Eurographics Symposium on Parallel Graphics and Visualization (2004)
Dirk Bartz, Bruno Raffin and Han-Wei Shen (Editors)

A Scalable Cluster-based Parallel Simplification Framework for Height Fields

V. Gouranton[1], S. Limet[1], S. Madougou[1,2] and E. Melin[1]

[1]Laboratoire d'Informatique Fondamentale d'Orléans, LIFO Orléans, France
[2]Bureau de Recherches Minières et Géologiques, BRGM, France
This work is partially supported by the RNTL project Geobench of the french ministries of research and industry.

Abstract

In this paper, we present a method to interactively render 3D large datasets on a PC Cluster. Classical methods use simplification to fill up the gap between such models and graphics card capabilities. Unfortunatelly, simplification algorithms are time and memory consuming and they allow real-time interaction only for a restricted size of models. This work focuses on parallelizing Rottger's simplification algorithm for height fields but the main ideas can be generalized to other scientific areas. The method benefits from the scalable computating power of clusters. As our results show it, this permits us to achieve a data scaling while maintaining an acceptable frame rate with real-time interaction. Moreover, the scheme can take avantage of tiled-display environments.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Parallel simplification, Virtual Reality, Real-time Rendering, Level of Detail Algorithms

1. Introduction

The work presented in this paper was initiated to solve a technological lock encountered by the French Geological Survey (BRGM) namely the rendering and the navigation in geological models displayed on 3D virtual reality environment driven by a cluster of PC. Our work intends to answer a more general question: how to use as best as possible the scalability of a VR Cluster in terms of storage capacity and graphical computing power for rendering such models ?

Geological models are often described by height fields *i.e.* rectangular grids of elevation data $H(x,y)$ with points regularly spaced in x and y axis. Owing to technological strides in collecting geographical and geological informations, height fields modeling Earth ground and underground have became very large. Today, large datasets consisting of hundreds of thousands to billions of polygons are commonplace. For example, geospatial data extracted from satellites easily go beyond tens of giga-bytes [gto]. So geological models usually exceed system memory storage capacity of current computers. Moreover, for Virtual Reality applications, the graphical rendering must be performed in real-time to achieve a realistic interaction with the virtual world. In this case, even if the model fits in system memory, it is not sufficient since it must be manipulated in the graphics board for rendering at interactive frame rate. Then the memory storage available is again reduced despite the spectacular progress in computer graphics area.

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. Simplification or level of detail algorithms (LOD) are good candidates to obtain a graphical rendering performed in real-time. This is essential to achieve a realistic interaction with the virtual world. These algorithms dynamically modify the visualized data according to their position in the space.

There are various kinds of simplification algorithms (see [HG97] for a survey). One class of them fits to large height fields: view-dependent algorithms [DWS*97, Hop98, LE97, LKR*96, CRM02, ESC00] and

© The Eurographics Association 2004.

delivered by
EUROGRAPHICS DIGITAL LIBRARY
www.eg.org diglib.eg.org

[EMB01, Pri00]. These algorithms rely on distance to view point to perform the simplification or the refinement. Therefore, two different regions can be at different resolutions, what is convenient for large terrains. Although they are often complex and deal with large datasets, 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 [LP02, CRM02, ESC00, EMB01, Hop97, Pri00] 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 height fields. For that we take into account that an alternative solution to high-end dedicated computers usually used for Virtual Reality has emerged for a few years: VR clusters. They are composed of off-the-shelf PC equipped with graphics boards and interconnected by an efficient network. 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 and as far as we know none of them allows a scalability on a cluster of PC with interactive frame rate.

El-Sana and Varshney [ESV99] proposed one of the first works on parallelizing view-dependent LOD algorithms. They worked on Xia's merge trees [XESV97] . Merge tree is a data structure built upon Hoppe's progressive meshes [Hop96]. 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 exclusively intended for shared memory machines which do not fit our purposes.

In [DLR00], 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 [BP02] is a parallel extension to R-Simp [BW00], 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 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 $log n$ iterations is used to merge remaining parts on master. This is costly and places a bottleneck on the master. PR-Simp allows 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 height fields. We choose Rottger's algorithm as an illustration of our framework, for convenience and performance reasons but other LOD algorithms may be parallelized in this way. The paper shows how one can benefit from storage and computing power of clusters to obtain data scaling by using the regular structure of height fields and rendering techniques such as view frustum culling. The remainder of the paper is structured as follows: in section 2 we give an overview of Rottger's algorithm. Parallelization description follows in section 3. After giving and analyzing benchmarks results in section 4, we conclude in section 5.

2. Rottger's algorithm

We have chosen Rottger's algorithm to illustrate our work because of three main qualities. It has been designed for height fields, it is view point dependent which is very important when navigating in the height field and it does not require sophisticated memory consuming data structures.

This algorithm [RHS98] is based of the work of Lindstrom and al. [LKR*96]. In the latter paper, Lindstrom and al. describe a view-dependent algorithm for height fields of size $(2^n+1)*(2^n+1)$. The algorithm dynamically modifies a quadtree by using a bottom-up strategy to determine whether a node needs to be subdivided or merged with adjacent nodes. For that a boolean criterion is evaluated. The criterion uses the upper bound of the projected pixel error. One disadvantage of this algorithm is that the pixel error function must be evaluated for each point of the height field, what is costly in computation time.

Rottger's algorithm, contrary to that of Lindstrom, uses a top-down approach to create the triangulation and the vertex removal is performed based on distance to view point and local surface roughness. The top-down strategy allows to visit just a fraction of the dataset: 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. For example, the quadtree matrix entries that correspond to points drawn in black in the triangulation in (fig.1) are set to 1. After the triangulation is finished, Rottger algorithm draws triangle fans whereas Lindstrom draws triangle strips. Both structures allow to reduce the amount of data sent to

© The Eurographics Association 2004.

graphics pipeline but fans better capture the roughness of a surface.

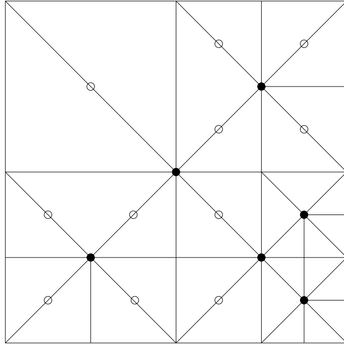

Figure 1: *A triangulation of* 9×9 *height field from Rottger's algorithm.*

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 raise 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 the LOD of adjacent sub-nodes differ by no more than one which is guaranteed by the way the algorithm computes and stores surface roughnesses.

3. Parallelizing the Level Of Detail Algorithm

Rottger's algorithm has two main drawbacks for our purposes. The first one is common to all in-core sequential simplification algorithms: the entire dataset needs to fit in the system memory and this is often not the case for geological models. The second one is that this algorithm has not been designed for tiled-display environments. Our aim is to obtain a scalable parallelization framework to benefit from the extensibility of cluster architecture with regard to memory storage capacity and computing power. To achieve this goal, the cluster nodes are partitioned into two classes: visualization nodes and computation nodes.

Basically, the algorithm is composed of three stages:

- **Initialization**: it takes place only once. The height field is distributed to the computation nodes. This allows to benefit from system memory of each computation node
- **Communication**: visualization nodes broadcast their current point of view and frustra to computation nodes and computation nodes send to each rendering node the part of scene they have to render.
- **Computation**: computation nodes run Rottger's algorithm on the part of the height field they are in charge while visualization nodes display the part of the scene they have received.

The algorithm can be summarized in the following pseudo-code:

```
computation node:
   get local data
   while true
      get view point and frusta
      perform LOD
      for each visualization node nv
         do culling against frustrum of nv
         send data to nv
      end for
   end while

visualization node:
   while true
      broadcast view point and frustum
                to computation nodes
      get rendering data
      draw the subscene
   end while
```

One can remark that computation nodes perform a special data culling 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 follows in section 3.1 and how to minimize the communication costs is described is section 3.2.

3.1. Data partitioning

Data partitioning takes place at the initialization stage. Data are 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. The grid size is constrained to be of the form $(2^n+1)*(2^n+1)$. In most cases, height fields do not have this structure therefore data distribution has to take into account this constraint. Given a height field of size $SzX * SzY$, we first search the smallest number m such that :

$$2^{m-1}+1 < max(SzX,SzY)*\frac{2}{CN} \leq 2^m+1$$

where CN is the number of computation nodes. Each computation node gets a portion of size 2^m+1, allowing overlap if necessary. We avoid sending overlapped regions several times. The first sender sends all its visible data, including overlapped. The second all overlapped that are not sent by

© The Eurographics Association 2004.

the first sender, and so on. This introduces load unbalance on communications. Unfortunately, this is not still adressed.

3.2. Minimizing communication costs

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

3.2.1. Sending only indices and heights

One advantage of working on height fields is that one can rely on their regular structure to do certain optimizations. Height fields often are regularly sized in the horizontal plane. 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. Given SzX and SzY (fig.2) the x and y sizes of the height field, (llx, lly) its lower left corner, let $Dim = 2^n + 1$ the extent of each portion and $SpaceX$ and $SpaceY$ the increments in x and y directions, we can compute x and y coordinates of any vertex v for which we know the holder(processor p_k) and indices in local grid (i_k, j_k). Indeed

$$x_v = llx + (i_k + (k-1) * Dim) * SpaceX \quad (1)$$

Thus, given a vertex $v = (x, y, h)$ that needs to be drawn, just $(f(i, j), h)$ is sent to visualization nodes, where $f(i, j) = j * Dim + i$. In the initialization stage, the values of llx, lly, SzX, SzY, $SpaceX$, $SpaceY$ and Dim are sent to visualization nodes. During the rendering, when they receive v, the indices are extracted from $f(i, j)$ and above equations are used to determine x and y coordinates. Although we give only the x coordinate, the equations above are valid for y. This scheme allows to reduce communication costs by 33% because 2 floats instead of 3 are sent for each point to be drawn.

3.2.2. 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 Rottger's 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 fan that intersects the frustum even partially, we have to test against the fan's bounding sphere. With this processing, only a relatively small part of computation nodes' portion needs to be sent (fig.3). 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.

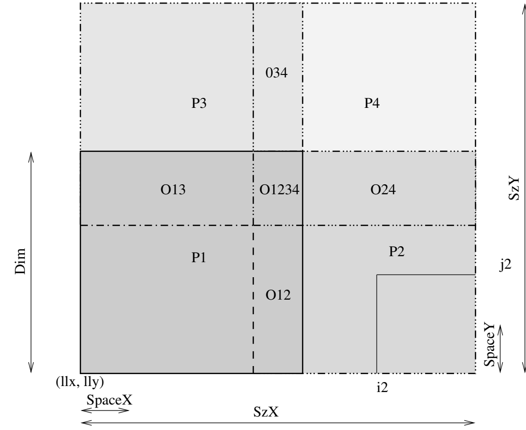

Figure 2: *Repering a vertex's x et y coordinates using indices. P_k is the squared area allocated to processor p_k. There are regions where these areas overlap, what we note $O_{ij[kl]}$.*

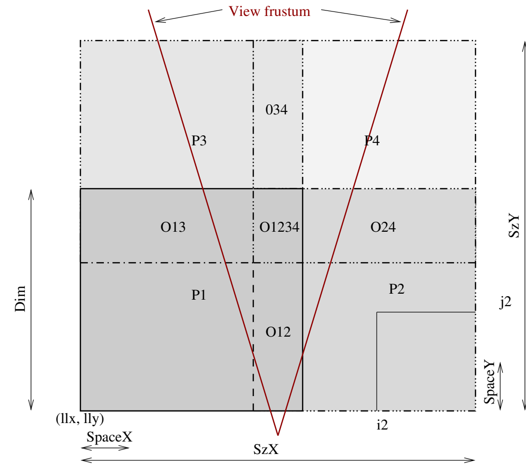

Figure 3: *Data sent per node in case of 4 computation nodes. Overlap regions are sent once. For the processor p_k, only the intersection of S_k and the viewing frustum is sent.*

Our parallelization scheme for LOD algorithm is data scaling since the height field is distributed among computation nodes only once, at the begining, and they never communicate to each other afterwards. It also strongly limits the communication overhead from LOD nodes to visualization nodes thanks to fustrum culling. It should be noticed that no process in our framework has a special task such as master-

© The Eurographics Association 2004.

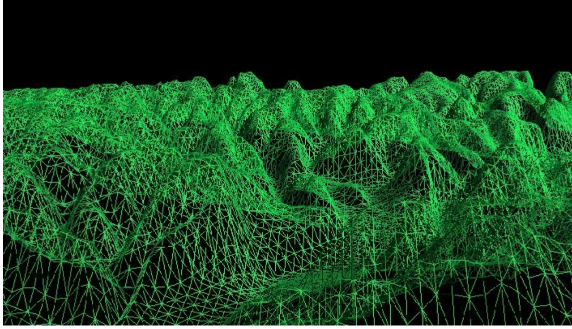

Figure 4: *LOD near the view point.*

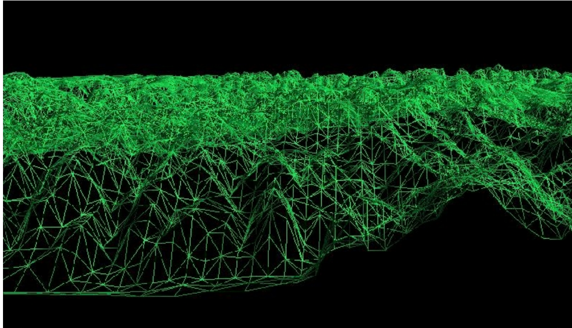

Figure 5: *LOD at farther regions to the view point.*

ing the others for gathering data, which increases its scalabity capabilities.

4. 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 [AGL*02] as cluster middleware. This facilitates inter-frame synchronization and communications as it allows direct calling of MPI [mpi] routines. As Net Juggler is built upon VR Juggler [Bie00], we easily call OpenGL Performer [per] commands through this software. We found Performer very convenient for its higher abstraction of graphics objects. It also easily makes available to the programmer informations about graphics context such as frustum and viewpoint. The data we use is provided by the French Geological Survey (BRGM) and represents a height field of 1000×760 points. This represents about 1,520,000 triangles. We use this as one layer and then create other layers for test purposes. Thus, these layers have the same amount of data and same characteristics as the original height field. They only are there to enable us to test more important data. The resolution we use is of 1280×1024 per visualization node. So the resolution for four visualization nodes, for example, is 2560×2048. All tests are performed in the same conditions such as model initial position and level of detail parameters. The interaction is automated.

Terrain portions processed by different computation nodes are assembled on visualization nodes. To avoid cracks at bounderies, these portions are kept at the same level. As Rottger's algorithm takes care about this in each portion, the whole terrain remains almost consistant. But artefacts are still visible at thin overlapped regions that persist after the processing described in section 3.1.

We have tried to render only one layer without any simplification. The frame rate of about 10 is approximatively divided by two when adding a second layer. This illustrates the need of a LOD algorithm for obtaining a framework usable in Virtual Reality. In figures below, $n-m$ stands for parallel LOD using n computation nodes and m visualization nodes.

4.1. Comparison between our sequential and parallel implementations

First, we compare the sequential algorithm with our parallel version according to the execution time of the pure LOD algorithm. The parallel LOD uses 4 computation nodes and 1 visualization node. To clearly show the contribution of parallelization, we evaluated the ratio between sequential LOD algorithm time and parallel LOD algorithm time for different amounts of data, *i.e.* for different numbers of layers. This ratio is constant (about 2.75), thus independent from the amount of data. The optimum is not reached because of overlapped regions.

Next we compare the sequential algorithm with our parallel version according to the frame rate (fig.6). The parallel version uses one or four computation nodes and only one visualization node in both case. Note that the sequential version uses the same node for both computation and visualization.

Results are summarized in figure 6. One can oberve that until the fourth layer (more than 4 millions of triangles at the finest level), the frame rate remains above 10. This is still usable for interactive applications. This figure also highlights that the parallel algorithm with one computation node and one visualization node provides a better frame rate than the sequential one. This illustrates the interest of distributing the LOD and the pure rendering work on different nodes. Figure 6 shows that there is a frame rate gain of about 80 % from pure sequential version to the parallel version using 1 visualization node and 1 LOD node. This is due to the implemented optimizations, namely the view frustum culling that seriously alleviates communications and graphics pipeline load. However, this frame rate gain drops to about 25% between parallel versions (1 and 4 nodes). There are three reasons to that. First, as the number of computation nodes increases a bottleneck appears on the visualization node. Second, in spite of optimizations, the communication time in-

© The Eurographics Association 2004.

creases with the amount of data . Third, we do not perform triangle budget [DWS*97, LE97], so despite the view frustum culling, data merely saturate the graphics board. However, communications time and rendering time directly determine the frame rate.

Note that the memory of single computation node is unable to manage a model composed of 6 layers. This illustrates another interest of the parallel algorithm which allows the use of the memory of the entire cluster.

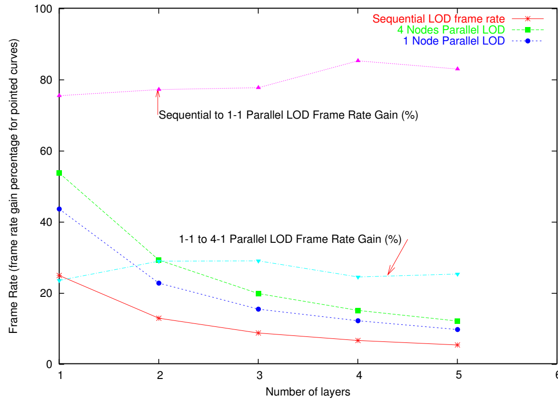

Figure 6: *Average frame rates and frame rate gains between sequential and parallel algorithms.*

4.2. Application to tiled-display environments

As described in section 3, our framework allows each computation node to send different graphics data to each visualization node according to its current view frustum. This partially avoids the bottleneck occuring when using one single visualization node. This is straightforward in a tiled-display environments driven by a VR Cluster. The figure 7 shows the average frame rate for different amounts of data (numbers of layers) in 2 setups: both with 4 computation nodes but one with only 1 visualization node and the second with 4 visualization nodes. As one can see, the second setup is better for all sizes. The curve pointed by an arrow shows the frame rate gain between the 2 setups. For interactive applications, the speed up is limited by uncontrolled load repartition between graphics boards. Indeed, the scene is ideally divided in 4 equal parts (i.e. each part contains the same number of triangles) but in fact, it is seldom true because the part of model displayed by each node depends on the user interaction. Despite this, we obtain an interesting speed up. There are two main causes for that. First our technique lightens the graphics board load. Second, the amount of data that are exchanged is almost the same when using one or four visualization nodes, but in the case of one visualization node bottleneck is unavoidable whereas in the case of four visualization nodes, two different computation nodes can send their data to different visualization nodes in parallel thanks to ethernet switch abilities.

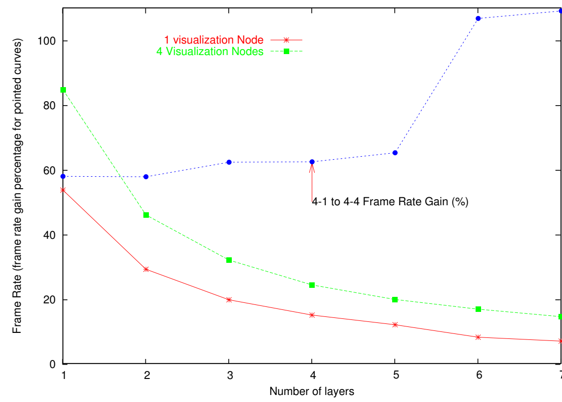

Figure 7: *Average frame rates and frame rate gain for 1 and 4 visualization nodes using 4 computation nodes.*

4.3. Towards processing very large datasets

We show that it is possible to create a parallel version of LOD algorithm to manage more data than a sequential version. Moreover this code is scalable and can achieve better frame rates as we use more computation nodes. We also take into advantage multi-display environment to prevent the creation of bottlenecks on visualization nodes.

Nevertheless if the datasets become too large for a given VR cluster, we have to implement a triangle budget technique to achieve a constant frame rate. Classically this approach has a noticeable over-cost since the LOD has to iterate until it reaches the desired number of triangles. Our approach is totally compatible with triangle budget implementation. Using our framework, the triangle budget algorithm is parallelized and its over-cost minimized. However, we do not activate it for benchmarks as we aim to determine real data scalability capacities of our method at a constant visual quality.

5. Conclusion

We have presented a parallelization framework for LOD algorithms to navigate in large height fields. We have shown that it has been designed to take advantage of the scalability of VR cluster architectures by distributing data on the cluster nodes and avoiding useless communication and data gathering. Our benchmarks exhibit encouraging results.

Some improvement may be done to reduce communication traffic on the network by observing that 80% to 90% of the scene does not change between two LOD computations. Therefore it should be possible to only send the differences between the two scenes. Such techniques usually have a memory overhead which should be limited to be used for our purposes.

It would be also interesting to apply our techniques for

visualizing and manipulating other kinds of datasets. For example, scientific simulations are able to produce large amount of time-varying data , in contrast to height fields. Some LOD techniques should be adapted to this kind of data and our framework should help to parallelize them.

Acknowledgements

We would like to acknowledge the BRGM (French Geological survey) for providing height fields data for our experiments. We also would like to thank the anonymous referees for their valuable remarks.

References

[AGL*02] ALLARD J., GOURANTON V., LECOINTRE L., MELIN E., RAFFIN B.: Net juggler: Running vr juggler with multiple displays on a commodity component cluster. *IEEE VR'02* (2002).

[Bie00] BIERBAUM A.: Vr juggler: A virtual platform for virtual reality applications development. *Master's thesis* (2000).

[BP02] BRODSKY D., PEDERSEN J. B.: Parallel model simplification of very large polygonal meshes. *PDPTA'02* (2002), 1207–1215.

[BW00] BRODSKY D., WATSON B.: Model simplification through refinement. *Graphics Interface'00* (2000).

[CRM02] CIGONI P., ROCCHINI C., MONTANI C. SCOPIGNO R.: External memory management and simplification of huge meshes. *IEEE Transaction on Visualization and Computer Graphics* (2002).

[DLR00] DEHNE F., LANGIS C., ROTH G.: Mesh simplification in parallel. *ICA3PP'00* (2000), 281–290.

[DWS*97] DUCHAINEAU M., WOLINSKY M., SIGETI D., MILLE M., ALDRICH C., MINEEV-WEINSTEIN M. B.: Roaming terrain: Real-time optimally adapting meshes. *IEEE Visualization* (1997), 81–88.

[EMB01] ERIKSON C., MANOCHA D., BAXTER W. V.: Hlods for faster display of large static and dynamic environments. *ACM Symposium on Interactive 3D Graphics* (March 2001), 111–120.

[ESC00] EL-SANA J., CHIANG Y.-J.: External memory view-dependent simplification. *Computer Graphics Forum 3*, 19 (August 2000), 139–150.

[ESV99] EL-SANA J., VARSHNEY A.: Parallel processing for view-dependent polygonal virtual environments. *Proceedings SIGGRAPH'99* (1999).

[gto] http://edcdaac.usgs.gov/gtopo30/gtopo30.htm.

[HG97] HECKBERT P. S., GARLAND M.: Survey of polygonal surface simplification alogrithms. *Multiresolution surface modeling course notes, ACM SIGGRAPH'97* (1997).

[Hop96] HOPPE H.: Progressive meshes. *In proceedings SIGGRAPH'96* (1996), 99–108.

[Hop97] HOPPE H.: Smooth view-dependent level-of-detail control and its application to terrain rendering. *IEEE Visualization '98 31* (1997), 189–198.

[Hop98] HOPPE H.: View-dependent refinement of progressive meshes. *Computer Graphics (In proceedings SIGGRAPH'97) 31* (October 1998), 35–42.

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygonal environments. *Computer Graphics (Proc. siggraph'97) 31*, 3 (1997), 199–208.

[LKR*96] LINDSTROM P., KOLLER D., RIBARSKY W., HODGES L. F., FAUST N., TURNER G.: Real-time, continuous level of detail rendering of height fields. *Computer Graphics, Proceedings SIGGRAPH'96* (1996), 109–118.

[LP02] LINDSTROM P., PASCUCCI V.: Terrain simplification simplified: a general framework for view-dependent out-of-core visualization. *IEEE Transaction on Visualization and Computer Graphics 8*, 3 (2002), 239–254.

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

[per] http://www.sgi.com/software/performer.

[Pri00] PRINCE C.: *Progressive Meshes for Large Models of Arbitrary Topology*. Master 's thesis, University of Washington, 2000.

[RHS98] ROTTGER S., HEIDRICH W., SLUSSALLEK P.: Real-time generation of continuous levels of detail for height fields. *Proceedings in 6th International Conference in Central Europe on Computer Graphics and Visualization* (1998), 315–322.

[XESV97] XIA J., EL-SANA J., VARSHNEY A.: Adaptive real-time level-of-detail-based rendering for polygonal models. *IEEE Transactions on Visualization and Computer Graphics* (1997), 171–183.

© The Eurographics Association 2004.