Eurographics Symposium on Parallel Graphics and Visualization (2009)
J. Comba, K. Debattista, and D. Weiskopf (Editors)

Distributed Visualization of Complex Black Oil Reservoir Models

Frederico Abraham and Waldemar Celes

Tecgraf - Computer Science Department, PUC-Rio
Rua Marquês de São Vicente 225, 22450-900 Rio de Janeiro, RJ, Brasil
{fabraham,celes}@inf.puc-rio.br

Abstract

Recent accomplishments in the computer simulation of black oil reservoirs have created a demand for the visualization of very large models. In this paper, we present a distributed system for the rendering of such models. Following recent trends in the high performance computing area, the system is intended to make the visualization of these models available to lightweight clients on corporate networks, through the use of a cluster of inexpensive off-the-shelf PCs equipped with multiple GPUs. The proposed system uses a sort-last approach and supports a diverse set of visualization techniques. Through an efficient use of each GPU and a partial composition stage on each cluster node, our solution tackles the scalability issues that arise when using mid-to-large GPU clusters. Experimental results show that our implementation can sustain the visualization of models with up to 60 million cells at interactive rates, using a cluster with 16 nodes, each one equipped with 4 GPUs. Experimental results also demonstrate the scalability of the proposed solution.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Graphics Systems—Distributed/network graphics I.3.3 [Computer Graphics]: Picture/Image Generation, Viewing algorithms—I.3.8 [Computer Graphics]: Applications—C.2.4 [Computer-Communication Networks]: Distributed Systems, Distributed Applications—

1. Introduction

Recent advances in parallel architectures for the numerical simulation of black oil reservoirs have allowed the use of very discretized domains. As a consequence, these simulations have produced very large datasets that must be visualized in 3D environments for analysis and inspection. Conventional scientific visualization techniques of such very large models are not viable. In order to achieve interactive visualization rates, one has to employ scalable visualization solutions.

Similar large-scale visualization problems have been handled through the use of clusters of commodity PCs equipped with specialized and inexpensive high-performance graphics hardware. Nowadays, computers equipped with multiple GPU cards have also proved to be another attractive option for the visualization of large datasets [MMD08]. Moreover, recent trends in high-performance computing have moved heavy computations from desktop computers to very well-equipped computing data centers, especially in the oil industry.

In this paper, we present a distributed visualization system for large-scale black oil reservoir models. Our solution attempts to explore the full graphics processing power of a cluster of PCs, where each node is equipped with multiple graphics cards. The system is designed to allow lightweight clients to benefit from a distributed rendering system accessed through conventional corporate networks. As a consequence, the end user does not need to be physically close to the cluster.

The proposed system employs the sort-last distributed strategy and supports a diverse set of visualization techniques. With an efficient use of the available GPUs, combined with a pipelined implementation and the use of partial image compositions on the cluster nodes, our solution tackles the scalability issues that arise when using mid-to-large GPU clusters. Experimental results demonstrate that our so-

© The Eurographics Association 2009.

DOI: 10.2312/EGPGV/EGPGV09/087-094

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

lution can sustain the visualization of very large reservoir models at interactive rates.

The remainder of this paper is organized as follows. In the next section, related works on distributed rendering and multiple GPU rendering are presented. Black oil reservoir models and the corresponding usually applied visualization techniques are briefly presented in Section 3. Section 4 presents our proposed distributed pipeline architecture and implementation. Section 5 presents the experimental results and demonstrates the effectiveness and efficiency of our solution. Finally, in Section 6, some concluding remarks are drawn and future work is discussed.

2. Related Work

Molnar et al. [MCEF94] presented a classification of parallel rendering strategies based on the placement of the visibility sorting phase in the graphics pipeline: sort-first, sort-middle and sort-last. In the sort-first strategy, the screen is partitioned into disjoint tiles, and each cluster node is responsible for rendering (thus sorting) all primitives overlapping its corresponding tile. Sort-middle systems distribute primitives after the geometry stage and before the rasterization stage, not being suitable for modern graphics hardware. Sort-last systems split the model among the rendering nodes, and each node runs the entire local rendering pipeline on its corresponding submodel. Sorting the partial images of each node resolves the visibility problem and is the last stage of the pipeline, executed on a master node. Molnar et al. [MCEF94] have elected sort-last as the most scalable strategy in terms of number of primitives, being definitely the choice for rendering massively complex models.

Wylie et al. [WPLM01] built a scalable sort-last system based on PC clusters. They implemented simple strategies to partition the data for parallel rendering, most of them based on the number of triangles, demonstrating excellent load-balancing characteristics in terms of geometry processing and good load-balancing in terms of rasterization when the triangle count is relatively large.

Humphreys et al. [HHN*02] have presented Chromium, the successor to WireGL [HEB*01], a system for manipulating streams of OpenGL API commands on a cluster of PCs. The system allows both sort-first and sort-last distribution of graphics application workload without requiring changes to the source code of the application. However, we believe application-dependent optimizations, while less generic, are necessary in the context of massively large datasets.

Van der Schaaf et al. [dSRG*02] have compared the use of immediate mode, such as WireGL [HEB*01], with retained mode rendering paradigms. Two approaches were experimented: replicating data on the nodes and broadcasting graphics commands over the network. The replicated-data approach achieves better performance while broadcasting simplifies transparency for handling user input and rendering synchronization.

Marchesin et al. [MMD08] presented a sort-last volume visualization system based on a single machine driving multiple GPUs. The results show that their system competes well with mid-sized clusters of single GPU PCs. This makes the use of multiple GPU cards a definite plus to increase the graphics power of a single PC at low cost. Other previous works on multiple card rendering, like NVIDIA's SLI [nvib] and Quadro Plex [nvia], use sort-first to distribute the workload among the GPUs, which does not scale well with the size of the model.

Cavin et al. [CMF05] implemented a pipelined sort-last rendering for large volumetric and polygonal datasets using commodity off-the-shelf PC clusters. Their solution demonstrates that low cost clusters, which exclude expensive specialized networks, such as Myrinet and Infiniband, and hardware image compositors, can be viable competitors to more expensive solutions. Later on, Cavin et al. [CM06] presented a theoretical and practical performance analysis of pipelined sort-last implementations for both polygonal and volume rendering.

Different from most previous proposals, we have designed our system to allow the connection of lightweight clients, through the use of conventional corporate networks. We have employed the sort-last strategy for rendering very large black oil reservoir models, taking advantage of nodes equipped with multiple GPUs.

3. Black oil reservoir simulation

Black oil reservoirs are formed through the accumulation of hydrocarbons in sedimentary rocks [Dak78]. These resources are extracted by drilling wells into the reservoir field and connecting them to a pipeline network for storage and processing. Natural or induced underground pressure forces the oil (and gas) to flow to the surface. In order to maximize resource recovery, the oil industry uses numerical simulators to predict fluid flow. By simulating different well arrangements and configurations, the reservoir is economically evaluated and its exploration is planned.

3.1. Reservoir models

A black oil reservoir model is defined by a discrete mesh of hexahedral cells topologically organized on a tridimensional grid. Each cell is identified by a triple $[i, j, k]$ with cells $[i+1, j, k]$, $[i-1, j, k]$, $[i, j+1, k]$, $[i, j-1, k]$, $[i, j, k+1]$ and $[i, j, k-1]$ as topological neighbors. The geometry is usually irregular, as shown in the Figures 1-5. Due to discontinuities in elevation, characterizing geological *faults*, topological neighbor cells may not share faces. Also, cells in the topological grid might be set as *inactive*, yielding irregular (and, possibly, disconnected) groups of cells. All cells of a

© The Eurographics Association 2009.

given k constitute a *layer*, which normally resembles an irregular terrain, with discontinuities.

Based on geophysical and geological information, the reservoir is characterized assigning properties to the cells. For a given well arrangement, the simulator computes the oil (and gas) flow based on its numerical model. As a result, the simulation outputs the well's production data and physical properties (such as oil, gas, and water saturation) at each cell for each time step of the simulation.

3.2. Reservoir visualization

The numerical simulation results in large datasets. Scientific visualization techniques are then applied for inspecting such results. The goal is to provide the engineers a rich set of graphics tools for unambiguous analysis of the model.

A variety set of 3D visualization techniques may be offered to the user:

- conventional iso-contouring of cell faces for scalar field visualization (Figure 1);
- smoothed scalar fields with iso-lines (Figure 2);
- separation of reservoir layers for better inspection of the internal reservoir structure (Figure 3);
- positioning of arbitrary cutting planes (Figure 4);
- volume rendering algorithm of non-structured meshes (Figure 5: in our system, volume rendering is achieved by using a GPU-based ray-tracing algorithm [EC05]).

One main challenge of black oil reservoir visualization is the ability to handle very large models. Due to the availability of increasing computer power and to improve simulation accuracy, the oil industry has employed reservoir models composed by tens of millions of cells – very recent advances in parallel simulations have allowed the simulation of a reservoir with one billion cells [sau]. Visualizing such complex models at interactive rates requires the use of more elaborate visualization techniques. The use of culling techniques, such as frustum and occlusion culling, reduces the

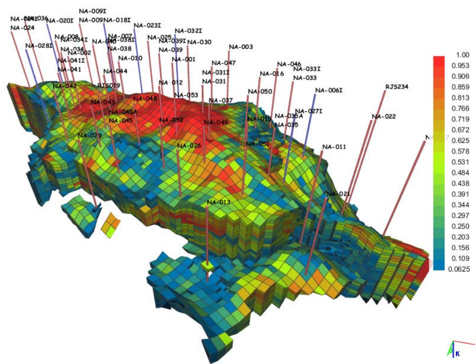

Figure 1: *Visualization algorithms: original coordinates iso-contour.*

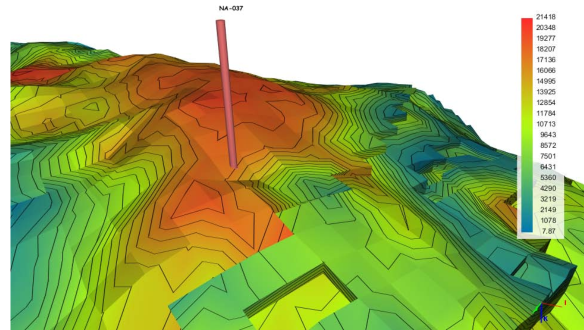

Figure 2: *Visualization algorithms: smoothed property with iso-lines.*

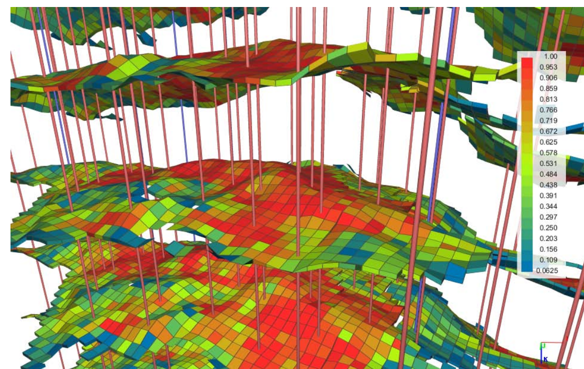

Figure 3: *Visualization algorithms: iso-contour with layer separation.*

number of primitives sent to the graphics pipeline but do not suffice for very large models. Two different approaches are eligible for handling such models: level-of-detail technique and distributed visualization. Level of detail, when applied to reservoir models, faces the following challenges: (i) the technique imposes changes in the topology of the model, which may lead to incorrect result interpretations; (ii) the model simplification may depend on the scalar field being visualized, turning it difficult to build, in a pre-processing phase, an unique hierarchical structure suitable for all scalar fields; (iii) the technique is not adequate for high resolution displays, which are currently a trend in the oil industry. The level-of-detail technique has the advantage of allowing the use of a single workstation for visualizing complex models, and this may appear as a strong requirement in some applications. On the other hand, distributed visualization does require the use of multiple processing units. However, in the context of the oil industry, the existence of large computing data centers is a reality. For that reason, this paper focuses on the use of a cluster of GPU-equipped PCs for reservoir visualization. Our goal is to build a distributed visualization sys-

© The Eurographics Association 2009.

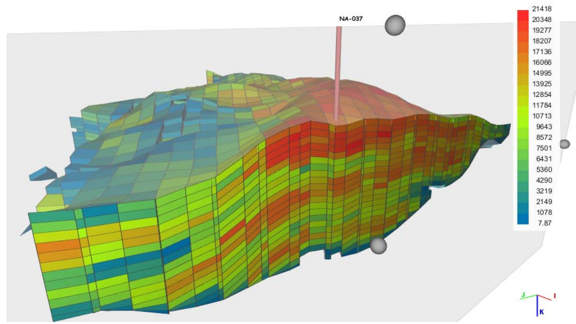

Figure 4: *Visualization algorithms: arbitrary cutting planes.*

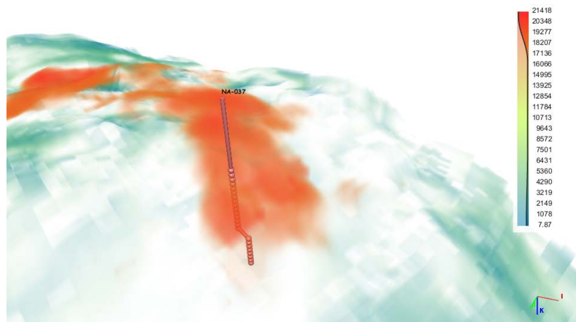

Figure 5: *Visualization algorithms: ray-casting volume rendering.*

tem that supports the different techniques commonly used for reservoir visualization, allowing the inspection of very large models at interactive frame rates.

4. Proposed distributed visualization

Previous related works indicate sort-last as the best suited parallel rendering strategy for very large models [MCEF94, WPLM01, CMF05, CM06, MMD08], mainly because the model is subdivided among the nodes. Sort-first and sort-middle, the other two distributed strategies, require the entire model to be loaded on each rendering node, which is not feasible for very large models given hardware constraints.

As pointed out by Molnar et al. [MCEF94], although scalable in terms of model size, the scalability of the sort-last strategy suffers from having to deal with an increasing count of partial images from the nodes. These images have to be sent over the network for composition, either by depth-buffer visibility sorting or alpha-blend composition.

In this section, we describe our distributed system for large reservoir model visualization, and discuss our proposal to achieve scalability and to minimize the impact of dealing with a large number of partial images.

4.1. Hardware architecture

Our system was designed to use a cluster of PCs, each one equipped with a PCI-express bridge with multiple GPU cards attached to it. In this way, we take advantage of the low cost per bandwidth ratio of such buses, as attested by Marchesin et al. [MMD08].

Figure 6 illustrates the hardware layout of our solution. Users requiring the visualization of large models can connect to the cluster *master node*. This node coordinates the distributed rendering of complete frames, which are then sent to the *client* over a conventional corporate network. The other cluster PCs serve as *renderer nodes*. The cluster nodes are interconnected by a fast network. Each server PC spawns, for each GPU, a rendering thread that controls its graphics pipeline. New reservoir models are transmitted from the client to the master node, which stores them in a shared file system.

4.2. Model partitioning

The first step for an efficient distributed visualization resides in adequately partitioning the model among the renderers. Once the model has been stored in the shared file system, the master node executes the partitioning algorithm. We use a KD-tree structure to recursively split the axis-aligned bounding box of the entire model. Considering that *nc* denotes the number of cells in the current (sub)model and n the number of renderer nodes, at each recursion step, the *nc* cells are sorted according to their centroid coordinates along the axis corresponding to the largest bounding box dimension. The (sub)model is then split in two parts, where the first part are allocated to the first $\lfloor \frac{n}{2} \rfloor$ nodes, and the second to the remaining $n - \lfloor \frac{n}{2} \rfloor$ nodes. The numbers of GPUs in the first and second node groups are used as weights to position the splitting plane: if the first and second groups have respectively g_1 and g_2 GPUs, the splitting plane is placed at the centroid of the cell at index $nc * \frac{g1}{g1+g2}$ of the sorted cell ar-

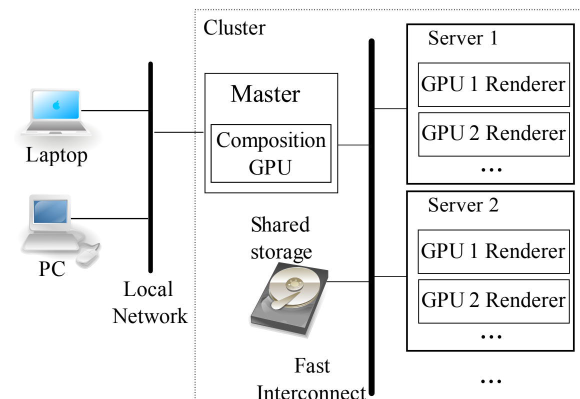

Figure 6: *Hardware layout of our solution.*

© The Eurographics Association 2009.

ray. Cells crossing the splitting plane are allocated to both groups, and clipping planes are added for restricting rendering to inside the subdivided bounding boxes.

When the recursion reaches a group with only one node, the cells allocated to it are once again split, among the g GPUs of the current node. The same procedure applies, where the first and second parts is allocated to the first $\lfloor \frac{g}{2} \rfloor$ and the remaining $g - \lfloor \frac{g}{2} \rfloor$ GPUs, respectively.

This procedure results in a well balanced partition in terms of number of cells. The average screen coverage of each partition is also reduced, due to spatial properties of a KD-tree subdivision. This alleviates both the network usage and the composition cost. The procedure is simple and efficient; however, it requires the storage of all centroid coordinates in the master node memory. This may be a constraint for very large models but is not addressed in this work.

After the model has been split, each final partition bounding box is sent to the corresponding server node. The node loads all associated submodels (one for each GPU) and is then ready to spawn rendering threads with the desired visualization technique.

Our system does all image compositions on the GPUs. Since all submodel bounding boxes are convex and adjacent, it is straightforward to sort the partitions according to the viewer. This allows us to use the painter's algorithm to correctly resolve visibility on the composition stage: instead of reading and sending the entire depth buffer of the covered screen area and performing z-buffer composition, it suffices to read and send the alpha component of each pixel. An alpha value equal to 0 signals that the pixel was not written during the rendering. The alpha channel also suffices for composing subimages with transparency, needed by the volume visualization technique.

4.3. Distributed Pipeline

Figure 7 details our implementation. The execution running on each entity (client, master, and renderer) are divided into pipeline stages, each one executed on a separate operating system thread.

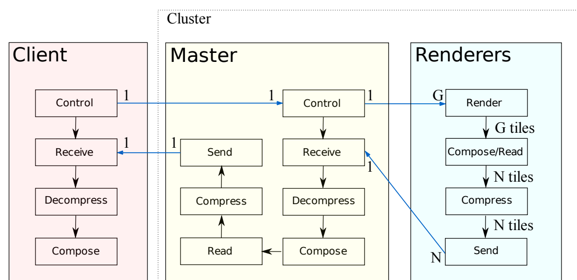

Figure 7: *Our pipelined sort-last architecture: G is the aggregate number of GPUs; N is the number of renderer nodes.*

4.3.1. Renderers

The renderer nodes are responsible for loading and effectively rendering the parts of the model assigned to all its GPUs. During a rendering session, each node runs a number of GPU rendering pipelines in parallel. As a result, each renderer node generates a set of partial images. However, instead of sending all partial images to the master node, we have opted for introducing a *partial composition* on the renderer node.

Based on the model partition described in Section 4.2, each renderer node composes the partial images generated by its GPUs in a back-to-front order. The first GPU in the back-to-front order is also responsible for performing the partial composition. Each other GPU rendering thread reads back the region covered by its bounding box and puts it available for the rendering thread of the first GPU, which is responsible for composing the partial images in the correct order. After composition, the union of all screen coverages is read back and queued for compression.

The compress stage compresses the partial images using the LZO compression library. This library provides extremely fast and simple lossless compression of byte arrays, being suitable for real-time applications like ours. The resulting compressed buffers are sent to the send stage. The send stage is responsible for sending the compressed images to the receive stage on the master node.

4.3.2. Master

The master node is responsible for controlling the distributed rendering. This control is implemented in the control pipeline thread. Its tasks include:

- keeping the camera parameters updated on all renderers;
- computing and sending the screen coverage of each bounding box; this is done at each frame using the algorithm described by Blinn [Bli96];
- forwarding all relevant application-specific parameters to the renderers;
- requesting the rendering of a new frame.

This thread also implements frame control. While one frame is being received and composed, another frame can be issued on all renderers, as in [ACCE04]. Although this results in an additional frame of latency, the parallelism between all pipeline stages is increased, resulting in better overall performance (in terms of frames per second).

The receive stage waits for the compressed partial images from each renderer node. Each received image is forwarded to the decompress stage, being then decompressed and sent to the compose stage. The partial images are finally composed in back-to-front order on the GPU of the master node.

Once the final image is ready, its RGB components are read back to the main memory for compression and transfer to the client node. In order to improve performance, the read,

© The Eurographics Association 2009.

compress, and send stages are implemented in parallel. The final image is split into horizontal tiles. The tiles are then read back using the asynchronous `glReadPixels` function, offered by the ARB_pixel_buffer_object OpenGL extension. In this way, compression and transfer are done in parallel with reading back: while one tile is being read, the previous readback result can be queued for compression and sent to the client node.

4.3.3. Client

The client node receives, decompresses and draws the final RBG image. These three stages run in parallel with the read, compress, and send stages on the master node. As we shall demonstrate, this parallelism in transferring the final image to the client results in a significant gain of performance when the client-master network bandwidth appears as the bottleneck of the system.

The client node also runs the control stage. This stage is responsible for handling user input and controlling the rendering engine implemented on the cluster.

5. Experimental Results

We have tested our system using a 16-node cluster, being each node equipped with a two dual-core 2.4 GHz AMD Opteron processor, 8 GB of RAM and connected by both a switched 1 Gbps Ethernet and an Infiniband 4x network. This allows us to test our system in both common off-the-shelf visualization clusters and clusters equipped with better and more expensive interconnections. The master node is equipped with one NVIDIA Geforce GTX 280 GPU. The 15 server nodes are equipped with 4 NVIDIA Quadro FX 5600, each one with 1.5 GB of video memory. These graphics cards were connected to two x16 PCI-express bridges. A detailed description of the cluster hardware can be found in [SEP*08]. The client system was a laptop connected to the cluster through a local Gigabit Ethernet network.

The system was implemented in C++, using OpenGL, pthreads and TCP/IP sockets on a Linux operating system. Each rendering thread uses frustum and occlusion culling techniques in order to reduce the number of primitives sent to the graphics pipeline at each frame. The implementation of these acceleration techniques follows the frustum culling algorithm presented by Assarsson et al. [AM00] and the hardware-assisted occlusion culling algorithm presented by Bittner et al. [BWPP04]. All tests were run at a resolution of 1680 x 970.

The system was tested on a black oil reservoir simulation model with different discretizations, ranging from 10 to 60 million cells in total. One limitation of our system is the memory consumption of the model partitioning phase, requiring a PC with enough memory to load the entire model. Since our master node has 8 GB of memory, the partitioning had to be done on a separate computer.

5.1. System Scalability

In order to test the system scalability, we have varied both the size of the model and the number of cluster renderer nodes in use. The test was run using only 2 GPUs at each node. Figure 8 illustrates the results obtained using the gigabit Ethernet network and the Infiniband network. The plots display the time spent for rendering each frame along a camera path. In this path, the original model is rotated until $t = 45s$. At this time, the layers of the model are separated, increasing the rendering load. The model is then rotated until $t = 90s$, when a navigation is performed to inspect the reservoir model from a closer view. As can be noted, the system scales reasonably well for both interconnection configurations, presenting good overall performance.

Note that the use of a slower interconnect does not affect performance for these tests. This is because the network transmission is not the bottleneck of the system in these cases. However, if compression is turned off, network transmission becomes the bottleneck, degrading performance even for the faster interconnect.

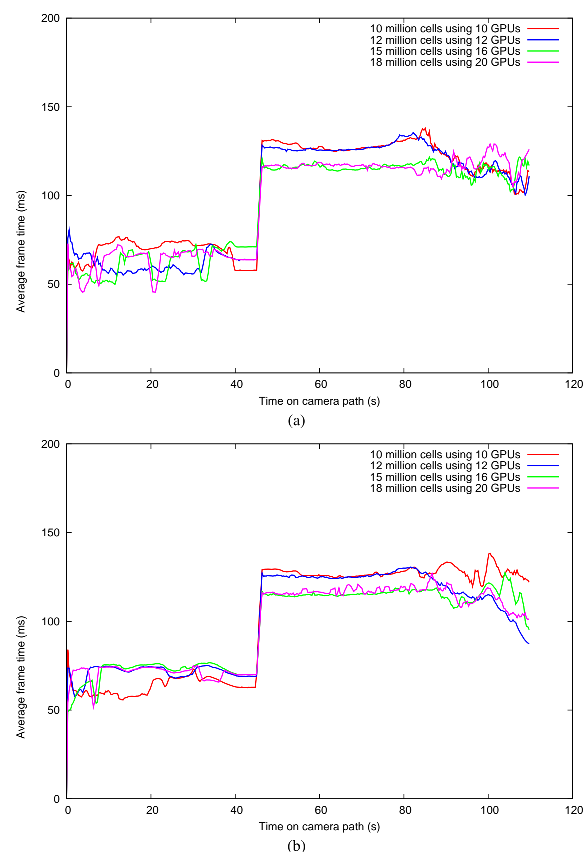

Figure 8: *System scalability using (a) the Ethernet network (b) the Infiniband network.*

© The Eurographics Association 2009.

5.2. Partial composition

As each renderer node was assigned a predefined part of the model, the system may suffer from load imbalance when zooming in the model. Additionally, in such situations, sub-models assigned to some renderer nodes may cover larger areas of the screen, requesting more network bandwidth and more composition efforts.

Under such circumstances, the proposed partial image composition on each renderer nodes has shown to be quite effective. This is demonstrated by testing the system with a different camera path. This path starts visualizing the entire model without layer separation. The camera zooms in until $t = 30s$ and returns to fit the whole model in the screen at $t = 60s$. At $t = 65s$, the reservoir layers are separated, and a navigation to inspect the model in between the layers is done until $t = 115s$. After that, the viewer returns to its original position.

This test was performed using 12 renderer nodes, using 4 GPUs at each, interconnected by the Ethernet network. Figure 9 illustrates the gain in performance due to applying partial composition on each node, especially for the second half of the experiment where performance is critical due to the layer separation.

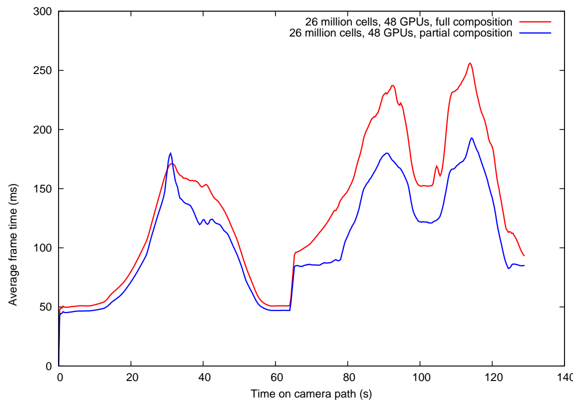

Figure 9: *Comparison between full and partial composition.*

5.3. Final image splitting test

We have used the same camera path of the last experiment to test the benefits of splitting the final image, parallelizing the read, compress, and send stages on the master node, and the receive, decompress, and compose stages on the client node.

The test was run using 11 4-GPU renderer nodes connected through the Infiniband network. We then varied the number of tiles for transferring the final image from the master to the client. As shown in Figure 10, we have achieved a better gain in performance when splitting the image in 4 tiles.

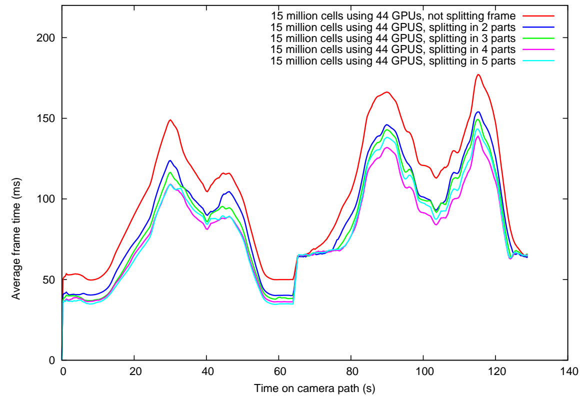

Figure 10: *Final image splitting comparison.*

5.4. Stressing the system

In order to stress the system, we have increased the size of the model to 60 million cells. Figure 11 illustrates the performance achieved by using a total number of 59 GPUs distributed among 15 renderer nodes, interconnected by the Ethernet network. In this test, the camera performed the same path as in the last two experiments.

6. Conclusions and Future Works

In this paper, we have presented a sort-last distributed rendering system for very large black oil reservoir models, which represents a demand of the oil industry due to the increasing use of parallel processing for reservoir simulations. Our system relies on clusters of PC equipped with multiple GPUs and, through the use of a pipelined implementation, is able to deliver fullscreen frames at interactive rates.

Experimental results have demonstrated the effectiveness

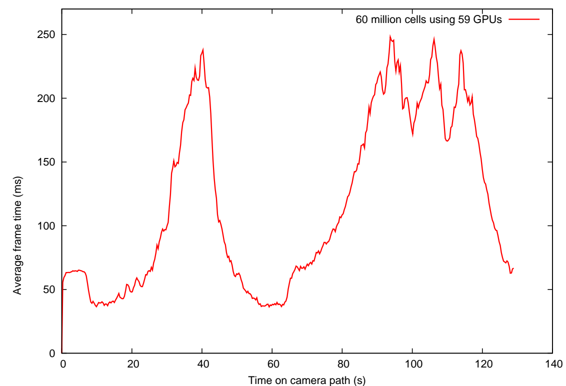

Figure 11: *System performance on the visualization of 60 million cells.*

© The Eurographics Association 2009.

and efficiency of the proposed solution. We highlight the following features of our system:

- it scales well with the size of the model and is able to handle models with tens of millions of cells;
- it allows the connection of lightweight clients, and we have proposed to parallelize the transferring of the final image by decomposing it into horizontal tiles;
- it takes advantage of nodes equipped with multiple GPUs, and we have proposed to perform partial image composition on each node.

We also presented a simple strategy for partitioning the model that results in a well balanced distribution. However, our current implementation requires an amount of memory proportional to the number of cells in the model. This may be prohibitive for very large models, and we plan to investigate a scalable partitioning algorithm in future work. We also plan to experiment other compression algorithms, since the run-length encoding type may not be the best choice for scientific visualization imagery. Another very important aspect that should be considered is load balancing; we plan to investigate dynamic load balancing based on rendering times, as in [ACCE04].

7. Acknowledgements

This research was financially supported by CNPq (Brazilian National Research and Development Council) and Petrobras (Brazilian oil company). We also thank NCSA (National Center for Supercomputing Applications) at the University of Illinois for the support and access to the cluster equipment.

References

[ACCE04] ABRAHAM F. R., CELES W., CERQUEIRA R., ELIAS J. L.: A Load-balancing Strategy for Sort-First Distributed Rendering. In *Proceedings of SIBGRAPI* (2004), IEEE Computer Society, pp. 292–299.

[AM00] ASSARSSON U., MÖLLER T.: Optimized View Frustum Culling Algorithms for Bounding Boxes. *Journal of Graphics Tools 5*, 1 (2000), 9–22.

[Bli96] BLINN J.: Calculating Screen Coverage. *IEEE Computer Graphics and Applications 16*, 3 (1996), 84–88.

[BWPP04] BITTNER J., WIMMER M., PIRINGER H., PURGATHOFER W.: Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful. In *Eurographics* (2004), vol. 23, pp. 615–624.

[CM06] CAVIN X., MION C.: Pipelined Sort-Last Rendering: Scalability, Performance and Beyond. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV06)* (2006), Eurographics Association.

[CMF05] CAVIN X., MION C., FILBOIS A.: COTS Cluster-based Sort-Last Rendering: Performance Evaluation and Pipelined Implementation. In *Proceedings of the IEEE Visualization Conference* (2005), IEEE Computer Society, pp. 111–118.

[Dak78] DAKE L. P.: *Fundamentals of Reservoir Engineering*. Elsevier Science, 1978.

[EC05] ESPINHA R., CELES W.: High-Quality Hardware-Based Ray-Casting Volume Rendering Using Partial Pre-Integration. In *Proceedings of SIBGRAPI* (2005), IEEE Computer Society, pp. 273–280.

[HEB*01] HUMPHREYS G., ELDRIDGE M., BUCK I., STOLL G., EVERETT M., HANRAHAN P.: WireGL: A Scalable Graphics System for Clusters. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 129–140.

[HHN*02] HUMPHREYS G., HOUSTON M., NG R., FRANK R., AHERN S., KIRCHNER P. D., KLOSOWSKI J. T.: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002), ACM Press, pp. 693–702.

[MCEF94] MOLNAR S., COX M., ELLSWORTH D., FUCHS H.: A Sorting Classification of Parallel Rendering. *IEEE Computer Graphics & Applications 14*, 4 (1994), 23–32.

[MMD08] MARCHESIN S., MONGENET C., DISCHLER J.: Multi-GPU Sort-Last Volume Visualization. In *Eurographics Symposium on Parallel Graphics and Visualization (EGPGV08)* (2008), Eurographics Association, pp. 1–8.

[nvia] http://www.nvidia.com/page/quadroplex.html.

[nvib] http://www.slizone.com.

[sau] Saudi Aramco Completes First Giga-Cell Reservoir Simulation Run. http://www.rigzone.com/news/article.asp?a_id=70015.

[dSRG*02] VAN DER SCHAAF T., RENAMBOT L., GERMANS D., SPOELDER H., BAL H.: Retained Mode Parallel Rendering for Scalable Tiled Displays. In *Proceedings of the 7th Immersive Projection Technology Symposium* (Orlando, Florida, 2002).

[SEP*08] SHOWERMAN M., ENOS J., PANT A., KINDRATENKO V., STEFFEN C., PENNINGTON R., MEI HWU W.: *QP: A Heterogeneous Multi-Accelerator Cluster*. Tech. rep., NCSA, University of Illinois at Urbana-Champaign, 2008.

[WPLM01] WYLIE B., PAVLAKOS C., LEWIS V., MORELAND K.: Scalable Rendering on PC Clusters. *IEEE Computer Graphics and Applications 21*, 4 (2001), 62–70.

© The Eurographics Association 2009.