacts when loading in different levels of detail.

4. From terrains to point cloud rendering

When using a DEM without filtering (c.f. AHN2 unfiltered), buildings appear to be “draped” with a sheet, see Figure 3 (left). This is caused by the interpolation between neighbouring grid cells. The results are most visible in areas where the continuity of the terrain is disturbed by structures on top of the terrain, such as buildings and trees. However, raw LiDAR measurements contain a wealth of 3D information, of which much is lost in the post-processing steps that are used to generate DEMs. In the AHN case, a point cloud consists of approximately 20 million points with XYZ coordinates (floating point number, RD coordinate system, cm notation, ~3 cm accuracy). In our work we aimed to directly visualize these point clouds without removing information or introducing new sampling artefacts. When this point cloud is directly converted to a scene graph model by using a graphical point primitive for each point, it provides a rich visual scene with different visual characteristics, see Figure 4 (right). However, frame rates will drop far below interactive rates, e.g. to one frame per second. To still maintain sufficient interactive performance, the point clouds need to be pre-processed in special scalable data structures, in a similar fashion to the TIN models as described above.

Figure 3: Visual comparison of two visualization techniques of the same area of the AHN2 dataset. The triangulated mesh (left) is generated from the unfiltered, gridded DEM and suffers from "draping effect", most visible in trees and buildings. The point cloud (right) is generated from the unfiltered point cloud and more directly depicts LiDAR measurements. It does suffer from see-through effects and "holes", most visible around and underneath buildings.

5. Data structures

We have developed a point cloud pre-processing mechanism which generates OpenSceneGraph data structures for point clouds. These data structures can be paged into memory and allow control of the levels-of-detail. In this section, some of the characteristics of this processing and data structure are discussed.

In a similar fashion to what the VirtualPlanetBuilder does for TINs, the pre-processing program constructs *a hierarchical tree structure* in an OSG scene graph, and *distributes* the available points of the point cloud over the nodes at different levels in this tree. For each node in the tree containing a subset of the points, it's LOD distance is set. When viewed from a distance, the level-of-detail algorithm displays only the higher level leaves, so only a subset of points in the tree are visible. A careful *balancing* of number of points and the number of levels of detail is necessary to avoid loading and rendering too much data.

First, we select a hierarchical data structure for spatial subdivision depending on the characteristics of the point cloud. We currently support a regular binary tree, a quad-tree or an octree. For large, terrain-like datasets a quad-tree is the most efficient of these options, whereas for a detailed scan of a single building an octree would be more suitable. The depth of the tree is currently estimated based on the density of points in the source LiDAR file, and the expected detailed viewing distance. We would like to balance the number of points in each node to minimize the overhead of the hierarchical structure, and not have too many points in a single node. Based on the domain size and tree depth, the hierarchical data structure is created in an OSG scene graph and the spatial extents of the individual bounding boxes are generated.

Second, for each point in a point cloud, we determine in which spatial tile it should be placed in. Here, in contrast to many other LOD approaches, *points are not duplicated* in order to save memory and bandwidth capacity. This approach was selected because we can render complementary point clouds that completely overlay each other, we consider this a special feature of point clouds. As a result, we not only place points in the leaf nodes of the LOD tree but also in all of the branch nodes.

To determine how high in the LOD hierarchy a point should reside, a *sampling strategy* is used. The goal of a sampling algorithm is to have a representative subset of the points in each of the LOD levels. As a result, if only the lower LOD level (largest distance) is rendered, a "reasonable" impression of the overall measured structure should be obtained. Once the LOD level of

the point concerned is determined, it is placed in a point set in the corresponding LOD level for that spatial subsection of the domain.

We currently use one very basic sampling mechanism that uses a bitwise interleaving scheme on the explicit ordering of the points in the point cloud. For example, every 16th point is placed on the lowest LOD level, every other 8th point is placed on the second lowest level, until the remaining points are placed on the correct, higher and more detailed LOD levels. If the measurement process is a complete random sampling of the structure and unordered, this approaches a random sub selection. It can be calculated what the expected number of points and density will be in the resulting LOD tiles. In the discussion section other suitable alternatives to this approach are discussed.

Per individual point we can also store available attributes. Examples of these are measured point color or intensity. We can also use ortho-photos to color individual points by performing a color look-up for each point in the corresponding photograph at its x and y location. A large downside to per-point attributes is the increase in data-size, e.g. a 3-float RGB component adds 12 bits per point, doubling the data size with respect to only position encoding.

6. Visualization

As soon as the point cloud tiles are loaded in the scene graph, they are visualized on-screen. It can be rendered concurrently with other (geo-) data and interface elements.

The visual representation of the point samples is done through rendering OpenGL point primitives. Only a limited perception of depth can be obtained from a single 3D point. However, when many points are grouped, visual depth cues of perspective, visual compression, motion parallax and occlusion give a compelling spatial impression.

The drawback of point clouds is that they do not necessarily give a continuous or filled visual representation, e.g. no sampling gaps are filled. This is not a problem when viewing from a distance, but can be if observed from close by. This impression may be improved when a point is rendered larger than single pixel. For this, a graphics *shader program* can determine the virtual size of a point to generate a larger (or smaller) *splat* on-screen, based on the distance to the viewer. This improves the depth perception of individual points, especially when they are closer to the viewer as they scale with motion and occlude other points and may fill holes. Also a *fogging* component is included to simulate *haze*, which emphasizes distance by decreasing the saturation of colors when points are more in the distance. If no color is present in the data, pseudo coloring can be used, e.g. to map a point's height to a color value, e.g. see Figure 4.

A combined representation of both raw point clouds and a terrain model generated from filtered data may provide a reasonable impression of a landscape with sharp features such as buildings. However, we did observe disturbing effects which appears to be a visual shadow or ghost. It suffers from see-through effects and “holes” in the point cloud, through which the same color is also seen on the ground surface. This is mostly visible around and underneath buildings, an could be solved by removing colors on the ground surface when buildings are present. We can also integrate 3D surface models of buildings that were modeled separately or that were reconstructed from the point cloud.

7. Resulting interactive visualizations

The techniques described in the previous section were applied for the interactive visual exploration of several LiDAR datasets in our Virtual Reality systems and demonstrated on numerous occasions. In addition to regular desktop visualizations, we can use *stereoscopic* rendering and 3D interaction in our Virtual Reality systems. The stereoscopic rendering can dramatically em-

phasize depth perception for complex point clouds which are not dense or lack structural or color information features.

General 3D viewers often do not include good interaction techniques to study 3D topography data. Google Earth is an example of a specialized viewer for topography content. However, this application has a main drawback that it cannot simply load custom terrain models and/or point clouds. The main aspect of this viewer is the 3D navigation. We created simplified mouse based view navigation, as well as navigation with specialized devices such as the *Space Mouse* and the *Wii Balance Board*. On more immersive VR systems, such as a Workbench or CAVE system, one can also use head tracking and 3D pointing devices. By using our Python-based prototyping system, the application can be quickly reprogrammed to include case-specific viewing techniques, specialized interfaces, or loading of models. As an example, we included mouse-based sketching to quickly draw sketches on a 3D terrain for communicating details.

Figure 4 depicts the use of a point cloud based visualization integrated with a flooding simulation. Although the simulation used here is non-realistic and simplistic, the visual effect and the 3D context of the information within the point cloud is found compelling. We are currently planning to integrate realistic simulation results from running the SOBEK Flooding Simulation from Deltares, and in the future to allow interactive steering of these simulation runs.

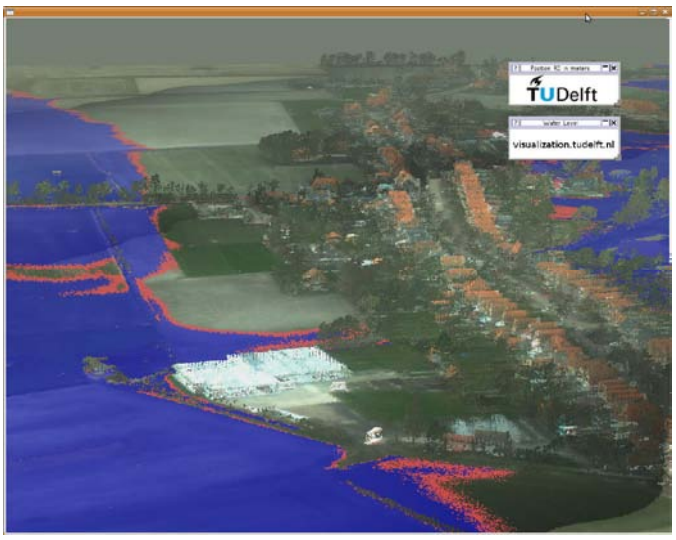

Figure 4: Screenshot of a point cloud -based visualization integrated with a simplistic flooding simulation. The use of per-point color, fogging, and larger point-sizes increases the perception of depth and context of simulation results.

8. State-of-the art: beyond scalable rendering to scalable visualization

Various techniques have been presented for out-of-core point cloud visualization, e.g. Kreylos et al. (2008a) and Wand et al. (2008) make use of a multi-resolution spatial data structure. In a pre-processing step the raw data is analyzed and written to disk in a special format that allows for efficient reloading of higher resolution parts of the scene. Both methods achieve interactive visualization of raw data and were demonstrated with sizes of up to 63 GB. The system presented by Kreylos et al. (2008a) allows for simple editing operations whereas the XGRT system presented by Wand et al. (2008) is a proof-of-concept of a complete tool chain for 3D data processing. In both systems editing of the scene that leads to reasonably small changes in the overall scene structure can be performed in real-time. A different method for extremely fast and efficient visualization of terrain data based on a regular grid is presented by Bhattacharjee et al.

Figure 5: Screenshot of the AHN2 dataset (filtered buildings, no color) surrounding TU Delft campus, ~6 tiles in view at 20 fps @ 1680x1050 (mono), nVidia Geforce 280 GTX. Performance when viewing 100 tiles currently drops to 5 fps.

(2008). In this work, most parts of the rendering pipeline are outsourced to the graphics card (GPU) and high frame rates are achieved on small-to-medium sized data sets. The method is only suited for structured point clouds (height maps) and does not support out-of-core data.

In the field of computer graphics and visualization, much work addresses issues of performance of data handling, data processing and rendering. However, equally important are visualization techniques; these are responsible for adding abstract or derived features to the rendering. Especially with Point Cloud datasets, elements of uncertainty, variability and the detection of changes are of features interest that require visual representation, see also the work by Pauly et al. (2004) and Girardeau et al. (2005). In their work, Butkiewicz et al. (2007) demonstrate the possibilities of visual analysis for exploring change detection based on multiple LiDAR scans. Recent developments demonstrate various new ways of displaying feature and abstract information cross-linked with geo- information (such as point clouds), see Butkiewicz et al. (2008). In addition, the possibilities to interactively act on the data enable users to navigate, select, annotate and manipulate parts of the data. For example, Kreylos et al. (2008b) demonstrate the use of 3D interactive visualization for use in a geoscience scientific workflow. Here, large datasets are visually explored and potential hypotheses are generated, where the 3D perception and intuitive interaction with the data plays an important role. Fuchs et al. (2009) propose an interactive cycle of knowledge-based analysis and automatic hypothesis generation. In this work, starting from initial hypotheses, created with linking and brushing, the user steers a heuristic search algorithm to look for alternative or related hypotheses. The results are analyzed in information visualization views that are linked to the volume rendering. Individual properties as well as global aggregates are visually presented to provide insight into the most relevant aspects of the generated hypotheses.

9. Discussion and future directions

Our current approach as of now can be made scalable enough to visualize the full AHN2 set, under special circumstances that is. However, one currently needs to manually select several parameters for LOD tree depth and sampling interleaving. This first results in much manual tweaking and unnecessary processing of failed runs. In addition, it will not provide a scalable solution in terms of lower bandwidth and slow clients or the updating or editing of new infor-

mation in tiles. It is under current investigation how the individual parameters influence performance and visual quality. We plan to provide more flexible approaches for sampling the data and develop metrics. An interesting approach would be to allow continuous Level-Of-Details and streaming of data, based on the performance of the network and the client. This should entail a better sampling strategy or defining an *ordering* in the importance of specific points.

Another topic of interest is the adaptive sampling strategy for the various levels of details. The current basic sampling strategy does not produce convincing results on some datasets. For example see Figure 6, where a dataset has regularity in point ordering, causing aliasing, or is not heterogeneous in terms of sampling density, causing holes or too dense areas. To the latter effect, for example, local point size could be increased when sampling density is low.

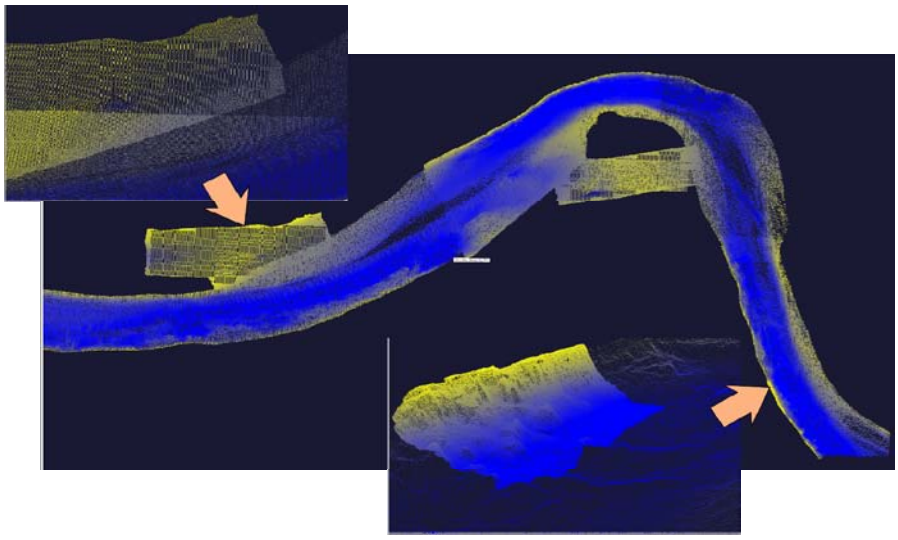

Figure 6: Point Cloud of a Bathymetry dataset. The inset top-left indicates aliasing errors caused by the regularity in data and the sampling algorithm, the other inset shows a region where point sampling density is much larger than its surrounding area.

An advantage of our OpenSceneGraph-based system over custom high-performance rendering approaches is that it benefits from compatibility, maturity and new developments in and surrounding the toolkit. For instance, currently new work is being done to integrate OpenGL ES support, which will enable small or embedded rendering clients such as mobile devices. Another interesting development is the integration of more mature GIS functionality within this framework, such as is being done through projects like OSGgis, OSGearth. An alternative development would be that current popular GIS applications adopt similar 3D viewing components in their existing applications.

In conclusion, the main objective of future work in this area is to both design fundamental algorithms and develop software assets which allow visualization and interaction on all the 3D data in a point cloud geo-database. More specifically, the objectives concern rendering (software methods and framework for scalable pre-processing and rendering), visualization (definition and evaluation of novel (human-supervised) feature extraction and feature visualization methods) and interaction (definition and evaluation of interaction techniques for 3D navigation, selection, annotation and manipulation).

Acknowledgements

The AHN2 dataset is copyright of RWS. The bathymetry dataset is copyright of RWS Zeeland.

References

Bhattacharjee, S., Patidar, S., Narayanan, P. J.: *"Real-Time Rendering and Manipulation of Large Terrains."* , ICVGIP, 2008.

Butkiewicz, T., Chang, R., Wartell, Z., Ribarsky, W., *"Visual Analysis and Semantic Exploration of Urban LIDAR Change Detection."*, Computer Graphics Forum, 2007.

Butkiewicz, T.; Dou, W.; Wartell, Z.; Ribarsky, W.; Chang, R.; *"Multi-Focused Geospatial Analysis Using Probes"*, IEEE Transactions on Visualization and Computer Graphics, Volume 14, Issue 6, Page(s): 1165 – 1172, Nov.-Dec., 2008.

Fuchs, R., Waser, J., Gröller, E., "*Visual Human+Machine Learning*", IEEE Visualisation, 2009.

Girardeau-Montaut, D., Roux, M., Marc, R., Thibault, G., *"Change detection on point cloud data acquired with a ground laser scanner"*, 2005.

de Haan, G., Griffith, E.J., Post, F.H., *"Using the Wii Balance Board(TM) as a Low-Cost VR Interaction Device"*, ACM Virtual Reality Software and Technology (VRST), 2008.

de Haan, G. and Post, F. H., *"StateStream: a developer-centric approach towards unifying interaction models and architecture"* In Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS), 2009.

Kreylos, O., Bawden, G. W., and Kellogg, L. H., *"Immersive Visualization and Analysis of LiDAR Data"*, In Proceedings of the 4th international Symposium on Advances, 2008.

Kreylos, O., Bawden, G., Bernardin, T., Billen, M. I., Cowgill, E. S., Gold, R. D., Hamann, B., Jadamec, M., Kellogg, L. H., Staadt, O. G., and Sumner, D. Y., "*Enabling Scientific Workflows in Virtual Reality"*, Proceedings of the 2006 ACM international Conference on Virtual Reality Continuum and Its Applications (VRCIA), 2006.

Pauly, M., Mitra, N.J. , Guibas, L.J. , *"Uncertainty and Variability in Point Cloud Surface Data"*, Eurographics Symposium on Point-Based Rendering, 2004.

Wand M., Berner A, Bokeloh, M. , P. Jenke, A. Fleck, M. Hoffmann, B. Maier, D. Staneker, A.Schilling, H.-P. Seidel: *"Processing and Interactive Editing of Huge Point Clouds from 3D Scanners"*, Computers and Graphics, 32(2), 204-220, 2008.

Web references

[OSG] OpenSceneGraph, http://openscenegraph.org

[OSGEarth] http://wush.net/trac/osgearth

[OSGGis] http://wush.net/trac/osgearth