UNIVERSITE D'ORLEANS
*Faculté des Sciences*
**LIFO**

Laboratoire d'Informatique Fondamentale d'Orléans
4, rue Léonard de Vinci, BP 6759
F-45067 Orléans Cedex 2
FRANCE

Rapport de Recherche

www : http://www.univ-orleans.fr/SCIENCES/LIFO/

# Distributed Level of Details for Graphics Clusters

Rapport N$^o$ **RR-2004-14**

Valérie Gouranton[1*] Sébastien Limet[1†] Souley Madougou[1,2‡]
Emmanuel Melin[1§]

[1]LIFO
Laboratoire d'Informatique Fondamentale d'Orléans
Orléans, FRANCE

[2]BRGM
Bureau de Recherches Minières et Géologiques
Orléans, FRANCE

February 28, 2005

**Abstract**

Today, large and complex data sets are commonplace. With enhancements in electronic components, even low-end computers are able to process large part of them. But problems arise when one has to visualize such data sets, especially in real time environments. Level of detail (LOD) techniques try to answer these issues, but so far, there is no satisfactory solutions because most of the solutions are designed for shared memory machines. In this paper, we present a method that benefits from cluster computing power and storage capacities to speed up LOD algorithms and to scale data sets that are processed. The method begins by distributing data among some cluster nodes called computation nodes. There, LOD algorithm is executed using locally held data. Results are culled against view frusta, modified data only are determinated and sent for visualization to a subset of nodes called visualization nodes. Here, rendered data are updated using received data. Furthermore, the scheme removes the possible bottleneck on visualization node by permitting the use of a potentially high number of visualization nodes.

# 1 Introduction

To have a better understanding of high-end scientific problems we need to go through more and more modelizing and high performance computing. In these fields critical

[*]e-mail: Valerie.Gouranton@lifo.univ-orleans.fr
[†]e-mail: Sebastien.Limet@lifo.univ-orleans.fr
[‡]e-mail: Souley.Madougou@lifo.univ-orleans.fr
[§]e-mail: Emmanuel.Melin@lifo.univ-orleans.fr

points seem to be the examination and the analysis of the very diverse data, the integration of models, as well as the communication of the results with the aim to present and collaborate. One must be able to display high resolution pictures of bulky and complex data. To reach this goal, methods and means of visualization must then be proportioned with the means of parallel calculation and storage of data. Strategies of distribution of the treatments are then a key point of the effectiveness of the solutions.

On one hand, immersive environments [8] are classically powered by dedicated graphics supercomputers like SGI Onyx machines [23]. On the other hand, the anatomy of super-computing is quickly and deeply changing. Clusters of commodity components are becoming the leading choice architecture. Such cluster architectures are now usual in the supercomputer top 500 [30]. They are scalable and modular with a high performance-price ratio. These architectures are proved efficient for classical (not interactive) intensive computations. Although clusters are accepted as a judicious hardware to perform parallel computing, only recently they are also used to run interactive virtual reality applications in multi-projector immersive environments [2, 4, 19, 27]. This is due to libraries like Net Juggler, syzygy [28] or WireGL which provide automatic parallelization schemes for rendering computations on clusters. In particular, these approaches ensure proper levels of synchronization to guarantee data coherency. They harness the distributed resources (CPU, memory, GPU) such that the images displayed with the different projectors appear as a single high resolution image.

There are two main approaches. One parallelizes the application at the level of graphics primitives [19]. The application is executed on one or several nodes. Graphics primitives are intercepted and broadcasted to each node controlling a display. These nodes render the scene according to their local viewport. The amount of data to send over the network can be important, limiting the performance. The WireGL protocol integrates techniques to reduce the network bottleneck. But these techniques have limiting effect for real-time applications with frequent updates of the scene. A simpler approach [2] consists in duplicating the application on each node connected to one display. Every frame, events are broadcasted to each graphics node which enforce coherency. The required network bandwidth is limited, ensuring high performance even for real-time applications making frequent updates of the scene. In both approaches, the number of cluster nodes dedicated to graphics is limited by the size of the multidisplay environment. There is only one node controlling one display. In all the cases the pure graphics cluster has a limited number of nodes and is not as scalable as the pure computational cluster.

We show that clusters can be used as graphics and calculus supercomputers, further works demonstrate that it is possible to use PC cluster to compute a parallel simulation and at the same time to render the result on a multi display screen [3, 13]. Theses approaches are based on an ad-hoc coupling between the simulation and the visualization. In [13] a fluid motion simulation is coupled with a parallel VR platform based on the dual VRJuggler and Netjuggler. The pure simulation is based on a multigrid algorithm. It is a general concept used by physicians to solve a differential equation starting from the discretization in a linear system [15]. Its work can be considered as a representative scientific modeling. Furthermore, this kind of applications requires realistic graphical rendering. This simulation generates a large amount of computations and data justifying the use of a cluster to be parallelized. A key point is then to reduce the communication volume from the simulation nodes to the rendering nodes. In [13] a specific way is proposed to decrease communications by a factor of four. If we consider that a visualization cluster has a limited number of nodes, this work illustrates the pressing necessity to take into account possible bottleneck or overloading of the network. These phenomena are greatly accentuated by the dissymmetry between the sizes of the simulation cluster and the visualization one. Finally the parallelism control has to be entirely managed by an advanced programmer without any general method proposed to build such application.

A very similar lock was encountered by the French Geological Survey (BRGM), namely the rendering and the navigation in geological models displayed on a virtual reality environment driven by a cluster of PC. In Geosciences, using VR allows a better comprehension of the composition of the basements and underground phenomena thanks to 3D rendering. The exploration of such a model has two aspects: one very interactive for searching a specific location (for example a typical geological configuration), a second more static where it is important to render a maximum of details of a part of the model to study the specific point selected during the first phase. The main difficulty is to reconcile a high rate interactive rendering on the graphical cluster with a very large data set (geological data base) potentially too big to be simplified locally by graphical nodes. This requires using level of detail (LOD) techniques. They are classically used either for removing redundant geometry or for speeding up rendering performances for interaction. Another use of LOD algorithms is for network transmission by allowing the reduction of the size of the model and consequently reducing the amount of communication. From this point of view, LOD techniques are the key point to link the computation nodes with the visualization ones. The solution is to create a LOD algorithm not to be taken by visualization node and reducing network load even at the price of an overload of calculus. Moreover, it is possible to minimize this over cost by using a parallel version of the LOD algorithm to take advantage of the scalability of pure computation cluster. There are very few parallel LOD algorithms and as far as we know none of them allows a scalability on a cluster of PC with interactive frame rate. In this paper [14] we propose a parallel LOD algorithm dedicated to geological models described by height fields. This approach gives encouraging results but performances suffer from an useless network overload due to transmission redundant information. Moreover it might be useful to extract from this work a general approach for coupling parallel application with parallel visualization framework.

We propose a general framework to accelerate the interactive analysis of scientific data by intensive calculations, in particular for the very large volumes of data met for example in the geological sciences or physics simulation. Our work intends to answer a general question: how to use as best as possible the scalability of a VR Cluster in term of storage capacity and graphical computing power for rendering large models ?

To reach this goal the power of the cluster is used to fill the gap between the voluminous set of data generated by simulation computed by potentially indefinitely scalable cluster and the visualization means necessarily constrained by the size of the immersive environment and its dedicated graphics cluster. We show that parallel simplification or level of detail algorithms (LOD) are good candidates to obtain a graphical rendering performed in real-time which is essential to achieve a realistic interaction with the virtual world. Moreover these principles are usable in different simulation and visualization contexts. We illustrate this with two different original scientific visualization applications in a VR multidisplay environment. The remainder of the paper is structured as follows: in section 2 we give an overview of our VR framework. A fluid simulation parallelization description follows in section 3 with analyzed benchmarks. We gives a parallel algorithm managing a large geological set of data in section 4. As for previous section, benchmarks are proposed at the end. We conclude in section 5.

# 2 General Framework

## 2.1 VR platform from Netjuggler and Softgenlock

The fast development of VR applications is a tricky task due to the complexity of the hardware like multiple displays, the human interface devices or the parallel computer with multi-graphics pipes. We developed a VR framework allowing us to develop easily applications computed on a PC cluster. This work is built on the open source library VRJuggler [5]. VRJuggler offers an abstraction of a VR platform giving to programmers the possibility to create and test VR application via proxies totally independently of the hardware. This library is composed of a kernel which drives the application, some managers adapted to each part of the system and a set of configuration files which ease the management of all the components such as the displays and the input or output devices. These configuration files describe devices whose data are processed and restored by the input manager to the VRJuggler application. This framework integrates graphics API as OpenGL or performer and is exploited for CAVE or Workbench platforms driven by supercomputers.

We proposed to replace the graphics supercomputers by PC clusters. They are composed of PC equipped with off-the-shelf graphics boards and interconnected by an efficient network. We solved two major problems: the management of the devices distributed on the different nodes of the cluster and the coherency of pictures displayed on several displays due to the distribution of the visualization computing on the cluster. Our response to the first point has been provided by Netjuggler [2]. It extends VRJuggler on clusters to control the distribution of the displays or the devices on different nodes of the cluster. Netjuggler is totally transparent to VRJuggler user. VRJuggler sessions are replicated on all nodes of the cluster. Netjuggler broadcasts data held by nodes supporting physically the devices. For each device Netjuggler replaces the VRJuggler proxy by the management of a client server communication via the network. In order to define the server and the clients, we need to add to the VRJuggler configuration files the node name of the host for each device. From this information and the definition of the cluster, Netjuggler is able to manage the cluster as well as a single graphics supercomputer.

To manage coherency of pictures displayed on multi-screen platform we guarantee three level of synchronization: datalock, swaplock and genlock. NetJuggler replicates the same VRJuggler session on all the nodes of the cluster, hence, it naturally enforces data coherency, and so, datalock is guarantee. The nodes also execute a synchronization barrier just before swapping their frames buffers to ensure the consistency of the images displayed, therefore swaplock is guarantee.

VR environments often integrate 3D active stereo glasses. The user wears these glasses with shutters opening alternatively to let him see the right eye images with the right eye only and vice-versa. Left and right eye images are displayed alternatively by video-projectors. In this case a video frame level genlock is mandatory. If not the user may see from the same eye both, a right and a left eye image displayed by two different projectors. We developed in collaboration with ID and LESI laboratories a software called Softgenlock [1] for Linux platform. Independently of the kind of graphics cards, on each graphic node, this software computes a precise time delay between its video signal and the master one. Then the graphical node can locally decide to control and adapt its signal generation speed. The result obtained is not a fine video signals synchronization at a pixel level but a sufficient video frame level synchronization.

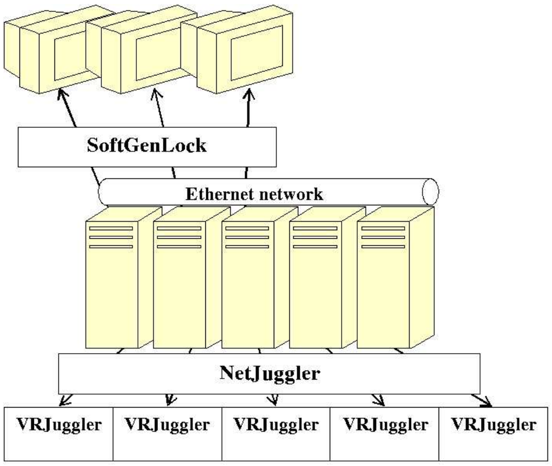


Figure 1: Graphic cluster

The gathering of all these tools, as described in Fig.1, provides a high performance VR environment for PC cluster. It had been ported with success on several VR platforms like Workbench by the BRGM research center [22] or display wall [3] by ID-IMAG laboratory.

## 2.2 Parallel LOD

As we show in the introduction, level of detail (LOD) is a key point to make possible the communication of a pure computational cluster and a pure graphics cluster. The computational cluster perform high cost parallel calculus in terms of time and/or in terms of memory space. The graphics cluster has to control displays each connected to one unique node. The amount of graphical primitives managed by each graphical node must be adapted to the desirable frame rate depending on the aimed interaction level. When real-time is required, high resources consuming applications like physics simulation or large data sets visualization requires using level of detail (LOD) for speeding up rendering performances and to reduce the size of data transmitted by the network.

LOD algorithms differ by the visual quality obtained after simplification, the preprocessing time and runtime performances. Any of these algorithms proceed either by sampling, adaptive subdivision, decimation or vertex clustering. In sampling, the initial model is sampled using the most significant vertices. Adaptive subdivision requires a base model that is refined recursively. Algorithms that use decimation iteratively remove vertices or faces and retriangulate the generated holes. Vertex clustering recursively gathers several vertices in one vertex that can be gathered in turn.

There are several kinds of LOD algorithms: static, dynamic and view-dependent. Static LOD algorithms create several discrete versions of each object in a preprocess, each at different level of detail. At runtime, the algorithm chooses the appropriate version for the object [25, 9]. Whereas static algorithms create individual LODs during the preprocessing stage, a dynamic algorithm creates a data structure, most often a tree, encoding a continuous spectrum of detail [16]. The desired LOD can be extracted from this structure at runtime. View-dependent algorithms extend dynamic algorithms by using view parameters to choose the desired level [20, 11, 17]. The choice of a specific algorithm depends on the kind of data processed. For instance, complex scenes composed of thousands of small objects can use static algorithms whereas one-piece large scenes like terrains require view-dependent algorithms.

Although they are often complex and deal with large data sets, most of view-dependent algorithms are sequential. They are consequently limited by computer resources like memory and CPU-time. Some of them include out-of-core scheme [21] but disk accesses still remain too long to satisfy real time requirements of Virtual Reality applications. Our aim is to obtain a scalable solution of rendering large data sets. For that we take into account that an alternative solution to high-end dedicated computers emerged for a few years: graphics clusters. An obvious advantage of clusters is that there is theoretically no limit to the number of nodes it can contain. Consequently, using parallel algorithms on this kind of architecture may be very interesting provided that the algorithm itself is scalable. Scalability can be obtained by carefully distributing the data on the cluster and avoiding as far as possible data gathering to keep off the two main bottlenecks due to communication network and memory storage.

There are very few parallel simplification algorithms.

El-Sana and Varshney [12] proposed one of the first works on parallelizing view-dependent LOD algorithms. They worked on Xia's merge trees [31] . Merge tree is a data structure built upon Hoppe's progressive meshes [18]. The input polygonal mesh is divided into independent subsets that are processed in parallel. The parallelization is only used in the merge tree construction not during the rendering. Furthermore, the implementation is intended for shared memory machines only which does not fit our purposes.

In [10], Dehne and al. describe a scheme for parallelizing the progressive meshes (PM). They started by partitioning the original mesh using a greedy graph partitioning. Each partition is then sent to a processor that converts it to the PM format. The resulting PM are then merged to produce the final PM for the original mesh. The PM algorithm they worked on is not view-dependent which is a drawback when LOD is used to navigate in the model as we intend to. Moreover it requires a lot of memory for the complex data structures used by PM.

PR-Simp presented by Brodsky and Pedersen [6] is a parallel extension to their R-Simp [7], a sequential model simplification algorithm. PR-Simp uses master/slave architecture. Master starts by computing a bounding box of the entire model that it sends to the slaves. When they receive the bounding box, they divide it in $n$ clusters, where $n$ is the number of processors. Each processor then scans all vertices and stores those that fall in its cluster. R-Simp is used to simplify the clusters on each processor. After that, a divide and conquer approach that takes $logn$ iterations is used to merge remaining parts on master. This is costly and places a bottleneck on the master. PR-Simp permits a data scaling but it is not view-dependent.

This paper is a contribution for the use of parallel approaches to view-dependent LOD algorithms for filling the gap between large computation and display means. We choose to present a general framework and then illustrate its benefits by realistic applications in 3 and 4. To achieve our goal, the cluster nodes are partitioned into two classes: visualization nodes and computation nodes. Visualization nodes are responsible of rendering all or a part of the scene. They perform no simplification. Computation nodes carry out an application generating a large amount of time computing and/or managing a large volume of data. Moreover, computation nodes compute the simplification algorithm on a portion of data generated by the entire computation cluster. Simplified data are culled taking into account the view frustum of each visualization nodes. Then a additional compression technique can be applied to reduce the data traffic on the network. The scheme of the general framework is described in fig. 2.

Basically, the algorithm is composed of three stages:

- **Initialization**: during this phase which takes place only once, the data and the work are distributed to the computation nodes. This allows to benefit from compute power and memory storage capacities of each computation node
- **Communication**: during this phase, visualization nodes broadcast the current view point and their frustum to computation nodes and each computation node sends to each rendering node the part of scene it has to render and that falls in its frustum.
- **Computation**: during this phase, computation nodes run LOD algorithm on the part of the data they are in charge while visualization nodes display the part of the scene they have received.

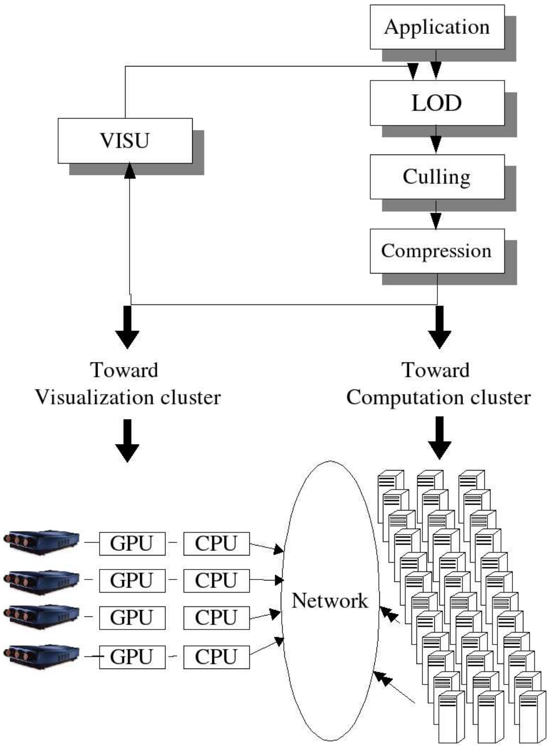


Figure 2: The framework

One can remark that computation nodes perform a special data culling and a compression for each visualization node which allows to optimize data transfer and lighten graphical boards by discarding non-visible part of the scene. Here are some more details about the parallelization framework: the data partitioning and how to minimize the communication costs are described is section 2.3.

## 2.3 Data partitioning and communication costs

Data partitioning takes place at the initialization stage. Data are received from simulation process or read from a file available on all computation nodes. According to its rank, each node knows exactly which portion of the data it has to store and compute. Each computation node gets a portion but we allow overlap if necessary. Our application needs only local data to be computed. Therefore only overlapped data need to be communicated to distant nodes see fig.3.

As usual in parallel computing, communications are the key point to obtain efficient algorithms, especially when dealing with large datasets. In our case, sending all data each frame should clearly be discarded. One way to bypass this issue is to reduce the

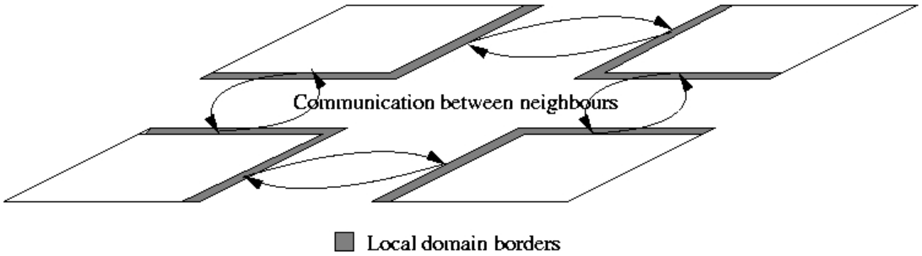


Figure 3: Data sent per node in case of 4 computation nodes. Overlap regions are sent to neighbors

amount of data that transit by the network. We use the structure of data, culling and compression to do this. In the following sections, we explain how these ideas are used to reduce the communication time.

### 2.3.1 Sending only indices

We turn the regular structure of our data to our's advantage to do certain optimizations. Scientific data issued from modelization or scanning are often structured through regularly sized grid in at least one plane. By a very simple coordinates system changing, this plane can be the reference for $x$ and $y$-directions. So, we just need the indices $(i, j)$ of a vertex, the extent of the grid , the spacings along the $x$ and $y$-directions and the corners to deduce the $x$ and $y$ coordinates in case of 3D graphics. This scheme allows to reduce communication costs by 33% because 2 floats instead of 3 are sent for each point to be drawn.

### 2.3.2 Cull data before communication phase

After using the structure of grid to reduce the network load, the communication time still remains too high for real-time rendering. To avoid this, the subscenes are culled before sending them to visualization nodes. At the beginning of each local simplification stage, computation nodes receive the view point and a frustum from each visualization node. The view point is used to carry out LOD simplication algorithm while the frusta are used to cull the output of the simplification step. We do not test each vertex against frusta because it is too costly. As we need to send each graphics primitive that intersects the frustum even partially, we have to test against it's bounding volume (box or sphere). With this processing, only a relatively small part of computation nodes' portion needs to be sent. When the viewer is close, the LOD is high but the area in the view frustum is small (fig.4). When he is far enough to see the entire scene (fig.5), the LOD drops so that the amount of data to be sent is always of the same order. This allows to maintain an interactive frame rate.

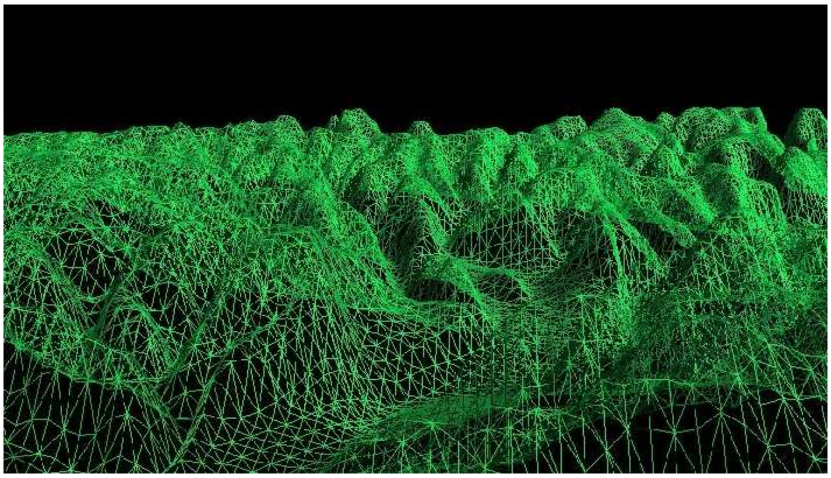

Figure 4: LOD near the view point

### 2.3.3 Data compression

Despite of LOD which dynamically generates representation of data, we observe that, between two frames, a great part of the rendered data does not change even with interaction. On the basis of this observation, we installed a mechanism allowing to detect the modified data. It should be noted that this stage takes place after culling, therefore, it applies on data already considerably reduced and it does not induce a significant calculus overload compared to the awaited benefits. Depending on the type of data which is handled by the application, the compression method can differ. For example, when sending sparce matrices, compression should consist in eliminating 0 values. For other application where the scene is slightly modified between two frames, a mechanism very similar to Unix commands **diff** and **patch** can be very efficient.

## 2.4 Conlusion

Our parallelization scheme (fig.2) using LOD, culling and compression algorithms is scalable since the entire sceen is distributed among computation nodes only once, at the begining, and communication are limited afterwards. We also strongly limits the communication overhead from computation nodes to visualization nodes thanks to fustrum culling and compression. It should be notice that no process in our framework has a special task such as mastering the others for gathering data, which increases its scalabity capabilities. Moreover we can take into advantage of scalability of the pure computation cluster since we can distribute LOD, culling and compression on different separate computation nodes. In this case the use of dedicated sub-network within the computation cluster may avoid overload of a single network for all trafic.

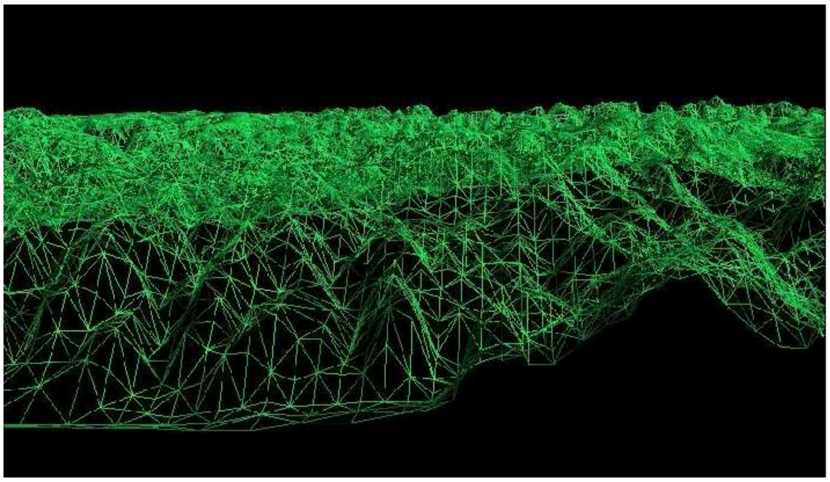

Figure 5: LOD at farther regions to the view point

# 3 Application to Fluid Motion Simulation

The first scientific application we use to illustrate our approach is a fluid motion simulation.

## 3.1 Fluid motion in a VR application

In [13], we propose an implementation of an interactive simulation of fluid motion with the aim to getting close to physical phenomena. The fluid motion is defined in term of Navier-Stokes equations whose solver is built by using the multigrid algorithm [15].

A finite representation of the fluid and its motion is given by associating two fields, velocity and density, to each point of a grid of interest. These two fields are obtained at each time step from the resolution of differential equations resulting from a decomposition of the Navier-Stokes equations .

The multigrid algorithm consits in a general method to solve a differential equation starting from the discretization in a linear system. A finite reprensentation of the fluid is defined by a grid of size $n \times n$. The linear system is of the form $Ax = b$ where $A$ is a $n^2 \times n^2$ float matrix. Note that the matrix is usually very sparse. Let $M$ be an elementary iterative method to solve the linear system from an initial solution such as the Gauss-Seidel relaxation. The multigrid algorithm is the following:

1. Use $M$ to approximate a first solution $x_1$ for $x$,
2. Estimate the error and solve an intermediary linear system,
3. Use $M$ to compute a new solution $x_2$ for $x$,

which can also be iterative.

The stage 2 constitutes the originality of the multigrid. An equation of the form $Ae = b - Ax_1$ is solved where $x_1$ is the first approximated solution found in stage 1 and $e$ denotes the error factor between the approximated solution and the exact one.

As the process is iterative, the multigrid method allows to safely decrease the step 2 complexity. Two operators, a restriction operator and a prolongation operator, relate the $n \times n$ grid called the fine grid to a coarse one of size $n/2 \times n/2$. The multigrid algorithm that is then 16 times less complex, becomes the following:

1. Find $x_1$ such that $Ax_1 = b$ from an initial vector (random one from for the first iteration)
2. (a) Apply the restriction operator to $x_1$, $A$ and $b$ resulting respectively to $x'$, $A'$ and $b'$
   (b) Find $e'$ such that $A'e' = b' - A'x'$
   (c) Apply the prolongation operator to $e'$ resulting to $e$
3. Find $x_2$ such as $Ax_2 = b$ from $x' + e$ as initial vector for this new iteration step

The algorithm can be recursive in solving the step 2 by a new two levels multigrid method. It can be useful in order to decrease the global complexity when the initial grid is strongly fine. Where the Stam's [29] Gauss-Seidel solver needs 20 iterations to converge, the multigrid algorithm reaches an equivalent level of accuracy with 5, 7 and 8 iterations respectively for the steps 1, 2 and 3, always with a Gauss-Seidel as elementary solver $M$. A gain of 20% is then obtained in term of complexity thanks to the use of the coarse grid in the step 2.

In [13], we propose an advanced approach to design the fluid motion (Fig. 6) with more parallelism control to experiment the optimizations that a strong link between the visualization and the simulation can offer.

The fluid motion in a VR application is a representative example of requirements in terms of efficient simulation and interactive visualization. In order to reach a real-time simulation the solver is parallelized by decomposing the fluid domain on the different cluster nodes. The most significant part of this parallelization is the multigrid algorithm because it could generate a great number of communications.

To avoid this disadvantage, the Gauss-Seidel is used as elementary $M$ method and modify the linear system resolution in order to restrict the communications to borders exchanges as described in Fig. 3.

The multigrid algorithm becomes the following:

1. Perform Gauss-Seidel relaxation on the local fine grid
2. Exchange borders with neighbour nodes
3. Restrict the error to the local coarse grid
4. Perform Gauss-Seidel relaxation on this coarse grid

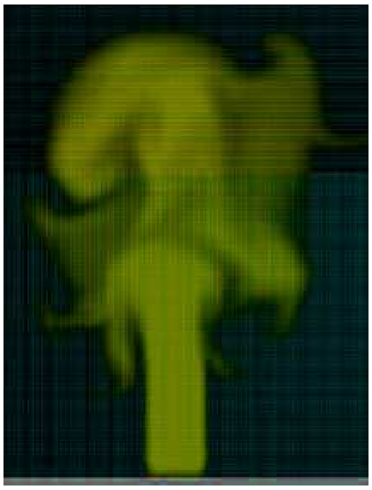

Figure 6: Fluid motion

5. Execute the prolongation of the result
6. Perform Gauss-Seidel relaxation on the fine grid
7. Exchange borders for the next step.

The parallelization of the sequential multigrid algorithm would have consisted of exchanging the borders between each iteration of the steps 1, 4 and 6. But this version delivers analogue simulation accuracy and convergence speed while ensuring scalability and low cost communications.

In case of high cost simulations, more computation than rendering nodes are needed to ensure the real-time constraint of a virtual reality application. Then the NetJuggler choice to replicate the rendering computation on all the nodes may become a handicap to performances. Processes which are doing graphical computations can be distinguished from those which are computing simulation steps thanks to some tricks with MPI rank inside NetJuggler. This method allows to improve several tens % in frames rate.

Depending on the number of computation nodes, it is possible to run the simulation on an increasingly fine grid which could become a problem for visualization nodes which are limited. In [13], it is proposed to differenciate the computed and the displayed grid. The second one being fixed to maintain an interactive frame rate. A new operation is defined which approximates the fluid solver results with a pertinent average of the density on a coarser rendering grid. This operation looks like a level of details mechanism but it is not view dependant. The main disavantage of this method is that is impossible to focus on some point of the scene to observe local fine details even if those details are available from the simulation.

In the implementation we propose here, we use the same multigrid solver as in [13]. We have included a view dependant LOD mecanism. The LOD procedure selects points of the grid which have to be effectively displayed according to the distance of the view point. The multigrid method distributes the simulation grid over the computation nodes. After each simulation step, the computation nodes perform the LOD mecanism on their local result. The LOD procedure produces two vectors: one containing the indices and the second the density of the displayed points. The frustum culling procedure runs on those points as described on section 2.3.2. The data compression simply consists in compacting 0 values.

## 3.2 Results

We perform our implementation on a cluster of 8 PC dual xeon equipped with NVIDIA Geforce FX 5900 128 Mo graphics boards. The interconnecting network is a gigabit ethernet using TCP/IP. We use Net Juggler [2] as cluster middleware. This facilitates inter-frame synchronization and communications as it permits direct calling of MPI [24] routines. We use OpenGL as graphics library as immediate mode fits particularly to view-dependent LOD.

The simulation grid is a square of $324 \times 324$ of fluid cells. We used 12 processors to run the simulation. Note that a LOD procedure is associated to each computing node. The fluid simulation consists in adding a fixed amount of velocity and density at each step. To obtain the same simplifications in the model, the navigation is pre-programmed. We have run this test on one or four display nodes under the following conditions: without any LOD mecanism, then activating LOD and finally adding the frustum culling and compression.

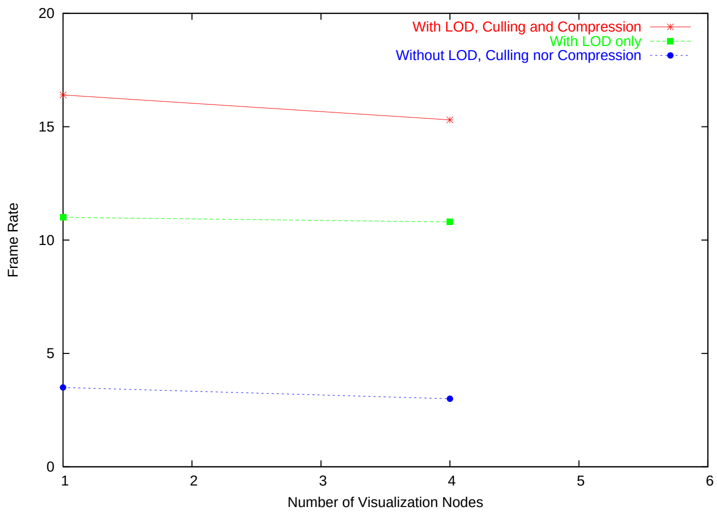


Figure 7: Average frame rate for 1 or 4 visualization nodes

The results (Fig. 7) show that without any LOD mecanism, the frame rate for a grid which is relatively small is not acceptable for interactivity. Adding LOD mechanism allows to multiply by three the frame rate but it still stays at a low level. The culling mecanism allows to increase the frame rate of 50%, reaching values arround 16. This becomes high enougth to permit interaction. Considering the results from one to four visualization nodes, we expected to obtain a better frame rate for four visualization nodes when the culling is activated since the graphics computation is distributed on the different GPU. In fact, we obtain equivalent frame rates. This is due to the fact that first the rending process is very simple and the amount of displayed points is reduced thanks to the culling (when the model is very closed to the view point) and thanks to the LOD (when the model is far). So the work of the GPU is reduced and the frame rate is then limited by the simulation. Recall that in NetJuggler, a synchronization barrier between processes, is performed at each frame. So the gain obtained by the distribution of the graphical computation is lost by the computation overhead for handling the four displays nodes. It should be notice that the graphical resolution with for nodes is 4 times better than with 1.

In the following section, we show that the distribution of the graphical computation can be usefull when dealing with large amount of data.

# 4 Application to the Visualization of Height fields

Height fields are good candidates for application of our framework. They have a regular structure and they are often very bulky. Our first work was to choose an algorithm that fits to height fields. We found that Rottger's algorithm, described in [26], corresponds perfectly to our requirements: fitting to height fields, view-dependent, efficient use of memory. Details can be found in [14].

## 4.1 Rottger's algorithm and parallelization

This algorithm is based of the work of Lindstrom who described a view-dependent algorithm for height fields of size $(2^n+1)*(2^n+1)$ in [20]. The algorithm dynamically modifies a quadtree by using a top-dow strategy to determine whether a node needs to be subdivided or merged with adjacent nodes. The subdivision or the merging is performed based on distance to view point and local surface roughness. For that a boolean criterion is evaluated. The top-down strategy allows to visit just a fraction of the data set: only one point per block. This fraction depends on the rendering quality. The algorithm uses a boolean matrix, so called quadtree matrix, to capture the state of the quadtree at each step of the triangulation. In this matrix, each node's center is set, if the node is further refined. Otherwise, a special value is used. After the triangulation is finished, Rottger algorithm draws triangle fans that allow to reduce the amount of data sent to graphics pipeline and better capture the roughness of a surface.

The triangulation is created by recursively descending the quadtree. At each node, a boolean subdivision criterion is evaluated and its result stored in the quadtree matrix. If it is evaluated to true and the finest LOD has not yet been reached, the recursion continues by visiting all four sub-nodes. The subdivision criterion depends on the distance to view point as well as local surface roughness. It allows to reduce the resolution as the distance to view point increases and to augment it in regions of high surface roughness.

When the finest LOD is reached, the height field is drawn by recursively traversing the quadtree where the corresponding matrix entries are set. Whenever a quadtree leaf is reached, a complete or partial fan is drawn. To avoid cracks between adjacent edges of nodes at different resolutions, the center vertex at these edges is skipped. This method works only when LOD of adjacent sub-nodes differ by no more than one which is guaranteed by the way the algorithm computes and stores surface roughnesses.

Rottger's algorithm is an in-core one. So, it can not processes height fields that not fit in system memory. Its, then, convenient to use parallel machines to bypass this drawback. In order to apply our framework, we first distribute height field data among computation nodes by allowing overlap. At each frame, each computation node receives the view point and the frustum of each visualization node. Rottger's algorithm then computes the current list of fans. This list is culled against frusta. For each pair visualization node and computation node, a sublist of visible fans is determinated. In addition to preceding work [14], we introduce a compression mechanism. Every computation node stores a backup of all its lists, one for each visualization node and corresponding to the previous frame. The backup and the new sublist are then compared. Modified fans are identified and sent to visualization node. The visualization node takes its corresponding local backup and patches it using incoming data. We experimented that these data never exceed 20% of data normally sent.

## 4.2 Results

The tests have been performed on the same software and hardware plateform as in section 3.2. The data we use is provided by the French Geological Survey (BRGM) and represents a height field of $760 \times 760$ points. This represents about 1,155,000 triangles. We use this as one layer and then create other layers for test purposes. The resolution we use is of $1280 \times 1024$ per visualization node. So the resolution for four visualization nodes, for example, is $2560 \times 2048$. All tests are performed in the same conditions such as model initial position, level of detail parameters and interaction.

First, we choose to compare different versions of our parallel algorithm according to the frame rate . These versions use 1, 4 and 9 computation nodes and only one visualization node. As fig.8 shows, there is a clear improvement in frame rate while passing from 1 to 4 and then to 9 computation nodes. This illustrates the interest of cluster architecture for distributing the LOD and the pure rendering work on different nodes. The softening of the slope of the curves is due the fact that as data become more and more larger by adding layers, the network becomes more encumbered and a bottleneck appears on the visualization node. This can be avoided by using more than one visualization node or by implementing a triangle budget mechanism [11]. Our framework includes a triangle budget but we do not activate it because this would mask the real contributions of implemented optimizations.

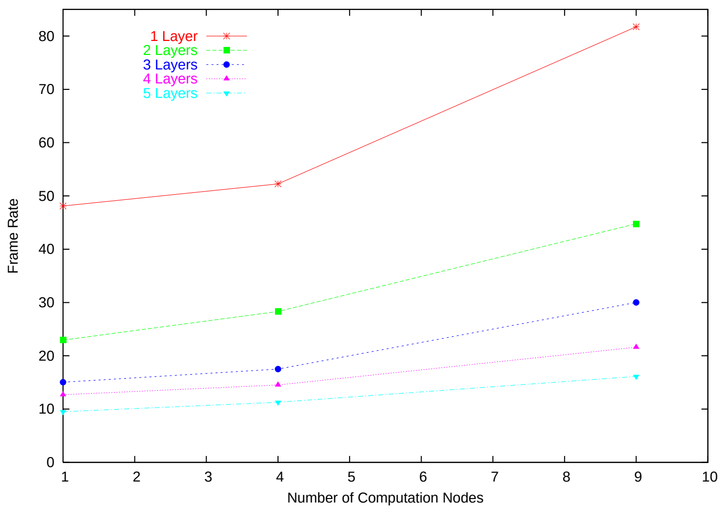


Figure 8: Average frame rates for 1 , 4 and 9 computation nodes (using 1 visualization node)

As described above (section 2), each computation node can send different graphics data to each visualization node according to their current frustum. This partially erases the bottleneck generated by one single visualization node. The figure fig.9 shows the gain in frame rate obtained by passing from 1 visualization node to 4. This shows an almost constant gain for any amount of data. This illustrates the interest of the Netjuggler parallel rendering.

To evaluate the contributions of the different optimizations, we perform tests by deactivating these mechanisms. Results are summarized in the figure fig.10. As one can see, performances drop dramatically. The figure shows that without culling, transmitting graphics data on network for real-time rendering is almost unfeasible and that the compression mechanism by reducing communications overhead contributes to improve the frame rate..

# 5 Conclusion and future work

We have presented a general method to perform a real time graphics rendering of data issued of cluster computing. This approach is based on the use of view-dependent level of detail algorithms, frustrum culling and raw compression. These combined methods make possible the communication between a pure visualization cluster and a very scalable hight performance cluster computing large amount of data. Network overload can be limited and the programmer can tune LOD parameters to adapt the visualization quality to hardware capability. Moreover this approach can turn into benefit the use of

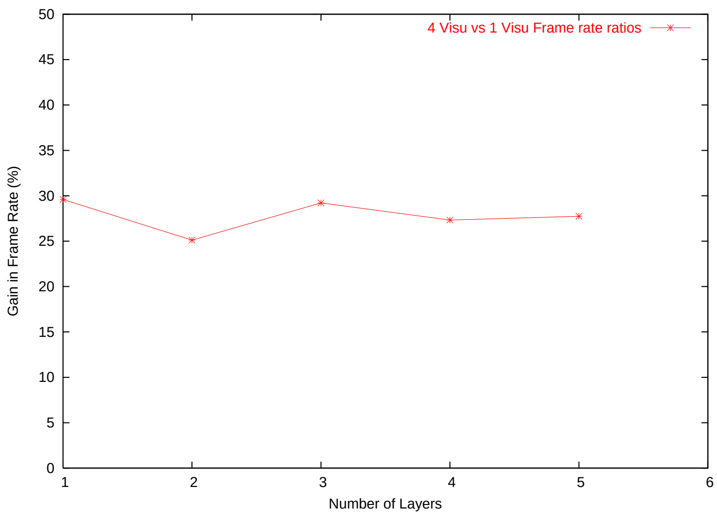

Figure 9: Frame rate gain by passing from 1 to 4 visualization nodes (using 4 computation nodes)

multi screen display. In this case, we avoid the network overload generated by communications to PCs controlling screens. When the graphic complexity raises we observe a constant gain obtained by distribution of the graphics computation.

With framework suffers from a real drawback: in code, pure computation and pure visualization are separated via a trick in Netjuggler and one's has to be an advanced programmer to control the parallelism and the visualization. Another main limitation is the difficulty to assemble and distribute the different (potentially parallel) components and to maintain the overall application coherent while guaranteeing a good quality interaction with low latency and high refresh rates. We need an extension of Netjuggler in order to provide a development environment which can ease the implementation of such sophisticated simulations. This new platform would compose parallel applications in such a way that it is possible to express a fine control parallelism and, at the same time, to reuse codes for example issued of classical physics simulations and LOD algorithms.

# References


[1] J. Allard, V. Gouranton, G.Lamarque, E. Melin, , and B. Raffin. Softgenlock: Active stereo and genlock for pc cluster. *IPT/EGVE'03 Workshop*, 2003.

[2] J. Allard, V. Gouranton, L. Lecointre, E. Melin, and B. Raffin. Net Juggler: Running VR Juggler with Multiple Displays on a Commodity Component Cluster. In *IEEE VR*, pages 275–276, Orlando, USA, March 2002.

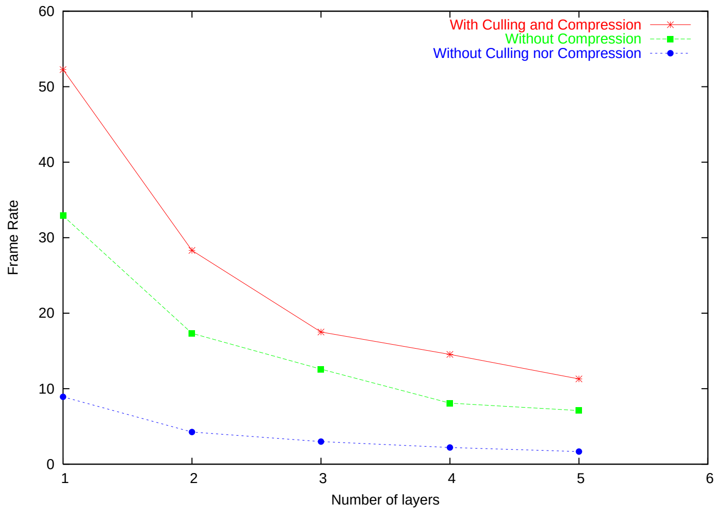


Figure 10: Average frame rates for 1 visualization node and 4 computation node with and without communication optimizations.


[3] J. Allard, B. Raffin, and F. Zara. Coupling parallel simulation and multi-display visualization on a pc cluster. In *Euro-par 2003*, Klagenfurt, Austria, August 2003.

[4] P. Augerat, C. Goudeseune, H. Kaczmarski, B. Raffin, B. Schaeffer, L. Soares, and M. K. Zuffo. Commodity clusters for immersive projection environments. Siggraph 2002 Course, July 2002.

[5] A. Bierhaum. Vr juggler: A virtual platform for virtual reality applications development. *Master's thesis*, 2000.

[6] D. Brodsky and J. B. Pedersen. Parallel model simplification of very large polygonal meshes. *PDPTA'02*, pages 1207–1215, 2002.

[7] D. Brodsky and B. Watson. Model simplification through refinement. *Graphics Interface'00*, 2000.

[8] C. Cruz-Neira, D. J. Sandin, T. A. DeFanti, R. V. Kenyon, and J. C. Hart. The Cave Audio VIsual Experience Automatic Virtual Environement. *Communication of the ACM*, 35(6):64–72, 1992.

[9] L. DeFloriani, P. Marzano, and E. Puppo. Multiresolution models for topographic surface description. *The visual computer*, 12(7):317–345, 1996.

[10] F. Dehne, C. Langis, and G. Roth. Mesh simplification in parallel. *ICA3PP'00*, pages 281–290, 2000.

[11] M. Duchaineau, M. Wolinsky, D. Sigeti, M. Mille, C Aldrich, and M. B. Mineev-Weinstein. Roaming terrain: Real-time optimally adapting meshes. *IEEE Visualization*, pages 81–88, 1997.

[12] J. El-Sana and A. Varshney. Parallel processing for view-dependent polygonal virtual environments. *Proceedings SPIE'99*, 1999.

[13] R. Gaugne, S. Jubertie, and S. Robert. Distributed multigrid algorithms for interactive scientific simulations on clusters. *ICAT 2003, Japan*, 2003.

[14] V. Gouranton, S. Limet, S. Madougou, and E. Melin. A scalable cluster-based parallel simplification framework for height fields. *Eurographics Symposium on Parallel Graphics and Visualization, EGPGV'04*, 2004.

[15] W. Hackbusch. Multigrid methods and applications. *Springer, Berlin*, 1985.

[16] H. Hoppe. Progressive meshes. *In proceedings SIGGRAPH'96*, pages 99–108, 1996.

[17] H. Hoppe. View-dependent refinement of progressive meshes. *In proceedings SIGGRAPH'97*, pages 189–198, 1997.

[18] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. *In Proceedings SIGGRAPH'93*, pages 19–26, 1993.

[19] G. Humphreys, M. Eldridge, I. Buck, G. Stoll, M. Everett, and P. Hanrahan. WireGL: A Scalable Graphics System for Clusters. In *Proceedings of SIGGRAPH 2001*, 2001.

[20] P. Lindstrom, D. Koller, W. Ribarsky, L. F. Hodges, N. Faust, and G. Turner. Real-time, continuous level of detail rendering of height fields. *Computer Graphics, Proceedings SIGGRAPH'96*, pages 109–118, 1996.

[21] P. Lindstrom and V. Pascucci. Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. *IEEE Transaction on Visualization and Computer Graphics*, 8(3):239–254, 2002.

[22] S. Madougou and J. Vairon. Fast deployment of a commodity vr cluster. *VR-Cluster'03 Review*, 2003.

[23] J.S. Montrym, D.R. Baum, D.L. Dignam, and C.J. Migdal. InfiniteReality: A Real-Time Graphics System. In *Computer Graphics (SIGGRAPH 97)*, pages 293–303. ACM Press, August 1997.

[24] http://www-unix.mcs.anl.gov/mpi.

[25] J. Rossignac and P. Borrel. Multi-resolution 3d approximations for rendering complex scenes. *Geometric Modeling in Computer Graphics*, pages 455–465, 1993.

[26] S. Rottger, W. Heidrich, and P. Slussallek. Real-time generation of continuous levels of detail for height fields. *Proceedings in 6th International Conference in Central Europe on Computer Graphics and Visualization*, pages 315–322, 1998.

[27] R. Samanta, T. Funkhouser, K. Li, and J. P. Singh. Hybrid Sort-First and Sort-Last Parallel Rendering with a Cluster of PCs. In *SIGGRAPH/Eurographics Workshop on Graphics Hardware*, August 2000.

[28] B. Schaeffer. A Software System for Inexpensive VR via Graphics Clusters. http://www.isl.uiuc.edu/ClusteredVR/ClusteredVR.htm, 2000.

[29] J. Stam. Real-time fluid dynamics for games. *Proceedings of the game developer conference*, 2003.

[30] http://www.top500.org.

[31] J. Xia, J. El-Sana, and A. Varshney. Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics*, pages 171–183, 1997.