Available online at www.sciencedirect.com
SCIENCE DIRECT®
Parallel Computing 31 (2005) 205–219
PARALLEL COMPUTING
www.elsevier.com/locate/parco

Large volume visualization of compressed time-dependent datasets on GPU clusters

M. Strengert [a,*], M. Magallón [b], D. Weiskopf [a], Stefan Guthe [c], T. Ertl [a]

[a] *Institute of Visualization and Interactive Systems, University of Stuttgart, Universitätstrasse 38, D-70569 Stuttgart, Germany*
[b] *Universidad de Costa Rica Escuela de Fisica, San Pedro de Montes de Oca 2060, Costa Rica*
[c] *WSI/GRIS, University of Tübingen, 72076 Tübingen, Germany*

Received 16 February 2004; revised 20 December 2004
Available online 14 April 2005

Abstract

We describe a system for the texture-based direct volume visualization of large data sets on a PC cluster equipped with GPUs. The data is partitioned into volume bricks in object space, and the intermediate images are combined to a final picture in a sort-last approach. Hierarchical wavelet compression is applied to increase the effective size of volumes that can be handled. An adaptive rendering mechanism takes into account the viewing parameters and the properties of the data set to adjust the texture resolution and number of slices. We discuss the specific issues of this adaptive and hierarchical approach in the context of a distributed memory architecture and present corresponding solutions. Furthermore, our compositing scheme takes into account the footprints of volume bricks to minimize the costs for reading from framebuffer, network communication, and blending. A detailed performance analysis is provided for several network, CPU, and GPU architectures—and scaling characteristics of the parallel system are discussed. For example, our tests on a eight-node AMD64 cluster with InfiniBand show a rendering speed of 6 frames per second for a $2048 \times 1024 \times 1878$ data set on a 1024^2 viewport.

* Corresponding author.
E-mail address: magnus.strengert@informatik.uni-stuttgart.de (M. Strengert).

0167-8191/$ - see front matter
doi:10.1016/j.parco.2005.02.006

Keywords: Graphics systems; Distributed/network graphics; Viewing algorithms

1. Introduction

Volume rendering is often to be applied to large data sets. For example, the increasing resolution of medical CT scanners leads to increasing sizes of scalar data sets, which can be in the range of gigabytes. Even more challenging is the visualization of time-dependent CFD simulation data, which can comprise several gigabytes for a single time step and several hundred or thousand time steps. Parallel visualization can be used to address the issues of large data processing in two ways: both the available memory and the visualization performance are scaled by the number of nodes in a cluster computer.

This paper is an extension to our previous work [1], which describes a combination of the "traditional" benefits of parallel computing with the high performance that is offered by GPU-based techniques. The basic idea is to apply hierarchical wavelet compression in the context of distributed volume visualization. In doing so, the effective size of the volume processed in a distributed memory architecture can be further increased. In addition an adaptive, texture-based rendering algorithm is presented and several optimizations for accelerating performance critical tasks such as framebuffer readbacks, image compositing, and network communication are discussed. This work is further extended by including the following topics. First, an extended approach for CPU-based image compositing using the capabilities of current 64-bit architectures is presented. Second, a detailed look into the system's performance with respect to time-dependent datasets is given. Third, we discuss the influence of the interconnecting network, on systems built upon Gigabit, Myrinet, and InfiniBand networks. Results are discussed for three different systems: a mid-price system with 16 GPU/dual-CPU nodes and Myrinet, a low-cost system with standard PCs connected by Gigabit Ethernet, and another mid-price system with 8 GPU/dual-AMD64-CPU nodes connected though InfiniBand. We believe that our findings are useful for working groups that have to visualize large-scale volume data.

2. Previous work

This work builds up on that of Guthe et al. [2], who represent a volumetric data set as an octree of cubic blocks to which a wavelet filter has been applied. By recursively applying this filter, a hierarchical multi-resolution structure is generated. Rendering is accomplished by computing a quality factor to select for which block the higher or lower resolution representations should be used. The decompression of the texture data is performed by the CPU. Based on this compression approach, Wang et al. [3] independently developed a parallel volume rendering system specially focused on large datasets and load balancing. Binotto et al. [4] presented a system that also uses a hierarchical representation, but is oriented towards the compression

of time-dependent, highly sparse and temporally coherent data sets. Their algorithm uses fragment programs to perform the decompression of the data sets, with a reported performance of over 4 fps for an image size of 512^2 pixels and a texture data set of 128^3 voxels.

Rosa et al. [5] described a system specifically developed for the visualization of time-varying volume data from thermal flow simulations for vehicle cabin and ventilation design. The system is based on the work of Lum et al. [6], which quantizes and lossily compresses the texture data by means of a discrete cosine transformation and stores the result as indexed textures. Textures represented in this way can be decoded in graphics hardware by just changing the texture palette. The disadvantage of this method is that support for paletted textures is being phased out by hardware vendors. This could be replaced by dependent texture look-ups, but these have a different behavior with respect to interpolation of the fetched data. In comparison to the other methods mentioned before, this approach achieves much lower compression ratios.

Stompel et al. [7] have recently proposed a compositing algorithm that takes advantage of the fact that in a configuration of n processing elements, there are on average $n^{\frac{1}{3}}$ partial images which are relevant for any given pixel of the final image. They report promising results, using a 100 Mbps Ethernet network as the underlying communications fabric. The efficiency of the algorithm is highly dependent on the viewing direction, but it compares favorably to the direct send and binary swap algorithms, which are commonly used for this task.

3. Distributed visualization

We use a *sort-last* [8] strategy to distribute the visualization process in a cluster environment. With increasing size of the input data set, this sorting scheme is favorable, since the input data becomes larger than the compositing data and hence a static partitioning in object space avoids communication regarding the scalar field during runtime. The basic structure of our implementation follows the approach by Magallón et al. [9].

During a preprocessing step object-based partitioning is performed to split the input data set into multiple, identically sized sub-volumes, depending on the number of nodes in the cluster configuration. To overcome possible memory limitations in connection with large data sets, this step is executed using the same set of nodes as the following render process. Once all sub-volumes are created and transferred to their corresponding nodes, the render loop is entered, which can be split into two consecutive tasks.

The first task is to render each brick separately on its corresponding node. An intermediate image is generated by texture-based direct volume visualization. We employ screen-aligned slices through a 3D texture with back-to-front ordering [10,11]. By adapting the model-view matrix for each node, it is assured that each sub-volume is rendered at its correct position in image space. Since the partitioning is performed in object space, the rendering process of different nodes can produce

output that overlaps each other in image space. The second task blends the intermediate images and takes into account that multiple nodes can contribute to a single pixel in the final image. The distributed images are depth sorted and processed through a compositing step based on alpha blending. To this end, each node reads back its framebuffer, including the alpha channel, and sends it to other nodes.

The original implementation by Magallón et al. takes advantage of all nodes for the computationally expensive alpha blending by using direct send communication scheme [12]. Each intermediate result is horizontally cut into a number of stripes matching the total number of nodes. All these regions are sorted and transferred between the nodes in a way that each node receives all stripes of a specific area in the image space. Then each node computes an identically sized part of the final image. In Section 4 this scheme is extended to reduce unnecessary transfer and blending of intermediate data.

The alpha blending of the intermediate images is completely performed on the CPU. Although the GPU is highly specialized for this task, the additional costs for loading all stripes into texture memory and reading back the information after blending would lead to a lower overall performance. The blending of pixel a onto pixel b is given by the equation

$$r = a + (1 - a_{\text{alpha}}) * b. \tag{1}$$

Since graphics hardware usually deals with 8 bit per color channel the result of the blending can be calculated using integer-arithmetic instead of the more costly floating-point arithmetic. By scaling all variables from $[0,1]$ to $[0,255]$, Eq. 1 can be rewritten as

$$r = a + \frac{(255 - a_{\text{alpha}}) * b}{255}. \tag{2}$$

The most expensive operation in terms of computational speed in Eq. 2 is the integer division by 255. With the substitution,

$$\frac{x}{255} = \frac{x + 128 + \frac{x + 128}{256}}{256},$$

the calculation of the division can be replaced by fast bit-shift operations. This expression is correct for all variables the range 0.255 when compared to the floating-point version rounded up and truncated to integer results. The resulting blending function is

$$\begin{aligned} x &= (255 - a_{\text{alpha}}) * \ b + 128, \\ r &= a + ((x + (x \gg 8)) \gg 8). \end{aligned} \tag{3}$$

To blend two RGBA pixels these equation have to be calculated four times, once for each color channel. With MMX operations [13] working on 32-bit integers it is possible to determine the result of the blend function for all four channels of a pair of pixel in parallel. A further extension to this is the simultaneous handling of two com-

Table 1
Blended pixels in million pixels per second

	Floating-point arithmetic (Eq. (1))	Integer arithmetic (Eq. (2))	Optimized integer arithmetic (Eq. (3))		MMX implementation (Appendix A)	
AMD Athlon XP 1800+	5.3	7.5	23.4	(3.12)	47.8	(6.37)
Intel Pentium IV 1.8	5.1	8.6	10.7	(1.24)	64.1	(7.45)
AMD Athlon MP+ 2000+	6.0	8.3	32.8	(3.95)	64.3	(7.74)
Intel Pentium IV 2.8	8.1	14.4	17.5	(1.21)	116.6	(8.09)
AMD Opteron 2.2	16.7	23.0	65.6	(2.85)	123.7	(5.37)

The program was compiled using gcc version 3.3.3.

pletely independent pairs of pixels at the same time using the capabilities of modern 64-bit architectures like the AMD Opteron CPUs. The actual implementation for an AMD64 system is given in Appendix A. Table 1 provides a performance comparison of all presented blending functions and the corresponding speedup is given with respect to the unoptimized integer arithmetic approach. In the best case two images of size 1280×1024 can be composited 95 times in one second on the tested single-CPU AMD Opteron system.

Without major changes this approach can also handle time-dependent scalar fields. During the bricking process a static partitioning scheme is used for all time steps, i.e., each sub-volume contains the complete temporal sequence for the corresponding part of the input volume. To synchronize all nodes the information regarding the current time step is broadcast to the render nodes.

4. Accelerated compositing scheme

Concerning distributed rendering the overall performance is limited by three factors: The process of reading back the results from the framebuffer, the data transfer between nodes, and the compositing step. In the following we address these issues by minimizing the amount of image data to be processed. The key observation is that the image footprint of a sub-volume usually covers only a fraction of the intermediate image. For the scaling behavior, it is important that the relative size of the footprint shrinks with increasing number of nodes. For simplicity, we determine an upper bound for the footprint by computing the axis-aligned bounding box of the projected sub-volume in image space. Since the time needed to read back a rectangular region from the framebuffer is nearly linearly dependent on the amount of data, reducing the area to be retrieved leads to a performance increase of this part of the rendering process. Similarly, the communication speed also benefits from

the reduction of image data. The compositing step is accelerated by avoiding unnecessary alpha blending operations for image regions outside the footprints. Similarly to SLIC [7], a line-based compositing scheme is employed. For each line the span containing already blended data is tracked. Since the images are blended in the depth-sorted order of their corresponding volume blocks and all blocks together represent the convex shape of the unpartitioned volume, the tracked region always forms one segment instead of multiple separated spans. If a projected volume face is parallel to the image plane, the depth sort results in an ambiguous ordering that may break this property. In this case the topology is used to ensure the connectivity of the marked span. With this information the new image data of the next compositing step can be separated into a maximum number of three segments. Two segments contain pixels that map into the region outside the marked span. These pixels need no further processing and can be copied into the resulting image. The remaining segment maps into an area where already other color information resides and alpha blending has to be performed. An example of this procedure is given in Fig. 1. After one iteration the size of the span containing data needs to be updated and the next image stripe can be processed. In doing so only a minimal amount of blending operations for a given volume partitioning must be carried out.

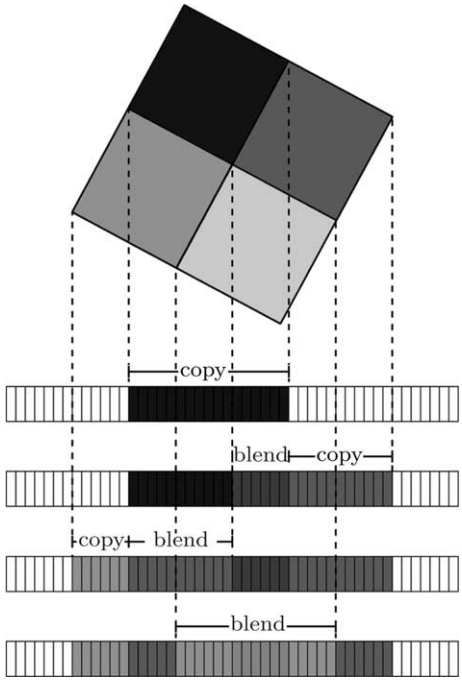

Fig. 1. Depth-sorted blending of footprints of four volume blocks. For each compositing step the regions with and without the need for blending are marked.

5. Hierarchical compression and adaptive rendering

Even with distributed rendering techniques the size of a data set can exceed the combined system memory of a cluster configuration and the already bricked data set is larger than one single node can handle. Another challenge is to further improve the rendering speed. We address the memory issue by using a hierarchical compression technique, and the performance issue by adaptive rendering.

5.1. Single-GPU wavelet compression

We adopt a single-GPU visualization approach that utilizes compression for large data sets [2]. The idea is to transform the input data set into a compressed hierarchical representation in a preprocessing step. An octree structure is created with the help of wavelet transformations. The input data set is split into cubes of size 15^3 voxels, which serve as starting point for the recursive preprocessing. Eight cubes sharing one corner are transformed at a time using linearly interpolating spline wavelets. The resulting low-pass filtered portion is a combined representation of the eight input cubes with half the resolution of the original data. The size of this portion is again 15^3 voxels. The wavelet coefficients representing the high frequencies replace the original data of the eight input blocks. After all cubes of the original data set are transformed, the next iteration starts using the newly created low-pass filtered cubes as input. The recursion stops as soon as the whole volume is represented through one single cube. This cube forms the root node of the hierarchical data structure and is the representation with the lowest quality. Except for the root node, all other nodes hold only high-pass filtered data, which is compressed through an arithmetic encoder [14]. While it is possible to increase the compression ratio by thresholding, we focus on lossless compression for best visualization results.

During rendering we use an adaptive decompression scheme that depends on the viewing position and the data set itself. Starting at the root node of the hierarchical data structure, a priority queue determines which parts of the volume are decompressed next. Depending on the ratio between the resolution of a volume block and the actual display resolution, regions closer to the viewer are more likely decompressed than others. Additionally an error criterion describing the difference between two representations of varying quality is used to identify regions that can be rendered in low quality without noticeable artifacts. After the quality classification is finished, all decompressed blocks are transferred to the graphics board's texture memory for rendering. Depending on the reconstructed quality level of a block, the number of slices used for rendering is determined. With increasing reconstruction quality the number of slices increases as well, delivering higher quality for areas closer to the viewer. Additionally a cache strategy is used to avoid the expensive decompression step for recently processed blocks. Unnecessary texture transfers are avoided by tracking the already loaded textures.

5.2. Extension to parallel rendering

In a distributed visualization system, this approach leads to a problem concerning correct texture interpolation between sub-volumes rendered on different nodes. A typical solution is to create the sub-volumes with an overlap of one voxel. With multi-resolution rendering techniques it is necessary to know not only the border voxels of the original data set but also the data value at the border of all other used quality levels [15]. This information can be determined in the preprocessing step. After creating the sub-volumes and constructing the hierarchical data structure, each node transfers the border information of all quality levels to its appropriate neighbors. But even with this information available on each node a correct texture interpolation cannot be generated easily. The remaining problem is to determine the quality level used for rendering of a neighboring node. This is necessary for choosing the correct border information of the previously transferred data. An example showing this problem is given in Fig. 2. Since communication between the nodes is costly due to network latency, requesting this information from the neighboring node is not suitable. Another approach is to compute the quality classification on each node for an expanded area. Unfortunately this is also impractical, because the quality classification is dependent on the volume data.

Instead, we propose an approximate solution that presumes that there are no changes in quality classification at the border of the sub-volumes. With this approach errors only occur if different qualities are used on each side of a sub-volume border (example visualization in Fig. 3). Due to the similar position of adjacent parts of the sub-volumes it is however likely that both regions are classified with the same quality. Experimental data showing the proportion of the error remaining under this presumption is given in Table 2 for both the unweighted number of transitions and for the area-weighted ratio. The measurement was performed while rendering the Visible Human data set on 16 rendering nodes (Fig. 4). In this configuration a total number of 185,212 cube transitions are present in the whole dataset. Considering only those transitions that lead to an interpolation error results in 723 cube borders, which is less than one percent of the total amount of transitions.

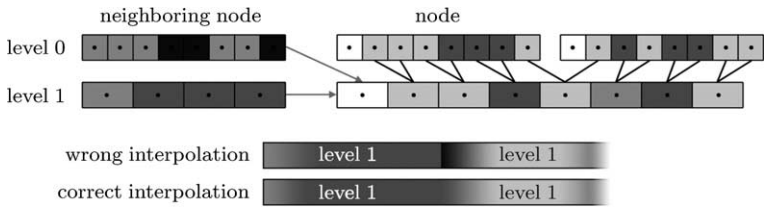

Fig. 2. Texture interpolation at a sub-volume border for a 1D case. Texels on one side of the border (white cells) are filled with previously transferred information of the neighboring node. Errors arise if the quality level of the neighboring node is unknown and hence a wrong level is chosen. For the incorrect, case border information of level 0 are used for interpolation, although the rendering of the neighboring node is performed on level 1.

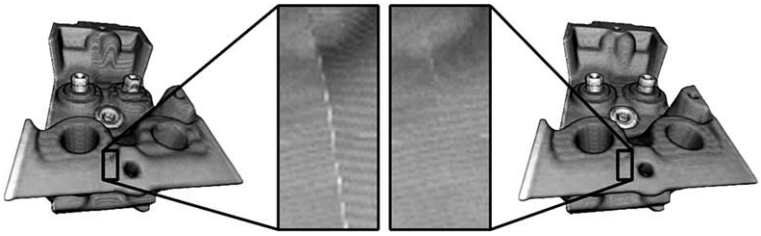

Fig. 3. In the left part of the image the volume was rendered using different quality levels for each of the two sub-volume blocks. Assuming identical classification for interpolation leads to visible artifacts as seen in the left magnified area. For comparison the right image was rendered with identical classification for the sub-volume blocks.

Table 2
Quantification of changes in quality classification at block faces

	Unweighted (%)	Area-weighted (%)
Total volume		
Same quality	89.8	81.7
Different quality	10.2	18.3
Sub-volume borders only		
Same quality	91.2	83.0
Different quality	8.8	17.0
Borders compared to total volume		
Same quality	99.6	99.1
Different quality	0.4	0.9

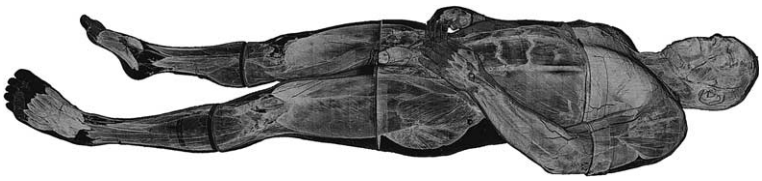

Fig. 4. Rendering result of the Visible Human Project male data set. The whole body is rendered on 16 nodes.

For a correct solution of the interpolation problem, we propose another approach that separates the computation of the quality classification and the rendering process. In each frame an adaptive classification is determined, but the associated rendering is delayed by one frame. In doing so, the information regarding the used quality levels can be transferred to the neighboring nodes at the time of distributing the intermediate results during the compositing step. Since at this time communication between all nodes must be performed anyway, the additional data can be appended to the image data. Having the transferred data available at the next frame

the rendering process can produce a properly interpolated visualization. The downside is that the latency between user interactions and the systems reaction is increased by one frame. To avoid this, a hybrid technique that exploits both described approaches is possible. While the viewing parameters are changed, the approximate solution is used to generate an image without increased latency times during user interaction. As soon as the camera parameters are kept constant, a correct image is rendered based on the quality classification that is transferred from the previous rendering step. Thus a fast user interaction is combined with a correct sub-volume interpolation for the static case.

6. Implementation and results

Our implementation is based on C++ and OpenGL. Volume rendering adopts post-shading realized either through NVIDIAs register combiners or alternatively through an ARB fragment program, depending on the available hardware support. MPI is used for all communication between nodes.

Three different cluster environments were used for developing and evaluation. The first one is a 16-node PC cluster. Each of these nodes runs a dual-CPU configuration with two AMD 1.6 GHz Athlon CPUs, 2 GB of system memory, and NVIDIA GeForce 4 Ti 4600 (128 MB) graphics boards. The interconnecting network is a Myrinet 1.28 GB/s switched LAN providing low latency times. Linux is used as operating system, the *SCore* MPI implementation drives the communication [16].

The second environment is built from standard PCs, using a Gigabit Ethernet interconnection with a maximum number of eight nodes. Each node has an Intel Pentium4 2.8 GHz CPU and 4 GB system memory. The installed graphics boards are a mixture of NVIDIA GeForce 4 Ti 4200 and GeForce 4 Ti 4600 both providing 128 MB of video memory. Running Linux, the MPI implementation *LAM/MPI* is used for node management and communication [17].

The last configuration consists of 8 PCs interconnected through an InfiniBand network. Each node is equipped with two AMD Opteron 248 CPUs clocked at 2.2 GHz, 4 GB of system memory and an NVIDIA Quadro FX 1100 (128 MB) graphics board. The 10 GB/s network devices are connected using the PCI-Express interface. Again Linux is used as operating system, and an InfiniBand-capable MPI implementation drives the communication.

We use various large-scale data sets to evaluate the performance of the implemented visualization system. The first static data set is an artificial scalar field showing a radial distance volume that is additionally combined with Perlin noise. For our testing purposes a 1024^3 sized volume is used. The second static data set is derived from the anatomical RGB cryosections of the Visible Human male data set [18]. The slices are reduced to 8 bit per voxel and cropped to exclude external data like gray scale cards and fiducial markers. The obtained data set has a resolution of $2048 \times 1024 \times 1878$ voxels (Fig. 4). For evaluating the systems performance with respect to time-dependent input data two additional data sets were selected. The first sequence was obtained from a CFD simulation of a flow field with increasing

turbulence. In total it consists of 89 individual time bins each 256^3 in size. Second, the IEEE Visualization 2004 contest data set showing a simulation of cloud movement during the hurricane Isabel in 2003 was used. Each time step of this simulation is $512 \times 512 \times 124$ in size and the complete sequence consists of 48 time steps resulting in a total data amount of 1.6 GB. Selected images of this data set are given in Fig. 5.

The Visible Human male data set can be visualized on a 1024^2 viewport using 16 nodes with 5.2 frames per second on our Myrinet-based cluster system. The quality classification was set to use the original resolution for most regions. Due to the uniform characteristic of the surroundings, these areas were displayed in a lower resolution without any noticeable disadvantages. With a viewport of half size in each dimension and the same settings the obtained framerate increases to 8.6 frames per second. Using the InfiniBand cluster system, the obtained results increased to 6.5 fps and 11.4 fps, depending on the viewport size. The increased performance is mainly the result of the faster image compositing possible in this cluster environment as well as the interconnecting network. The total amount of communication per frame is mainly determined by the image compositing step. The maximal amount of data to be transferred with a viewport of $M \times N$ pixels in a configuration with k nodes is (in bytes)

$$\text{vol} = 4 \ * (M * \ N) * (k - 1) * (1 + 1/k).$$

The constant factor of 4 is due to the need for sending RGBA images for correct image compositing. With a viewport of 1024^2 this leads to a data amount of 15 MB per frame. Using the Myrinet environment the maximal performance is solely limited by the communication to nearly 14 fps. Besides the communication bandwidth the network latency directly influences the system's performance. Using the Gigabit cluster environment with its eight nodes, only 2 frames per second are achieved for rendering the distance volume, while eight nodes of the Myrinet based cluster achieve with 4.3 frames per second. Due to the similar configuration of each node this gap is solely caused by the Gigabit Ethernet in comparison to Myrinet. While delivering comparable bandwidth, the Myrinet clearly outperforms a conventional Gigabit Ethernet regarding latency times. Switching to the InfiniBand architecture the same test results in 6.4 fps. Besides the faster image compositing the further optimized

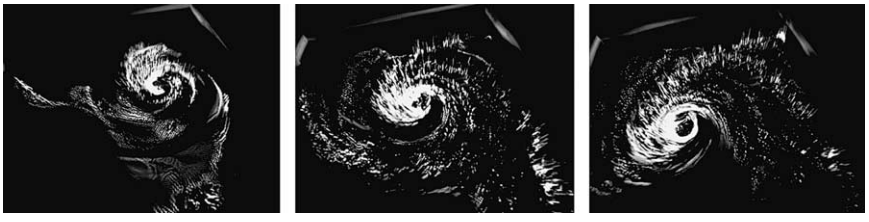

Fig. 5. Images of three different time steps of the IEEE Visualization 2004 Contest dataset. The content of the data shows a simulation of the cloud movement during the hurricane Isabel.

latency times on an InfiniBand architecture are the main reason for this performance increase. For a better comparison of the latency times, Fig. 6 shows the point-to-point performance for a typical range of packet sizes used in this application.

To show the scaling behavior of the visualization system configurations of 2 up to 16 render nodes were measured using the Myrinet based cluster environment. The data set for all these tests is the distorted radial distance volume with a size of 1 GB. Running on two nodes a new frame is rendered every 340 ms. Due to the limited amount of texture memory available using two 256 MB graphics boards, the data set cannot be reconstructed to the finest detail possible and is shown in a coarser representation. For all the following configurations the dataset is rendered in the original quality. Adding two more nodes reduces the time required per frame to 272 ms and with a total of eight nodes 233 ms are achieved. For a 16 node configuration the data set can be rendered in 174 ms, which corresponds to a refresh rate of 5.7 Hz.

For the time-dependent of the cloud simulation (Fig. 5) the whole sequence is rendered with an average of 3.3 fps on the InfiniBand cluster system, using a configuration that forces the original quality to be rendered in a 1024^2 viewport. For the CFD data set, Fig. 7 shows the results for rendering each time step in a row on the Myrinet cluster. The test was performed using three different quality levels. In case of the original quality the required time clearly increases towards the end of the sequence. The reason for this behavior is found in the characteristic of the data

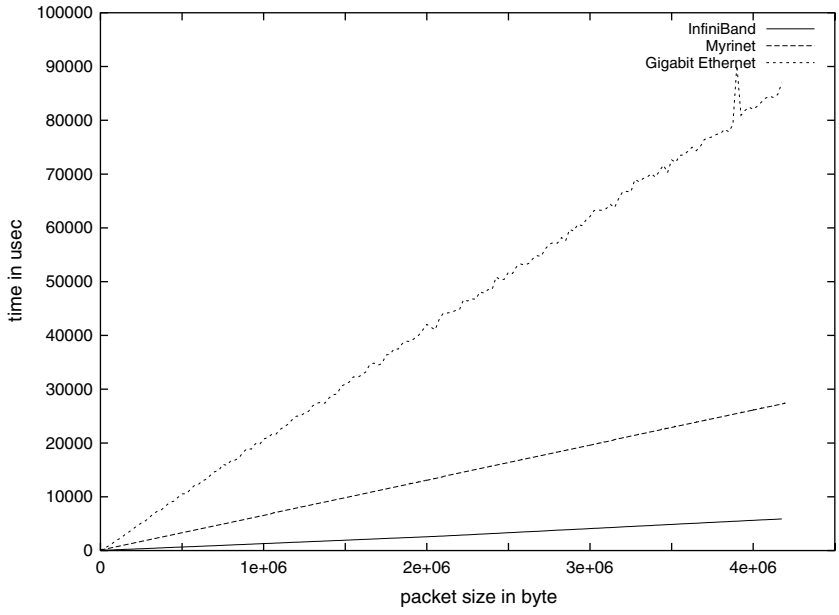

Fig. 6. Comparison of latency times for point-to-point communication using three different network architectures.

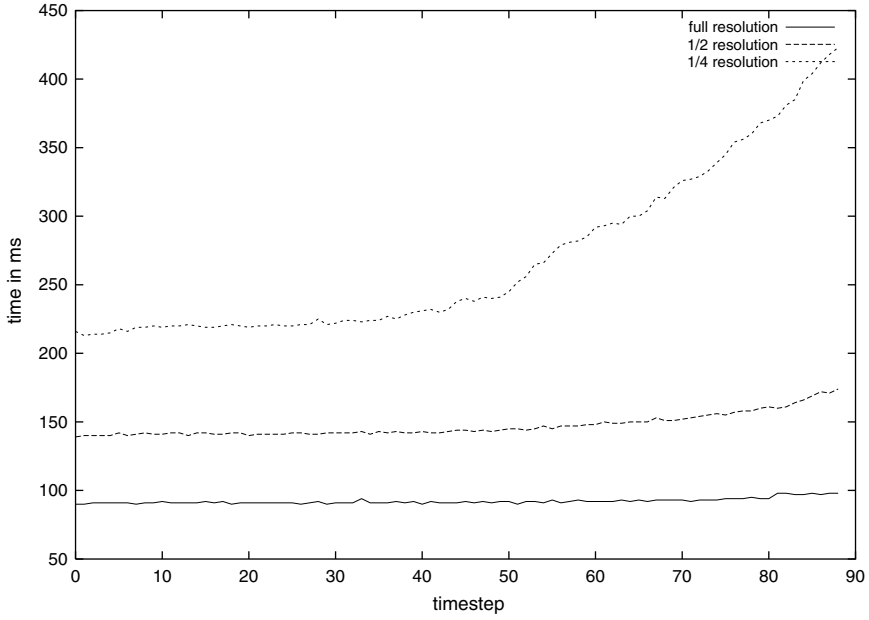

Fig. 7. Performance rendering time-dependent turbulence data set in different qualities.

set, which becomes more and more turbulent over time, leading to a higher number of blocks that have to be decompressed. In the case of time-dependent data sets the performance is mainly limited by the speed of decompressing. Stepping from one time bin to another cancels the possibility to make use of the current cached blocks and everything has to be decompressed again starting at the root node for each block. Therefore the performance is rather slow for time-dependent data sets compared to the static ones.

7. Conclusion and future work

We have presented a distributed rendering system for texture-based direct volume visualization. By adapting a hierarchical wavelet compression technique to a cluster environment the effective size of volume data that can be handled is further improved. The adaptive decompression and rendering scheme results in a reduction of rendering costs depending on the viewing positing and the characteristics of the data set without leading to noticeable artifacts in the final image. The problem of texture interpolation at brick borders in connection with multi-resolution rendering has been addressed and different solutions have been provided. Parts of the rendering process crucial to the systems performance benefit from the applied reduction of the processed region in image space, especially with increasing numbers of rendering nodes.

The achieved performance is often restricted by the capabilities of the interconnection between the rendering nodes and the computation of blending operations during the compositing step. We have measured performance characteristics for largely different network architectures and CPU/GPU combinations. With viewports sized 1024^2 the best performance is approximately 14 frames per second on our AMD64 cluster with InfiniBand.

To increase this upper limit an exact calculation of the footprints instead of using a bounding box could be helpful. Doing so avoids the remaining unnecessary blending operations and further reduces communication costs. In case of time-dependent data sets the performance is additionally bound by the decompression step because the performed caching of decompressed blocks cannot be used in this context. Future acceleration techniques of time-dependent data could be based on the coherence between time steps.

Appendix A. Blending using MMX operations on AMD64

The following code performs the operation $r = a + ((1 - a_{\text{alpha}})*b)/255$ for two pairs of pixels simultaneously. It uses 128-bit registers as found on recent AMD64 hardware. The code is written in the GNU Compiler Collection's (GCC) "extended assembly" notation, which means the operands are in AT&T syntax. `%0`, `%1` and `%2` are (r, r'), (a, a') and (b,b'), respectively. Each parameter is 64 bit in size, storing two non-interleaved RGBA pixels.

```
rex64 movd          (%1), %xmm0
rex64 movd          (%2), %xmm1

      // copy 128 (0x80) to all
      // words in x2
      mov     $0x00800080, %rax
      movd          %rax, %xmm2
      pshufd    $0, %xmm2, %xmm2

      // clear x3 and x4
      pxor          %xmm3, %xmm3
      pxor          %xmm4, %xmm4

      // prepare a pixels
      punpcklbw     %xmm3, %xmm0
      // prepare b pixels
      punpcklbw     %xmm3, %xmm1

      // fill x3 with 1's
      pcmpeqd       %xmm3,%xmm3
      // constuct 16-bit 255
      punpcklbw     %mm4, %xmm3
      // x3 = 1 -alpha
      pxor          %xmm0, %xmm3

      // 1-alpha on all words
      pshufhw   $0, %xmm3, %xmm3
      pshuflw   $0, %xmm3, %xmm3

      // x1 = (1-a)*b
      psmulw        %xmm3, %xmm1
      // x1 += 128
      paddusw       %xmm2, %xmm1
      // x2 = x1
      movdaq        %xmm1, %xmm2
      // x2 /= 256
      psrlw            $8, %xmm2
      // x1 += x1/256
      paddusw       %xmm2, %xmm1
      // x1 /= 256
      psrlw            $8, %xmm1

      // pack result
      packuswb      %xmm1, %xmm1
      packuswb      %xmm0, %xmm0

      // x0 += x1
      paddusb       %xmm1, %xmm0
rex64 moved         %xmm0, (%0)
```

References

[1] M. Strengert, M. Magallón, D. Weiskopf, S. Guthe, T. Ertl, Hierarchical visualization and compression of large volume datasets using GPU clusters, in: Eurographics Symposium on Parallel Graphics and Visualization (EGPGV04), Eurographics Association, 2004, pp. 41–48.
[2] S. Guthe, M. Wand, J. Gonser, W. Strasser, Interactive rendering of large volume data sets, in: Proceedings of the Conference on Visualization '02, 2002, pp. 53–60.
[3] C. Wang, J. Gao, H.-W. Shen, Parallel multiresolution volume rendering of large data sets with error-guided load balancing, in: Eurographics Symposium on Parallel Graphics and Visualization (EGPGV04), Eurographics Association, 2004, pp. 23–30.
[4] A.P.D. Binotto, J.L.D. Comba, C.M.D. Freitas, Real-time volume rendering of time-varying data using a fragment-shader compression approach, in: IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003, p. 10.
[5] G.G. Rosa, E.B. Lum, K.-L. Ma, K. Ono, An interactive volume visualization system for transient flow analysis, in: Proceedings of the 2003 Eurographics/IEEE TVCG Workshop on Volume graphics, 2003, pp. 137–144.
[6] E.B. Lum, K.-L. Ma, J. Clyne, A hardware-assisted scalable solution for interactive volume rendering of time-varying data, IEEE Transactions on Visualization and Computer Graphics 8 (3) (2002) 286–301.
[7] A. Stompel, K.-L. Ma, E.B. Lum, J.P. Ahrens, J. Patchett, SLIC: scheduled linear image compositing for parallel volume rendering, in: IEEE Symposium on Parallel and Large-Data Visualization and Graphics, 2003, pp. 33–40.
[8] S. Molnar, M. Cox, D. Ellsworth, H. Fuchs, A sorting classification of parallel rendering, IEEE Computer Graphics and Applications 14 (4) (1994) 23–32.
[9] M. Magallón, M. Hopf, T. Ertl, Parallel volume rendering using PC graphics hardware, in: Pacific Graphics, 2001, pp. 384–389.
[10] B. Cabral, N. Cam, J. Foran, Accelerated volume rendering and tomographic reconstruction using texture mapping hardware, in: Proceedings of the 1994 Symposium on Volume Visualization, 1994, pp. 91–98.
[11] T. Cullip, U. Neumann, Accelerating volume reconstruction with 3d texture mapping hardware, Tech. Rep. TR93-027, Department of Computer Science at the University of North Carolina, Chapel Hill, 1993.
[12] U. Neumann, Parallel volume rendering algorithm performance on mesh-connected multicomputers, in: IEEE/SIGGRAPH Parallel Rendering Symposium, 1993, pp. 97–104.
[13] A. Peleg, U. Weiser, MMX technology extension to the Intel architecture, IEEE Micro 16 (4) (1996) 42–50.
[14] S. Guthe, W. Strasser, Real-time decompression and visualization of animated volume data, in: Proceedings of the Conference on Visualization '01, 2001, pp. 349–356.
[15] M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, T. Ertl, Level-of-detail volume rendering via 3D textures, in: Volume Visualization and Graphics Symposium 2000, 2000, pp. 7–13.
[16] PC Cluster Consortium, Web page: http://www.pccluster.org/.
[17] LAM/MPI Parallel Computing, Web page: http://www.lam-mpi.org/.
[18] The National Library of Medicine's Visible Human Project, Web page: www.nlm.nih.gov/research/visible/.