Optimizations in the Grid Visualization Kernel

Dieter Kranzlmüller, Gerhard Kurka,
Paul Heinzlreiter, Jens Volkert

TR-GUP-02-007

Johannes Kepler Universität Linz,
Institut für Technische Informatik und Telematik
Abteilung für Graphische und Parallele Datenverarbeitung

Optimizations in the Grid Visualization Kernel

Dieter Kranzlmüller, Gerhard Kurka, Paul Heinzlreiter, Jens Volkert
GUP Linz, Joh. Kepler University Linz
Altenbergerstr. 69, A-4040 Linz, Austria/Europe
kranzlmueller@gup.uni-linz.ac.at

Abstract

Computational grids are large-scale computing infrastructures, which offer ubiquitous access to networked resources for integrated and collaborative use by multiple organizations. The required abstraction and management for grids is transparently performed in the grid middleware layer. An example is the Grid Visualization Kernel GVK*, which supports distributed near real-time interaction between distant simulation servers and visualization clients. Efficient transportation of data via limited and varying bandwidth is achieved by applying optimized abstraction and filtering techniques. The basic idea is to incorporate knowledge about visualization data structures and user viewpoints before transferring the data between server and client. Corresponding ideas include optimizations like multiple levels of detail, occlusion-culling, reference and image-based rendering.*

1. Introduction

Grid computing denotes distributed computing infrastructures for advanced science and engineering [4]. Within a persistent environment, grids enable new kinds of software applications by integrating distinct computational and information resources at diverse wide-spread locations. The geographical distribution of processing capabilities requires novel approaches to access and process data.

The basic technology for exploiting computational grids is grid middleware, which is provided by toolkits such as Globus [3], Legion [8], and UNICORE [15]. Such grid middleware toolkits offer information infrastructure services for security, communication, fault detection, resource and data management, portability, and more. With the availability of these basic services raises the demand for advanced presentation technology, which allows to interactively explore the data processed within grids. In particular, more and more grid-enabled applications require near real-time visualization of simulation data, which are easy to use (for the purpose of the grid) and specifically adjusted to the characteristics of grid infrastructures.

The Grid Visualization Kernel GVK[1] represents a corresponding middleware extension, which allows to interconnect the data sources, the simulation processes, and the visualization clients at different, possibly far away, locations. On the one hand, GVK hides the partitioning and scheduling of the application, which may dynamically change over time. On the other hand, GVK offers a transparent interface for simultaneous visualization clients with different output devices, which may be activated anywhere on the grid.

The connection between the server application and the visualization client faces several problems, such as heterogeneity, limited and varying bandwidth, the possibility of communication failures, and the dynamics of the grid environment itself. The approach of GVK is to adapt the visualization pipeline according to the actual characteristics of the network. By using optimized visualization and rendering techniques at different places between the computational source and the visualization client, the characteristics of the visualized object can be adjusted accordingly.

This paper describes the problems of real-time visualization on the grid and the optimized visualization pipeline configurations applied in GVK. Section 2 sketches the proposed grid middleware extension and discusses the transportation problems of visualization in context with grids. Three example pipeline configurations are introduced in Section 3, which demonstrate the feasibility of the GVK approach. Afterwards, a short summary and an outlook on future work in this project is given.

2. Real-time visualization on the grid

2.1 A middleware extension for visualization

The design of a grid-enabled visualization system faces three main problems:

[1]The Grid Visualization Kernel (GVK) described in this paper is partially supported by the IST Programme of the European Commission under contract number IST-2001-32243.

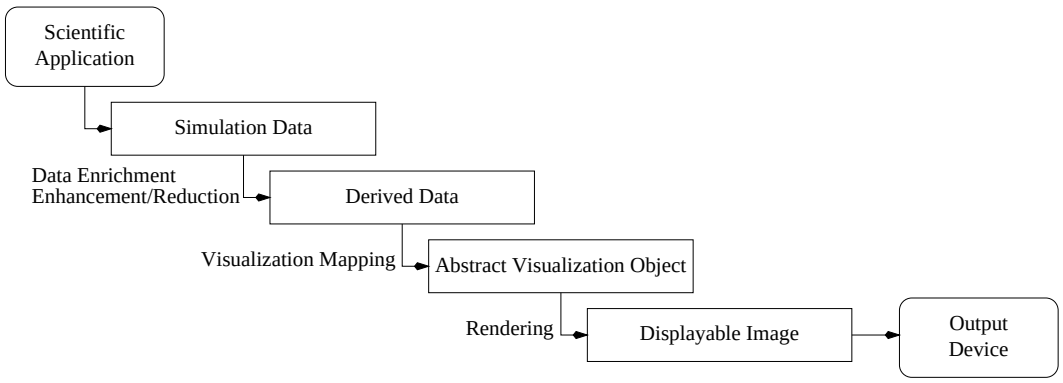

Figure 1. Visualization Process [9]

- Overcoming heterogeneity of the system.
- Maximizing the system's throughput.
- Minimizing its latency.

Heterogeneity is an intrinsic problem of grid infrastructures, which have to cope with a wide range of different hardware and software components. Throughput is important for generating visual representations of time-evolving data while the data is generated by the simulation process. Latency is of major importance whenever interaction and feedback between visualization and simulation are established.

The problem of heterogeneity is typically solved by well-defined interfaces, which hide the complexity of the underlying architecture from the user. For grid infrastructures, the abstraction is achieved in the middleware layer that connects the applications to the system. With the availability of middleware solutions for different aspects of grid computing, it is feasible to implement a grid-enabled visualization system as a kind of middleware extension.

The functionality of GVK is available as a set of input and output modules for well-known visualization packages. Currently the Open Visualization Data Explorer *OpenDX* (`http://www.opendx.org/`) is supported, while an implementation for AVS (`http://www.avs.com/`) is underway, and a possible integration of the Visualization Toolkit VTK (`http://public.kitware.com/vtk/`) is investigated. Each of these visualization packages supports a data-flow model and an application programming interface (API), which allows to develop customized extensions to the provided visualization functionality.

By permitting the usage of existing and well-known visualization techniques, application development should be accelerated and simplified, because users do not need to learn "another" visualization toolkit. Instead, the connection to GVK is established (on both sides) by using an input interface at the simulation server and a corresponding output interface at the visualization client. This means, that the user includes the visualization in the simulation process by specifying it with the usual (and established) visualization practices. The difference is, that the last module (on the simulation server side) and the first module (on the visualization client side), respectively, is an interface module to GVK.

Besides including the interface modules, the user must also specify a description of the visualization, so that the operation of GVK can be adapted to the characteristics of the visualization data. This includes a meta definition of the minimum required output characteristics (e.g. resolution, color-depth, ...), but also some details about the visual content. The latter is needed to enable the optimizations of the visualization pipeline as described in Section 3. At present, the description is defined via some pre-defined data structures, which limits the approach to a small subset of all possibilities. However, a more advanced method based on a specialized visual content description language is currently being implemented.

With the interface modules and the description language, arbitrary interconnections between multiple servers and clients can be established, because the actual interconnection, transportation, and (if required) multiplexing is handled transparently inside GVK. The same applies for the remaining problems of grid-enabled visualization systems - throughput and latency - which are handled automatically within the kernel.

2.2 Performance issues

The main goal of maximizing a system's throughput is effective usage of the available network capacity. The important issues are data selection, display rate, distance between data producer and rendering consumer, and data compression. Data selection is determined by the target data

set and its data structures, as well as the viewpoint of the human user. The demanded display rate is determined by the kind and speed of user interaction, and the number of different views on the visualization device. The distance between producer and consumer is given by the network interconnection between the simulation machine and the output device, which may vary over time since endpoints may be changed dynamically. Data compression uses traditional techniques to reduce the number of bytes in the pure data streams by packing the data at the source and unpacking it at the destination.

The latency of the visualization is experienced by the user as the delay between interactive commands and the system's response. If the complete data set is available at the visualization site, latency is mainly determined by the performance of the rendering hardware. In this case, interaction and exploration is performed locally and not affected by the underlying network. However, in many simulations, the visualized data set changes over time and requires constant network transportation. Furthermore, the huge amounts of data may not be manageable at each visualization client in the grid, and the delay during startup is often unacceptable.

2.3 Traditional visualization pipeline configurations

In order to overcome these problems, GVK optimizes parts of the visualization process (as shown in Figure 1) for the specific characteristics of the grid. The three standard transformations required to generate a visual representation for a given scientific application on a corresponding output device are as follows [9]:

- *Data enrichment/enhancement/reduction (=Filtering)* operates on the raw data to derive data required for visualization (extraction, interpolation, approximation, smoothing, ...).
- *Visualization mapping* constructs an imaginary Abstract Visualization Object (AVO) and maps the derived data onto the AVO's attributes (geometry, time, color, transparency, luminosity, texturing, ...).
- *Rendering* produces the displayable image from the AVO by means of computer graphics and image processing methods (view transformations, hidden surface removal, shading, antialiasing, ...).

The idea is to move parts of the visualization process away from the visualization client towards the simulation server by redesigning the visualization pipeline. The throughput and latency can be controlled by choosing different partitioning techniques. In general, the following pipeline configurations are used in most cases:

(1) **Client**: Filtering + Visualization + Rendering

(2) **Server**: Filtering
Client: Visualization + Rendering

(3) **Server**: Filtering + Visualization
Client: Rendering

(4) **Server**: Filtering + Visualization + Rendering

Approach (1) corresponds to the conventional situation, where all processing is done at the client, which receives the input data from the remote server. In approach (2), the data sent to the remote visualization client is reduced to those fractions actually needed for the output. For example, when visualizing vector flow fields as flow-ribbons, only the positions, directions, and lengths of the computed vectors are transferred instead of the complete data set. Consequently, this approach is restricted to visualization techniques, that rely on sparse and simple data structures, which can easily be obtained by preprocessing the input data set.

In approach (3) and (4), the server produces the AVO by implementing all pipeline stages until polygonal objects for the final output are generated. Afterwards these AVOs are rendered either at the client or at the server. When remote rendering is performed as in approach (3), polygonal data can be cached at the visualization site with some kind of display-list. In this case, subsequent rendering cycles do not need to transfer the complete data again, if only viewing parameters (position, perspective, and direction) are changed. Instead, only local transformation matrices need to be modified at the client and the cached polygon lists can be displayed.

The drawback of this approach is the size of the polygon lists, which tend to become rather large, e.g. when creating multiple iso-surfaces or visualizing meshes. As a result, the transfer of the (compressed) geometry data may demand lots of communication bandwidth again. In approach (4), only completely rendered images are sent from the server to the visualization site. This approach is well suited for situations where large polygonal data sets have to be sent to the visualization site. Its disadvantages are high latencies with respect to viewpoint changes, because each image has to be fetched from the server site.

3. Optimized pipeline configurations

3.1 Overview of optimization techniques

In addition to the traditional techniques described above, GVK implements the following enhanced techniques:

(2a) **Server:** Level-of-detail filtering
Client: Visualization + Rendering

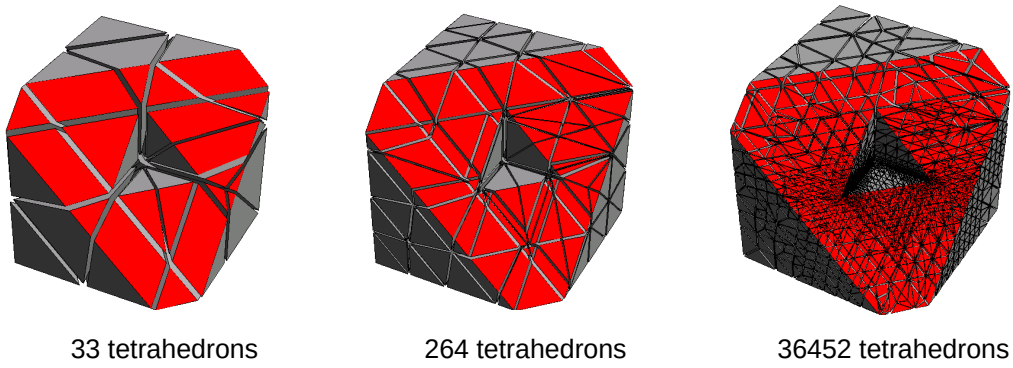

Figure 2. Multiple Levels of Detail

(3a) **Server:** Filtering + Visualization + Occlusion-culling
Client: Rendering

(4a) **Server:** Filtering+Visualization+Reference rendering
Client: Image-based rendering

3.2 Level-of-detail filtering

Filtering by choosing a corresponding Level of Detail (LOD) is a common technique to tackle polygonal complexity among computer graphics related fields [1, 18]. In case of multiresolutional meshes, as used by a wide variety of numerics codes, the mesh itself has an inherently hierarchical structure. The provided hierarchy can be exploited by LOD-methods to increase the frame rate during latency-sensitive user interaction or to adapt the AVO to the limiting network properties.

An example for approach (2a) is given in Figure 2. It shows an adaptively refined volume mesh used for a flow calculation. The specific depiction unveils the different levels of detail determined by the visualization system according to the given network characteristics.

3.3 Occlusion-culling

Approach (3a) is a extension of approach (3), which integrates occlusion-culling methods [2, 11, 19] to reduce the number of polygons for transmission. The basic idea is to divide the polygon set into two pieces, one containing the visible polygons and one containing the remainder. Based on this distinction, visible polygons are immediately transferred to the visualization client and a first image can be generated much earlier. While the user investigates these visible polygons, the remaining occluded polygons are fetched from the server to enable subsequent viewpoint changes.

An advantage of this approach is that early visibility tests can be performed on the server, which is able to test millions of polygons in parallel [5, 7]. With sufficient memory available, deep object-space hierarchies can be exploited to minimize the run-time overhead of visibility related methods. Such hierarchies can be built specifically for the visualization cycle or derived by hierarchies already available, e. g. in case of numerical multigrid computations. In addition, abundant main memory allows novel occluder-selection strategies, like memory intensive image-based occluder selection [12], which allows run-time efficient occlusion-culling.

An example of approach (3a) is shown in Figure 3. The whole crankshaft model as shown on the top-right of Figure 3 consists of 66 223 tetrahedrons. After applying occlusion-culling to the scenario shown on the left, only 18 511 tetrahedrons remain, forming a conservative superset of the visible amount of tetrahedrons. Please note, that the mesh has been refined locally, meaning that there are severe variations in mesh density. The depiction shows only the surface polygons of the complete volumetric mesh.

3.4 Reference and image-based rendering

Approach (4a) extends approach (4) by image-based rendering techniques like depth-image warping [14], nailboards [16], or displacement mapping [17]. Such methods are useful for data transfer reduction, if the AVO does neither consist of densely occluded polygonal data, nor of polygonal data at all [13]. This is the case in volume rendering by means of ray-casting techniques, which would again require high communication bandwidth.

The solution to approach (4a) is to split the final stage

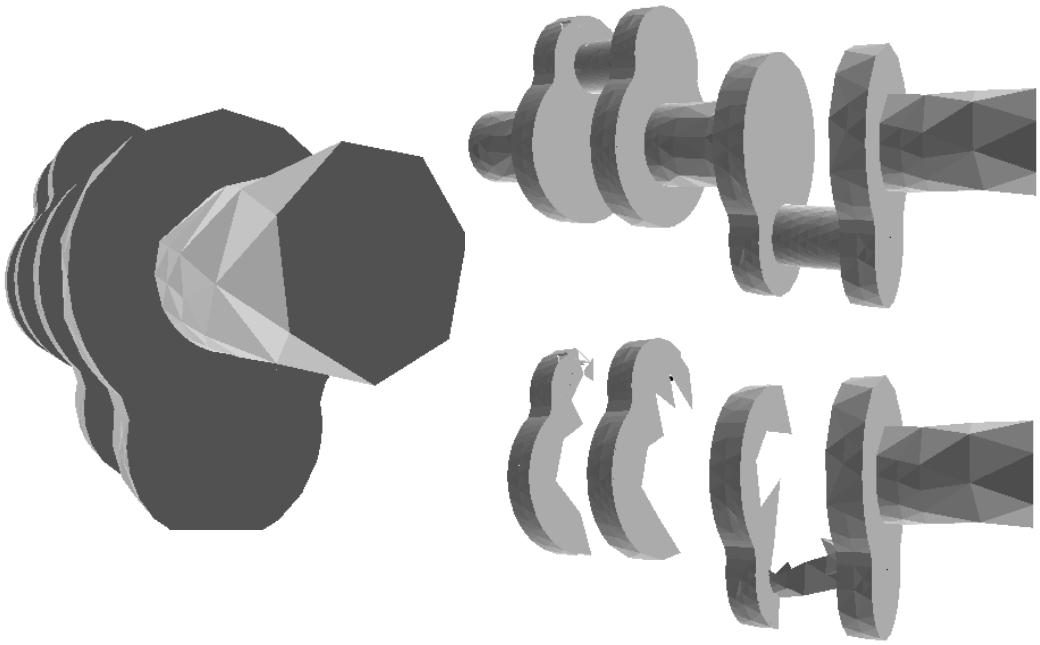

Figure 3. Occlusion culling technique

of the visualization process, the rendering stage, into two parts, reference rendering and image-based rendering. Reference rendering is performed on the simulation server and produces series of so called reference images. Depending on the actual image-based rendering technique, these reference images are images taken from different viewpoints around the object to be visualized or they are built by combining series of such images in order to create reference images with extended depth information like layered depth images [6]. By producing these images on the server instead of the remote visualization client, more images with better quality can be produced or combined in less time. The produced reference images are transferred to the visualization client, which generates the final depiction by means of image-based rendering techniques like post-rendering image-warping [12] or simpler hardware accelerated texture mapping techniques like multiple layers of alpha-blended textures common in the fields of volume visualization. In case of image-warping the high quality of the combined reference images created by exploiting the computational power of the server, only a reduced number of images for the warping-stage is necessary.

An example of approach (4a) is shown in Figure 4. The left part of Figure 4 shows four reference images of a cloud generated with different camera positions. These images are sent over the network to the visualization site, where they can be used as building blocks for 3D-textures or texture-mapped billboards. The final image, generated after merging the four input images using viewpoint-dependent alpha-blending, is shown in the right part of Figure 4. As demonstrated, this method is especially feasible for visualization of amorphous phenomena described by a three-dimensional density distribution like clouds, smoke, or haze [10]. Due the fuzzy nature of such phenomena, blending between different reference images does not introduce unpleasant visual artifacts or even popping during movements.

Another advantage of this approach is that the reference images can be generated and transferred to the client at a low rate, while the client produces the displayable output from these references at a high rate. This reduces the response time of the output devices, especially when changing local viewing parameters (because the new view can often be produced with image warping instead of transferring the images from the server to the client). In addition, this opportunity to generate images with slightly different viewing parameters from a single reference image can easily be exploited for VR stereoscopic viewing devices.

Please note, that in general an optimal solution is always context dependent. Thus, none of the described methods provide an optimal solution to every visualization problem. Consequently, the user has to select a pipeline configuration appropriate for the desired visualization task and the involved data structures, which is provided to GVK via the visualization description at the input interface (see Section 2.1).

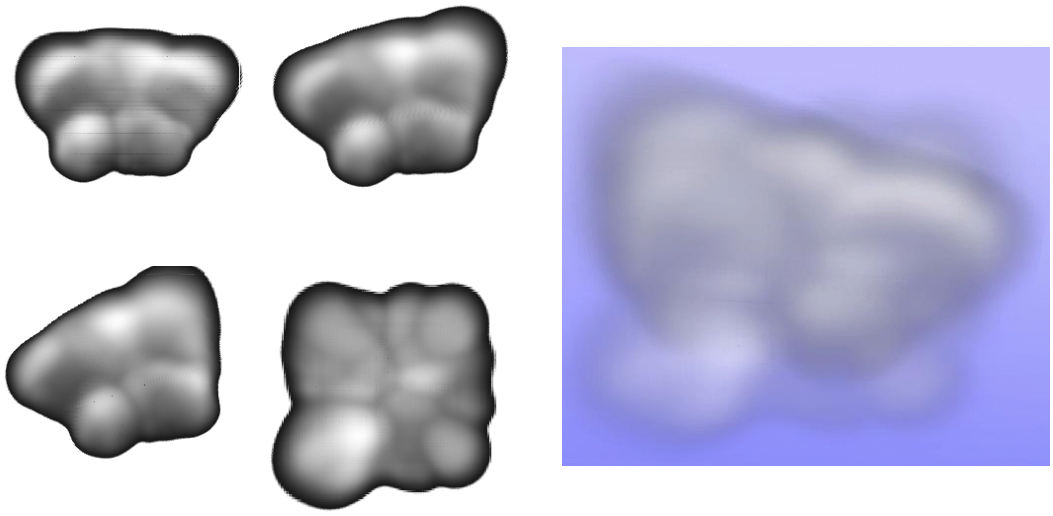

Figure 4. Four reference images generated on the grid and the final image displayed on the user's output device [10]

4. Conclusions and future work

The Grid Visualization Kernel described in this paper tries to address the needs of scientists and engineers for visualization of grid-enabled application data. Following the typical grid middleware approach, GVK provides an adaptable interface, which hides as much of its low-level functionality as possible. The key characteristics of GVK are its advanced configurations of the visualization pipeline. By using different kinds of optimizations corresponding to the given data structures, the amount of transferred data can be controlled. While the actual optimizations have traditionally been used close to the graphics hardware or in the hardware itself, the novel idea of GVK is to apply these technique at the optimal place between the simulation server and the visualization client.

The current results have been obtained with a first prototype of GVK, which has been assembled from the available optimization codes. This limits the tool to a subset of the possible data structures, that may serve as input for the visualization. In addition, the current approach relies heavily on the user's knowledge of the application's data structures and the potential benefits of selecting a corresponding compression technique. To overcome this drawback, it will be necessary to investigate adaptive optimization mechanisms, which measure the achieved transfer rate and adapt the pipeline configuration at runtime to the desired visualization technique, the available network bandwidth, and user specified requirements.

Acknowledgments This work was supported by several of our colleagues, most notably Jürgen Zauner and Bernhard Reitinger, who contributed to the prototype implementation of GVK.

References

[1] J. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10), 1976.

[2] S. Coorg and S. Teller. Real-time occlusion culling for models with large occluders. In *Proc. of the ACM Symposium on Interactive 3D Graphics*, pages 83–90, 1997.

[3] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):4–18, 1997.

[4] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid - enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[5] C. Georges. Obscuration Culling on Parallel Graphics Architectures. Technical Report 95-017, University of North Carolina at Chapel Hill, Department of Computer Science, 1995.

[6] S. Gortler, L. He, and M. Cohen. Rendering Layered Depth Images. Technical Report MSTR-TR-97-09, Microsoft Research, 1995.

[7] N. Greene. *Hierarchical Rendering of Complex Environments*. PhD thesis, University of California at Santa Cruz, 1995.

[8] A. Grimshaw, A. Ferrari, F. Knabe, and M. Humphrey. Legion: An operating system for wide-area computing. *IEEE Computer*, 32(5):29–37, May 1999.

[9] R. B. Haber and D. A. McNabb. Visualization Idioms: A Conceptual Model for Scientific Visualization Systems. In G. M. Nielson, B. Shriver, and L. J. Rosenblum, editors,

Visualization in Scientific Computing, pages 74–93. IEEE Computer Society Press, Los Alamitos, NM, USA, 1990.

[10] P. Heinzlreiter, G. Kurka, and J. Volkert. Real-time visualization of clouds. In *Proc. WSCG 2002, 10th Intl. Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, February 2002. [in print].

[11] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated occlusion culling using shadow. In *Proc. of the 13th ACM Symposium on Computational Geometry*, 1997.

[12] G. Kurka. *Bildbasierte Auswahl verdeckender Objekte (Image-based Occluder Selection)*. PhD thesis, Johannes Kepler University, Altenbergerstrasse 69, 4040 LINZ, Austria, Europe, September 2001. 205 pages, German.

[13] W. Mark. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. PhD thesis, University of North Carolina at Chapel Hill, Department of Computer Science, 1999. available as Tech.-Rep.: 99-022.

[14] L. McMillan and G. Bishop. Plenoptic modeling: an image-based rendering system. In *Proc. of ACM SIGGRAPH '95*, pages 39–46, 1995.

[15] M. Romberg. The unicore architecture: Seamless access to distributed resources. In *Proc. 8th IEEE Intl. Symposium on 'High Performance Distributed Computing' (HPDC-8)*, pages 287–293, Los Alamitos, CA, August 1999. IEEE Computer Society.

[16] G. Schaufler. Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes. In *Eurographics Workshop on Rendering 97*, pages 151–162, 1997.

[17] G. Schaufler and M. Priglinger. Efficient Displacement Mapping by Image Warping. In *Eurographics Workshop on Rendering*, 1999.

[18] G. Schaufler and W. Stürzlinger. Generating multiple levels of detail from polygonal geometry models. In Göbel, editor, *Proc. of the Eurographics Workshop on Virtual Environments '95*, pages 33–41, Monte Carlo, 1995. Springer Verlag.

[19] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proc. of ACM SIGGRAPH '97*, pages 77–88, 1997.