Distributed Volume Visualization: A Step Towards Integrated Data Analysis and Image Synthesis

Rüdiger Westermann and Thomas Ertl

Computer Graphics Group, University of Erlangen-Nuremberg
Erlangen, Germany

Integrated on-the-fly data analysis and image synthesis is one of the most dominant challenges on modern volume visualization tools. Since volume visualization algorithms are them self computationally complex and memory intensive there is hardly a chance to efficiently integrate data analysis tools on standard single processor architectures. The development and spreading of multiprocessor systems with large scale memory allow for the direct analysis and modification of the data during the rendering process. But to efficiently integrate both tasks similar parallelization strategies and a unique data layout must be chosen. Furthermore, since more and more systems with hardware assisted processing and visualization options can be accessed across high performance networks, distributed visualization tools should benefit from these options whenever possible. In the following paper a prototyped parallel visualization environment will be exemplified in which data analysis and image synthesis are simultaneously performed on demand. The objective is to outline basic parallelization and distribution aspects which enable flexible integration of different tasks rather than to present a concrete implementation. In this context the development and design of a unique algorithmic framework for integrated and/or distributed data analysis and image synthesis dominates the work hereafter.

1 Introduction

Although volume rendering has become fairly standard in a wide area of different disciplines like medical imaging, physical simulation, or the generation of photo realistic images, not that much work has been spent on the analysis of the data so far. Analyzing the data is mostly accomplished by repeatedly analyzing the generated two dimensional images. In the majority of applications the examination of the three dimensional information content is performed in a pre-processing step once before the visualization takes place.

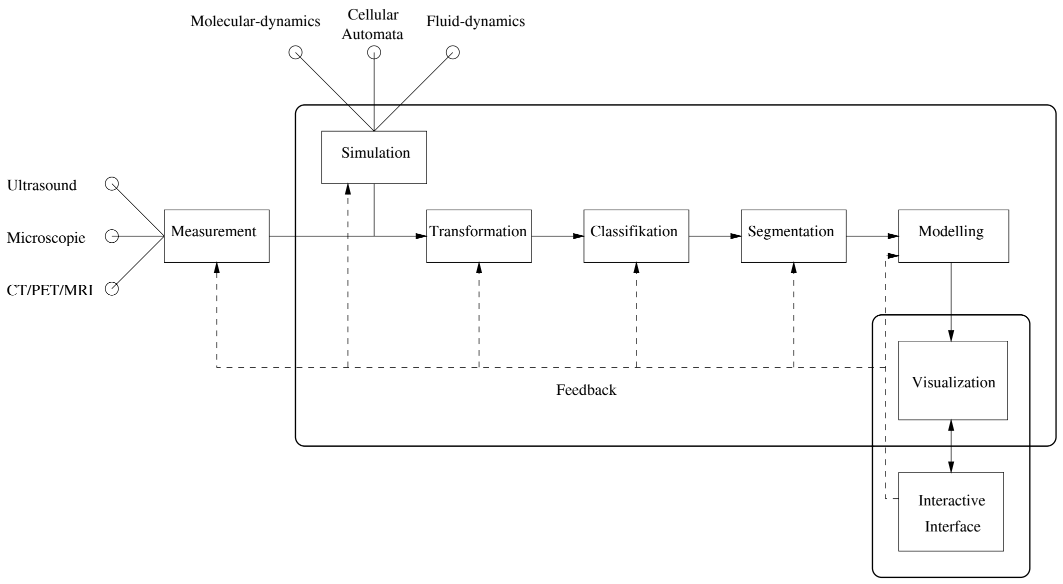


Fig. 1. Volume visualization pipeline.

This is due to the numerical complexity of the available data analysis techniques which in most cases also result in increasing memory requirements during run-time. As a matter of fact, putting both techniques together seems to be a problem that can not be solved adequately, whereas the problems are intensified due to the complexity of traditional imaging techniques.

On the other hand, over the last years different approaches have been investigated to accelerate volume rendering on single processor architectures. Recently, object space methods which allow for almost interactive speed like hierarchical splatting [29], Fourier rendering [19], or Shear-Warp Factorization on single processor machines [11] influenced the volume rendering community in a revolutionary manner. Even more the development of specialized graphics hardware enabling 3D texture mapping in real time pushed the achievable frame rates up to a satisfying limit.

In this context the question how to directly integrate data analysis methods into the rendering process becomes a new dimension. One obvious question is in which way and at which stage of the visualization pipeline shown in Figure 1 multiprocessor systems should be integrated. Taking into account the speed of todays single processor implementations the question should be asked whether the development of specialized parallel volume rendering algorithms at their own sake really makes sense. Up to now the most known parallel implementations can hardly compete with fast single processor solutions, especially if we also compare the time needed to develop the algorithms and the cost of a massively parallel system. Furthermore, very often parallel implementations exhibit the same problems fast single processor solutions do, e.g. memory overhead, poor sampling quality, no higher order scattering events. Since these algorithms are mostly designed in a way that perfectly fits onto the available architecture they often fail to integrate other tasks which are based on different parallelization strategies into the computation process.

We draw conclusion from the above observations in that we focus on the design of algorithms which allow us to appropriately integrate different tasks into each other, instead of designing either the data analysis or the image synthesis part in a way that fits as optimal as possible onto a certain architecture. Above all, the major goal is to build upon a unique parallelization strategy and data layout which all involved algorithms should rely on. However, particularly in a distributed environment where arbitrary platforms are available and can be linked together across high speed networks, one dominant requirement is to benefit from these systems whenever they are available. Specific tasks should be performed on those architectures which solve the emerging problems as good as possible.

Actually, we focus on the processing of three dimensional orthogonal grids for which data analysis and image synthesis algorithms are available at this time. Of course, there are different other known topologies which can not be processed in this way, e.g. irregular or unstructured grids. In this case decomposition schemes, but above all efficient analysis and synthesis algorithms need to be developed in more detail.

In section 2 we will briefly outline some of the desirable features parallel implementations should benefit from. Section 3 describes two basic parallelization strategies for multi-dimensional data transforms. The three dimensional separable wavelet transform has been chosen as a prototype which exemplifies the basic strategies. In section 4 we will focus on the parallel volume rendering algorithm and we will outline how to design the algorithm in such a way that enables integrated data analysis on arbitrary platforms. Implementation details are given in section 5 and a short conclusion and outlook is presented in section 6.

2 Requirements

Since the future will bring more and more multiprocessor systems either as closed solutions or as loosely but with high bandwidth interconnected clusters, it will be important to develop parallel algorithms which fit onto the available architectures. On the other hand, the past has shown that most of the algorithms that where particularly designed for a special system layout failed to be flexible enough to integrate arbitrary tasks, and very often these implementations performed in a less optimal manner on other architectures. Consequently, since we aim to efficiently perform data analysis and image synthesis, data coherence should be retained where ever possible and the implementation should be rid of any architectural constraints. We propose involving as less as possible knowledge about the underlying target architecture in the design phase of the algorithms, while the use of coarse grain parallelization schemes together with explicit data domain knowledge will fit onto shared and distributed memory architectures equally well.

The hope is that we develop open implementations which can be extended by other algorithms and ported from one platform to another without the need to reiterate the whole design phase. This will be exemplified in the following chapter.

3 Parallel Volume Processing

In general, arbitrary data processing techniques can be applied to enhance the extraction process of the three dimensional information content of volumetric data. In the following we will focus on a particular kind of data transformation, the discrete wavelet transform. The motivation for studying wavelet techniques especially in the context of parallel volume processing is that the underlying algorithms are representative for a large variety of other methods. Furthermore, wavelet transforms define a theoretical framework for the detection, classification, and compression of signals. Consequently, the same method can be used to perform different tasks at once.

In three dimensions the wavelet transform (WT) expands any finite energy function $f(\bar{x}) \in L^2(\mathbb{R}^3)$ into a sequence of hierarchically ordered projection spaces [17,1,3,8]. In the approximation spaces the original function is represented with decreasing resolution while all the information that is lost is retained in the corresponding difference spaces at a certain scale. All vector spaces are build from dilated and scaled versions of a wavelet function $\Psi(\bar{x})$. In its continuous form the projections, namely the wavelet coefficients, are computed as inner products

$$\langle f, \Psi^s_{i,j,k} \rangle = \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} \int\limits_{-\infty}^{\infty} f(\bar{x}) \Psi^s_{i,j,k}(\bar{x}) dx\, dy\, dz, \tag{1}$$

where s and $i, j, k \in \mathbb{Z}$ are the scaling and translation parameters, respectively. The wavelets $\Psi^s_{i,j,k}(\bar{x})$ are constructed from one prototyped version $\Psi(\bar{x})$ by

$$\Psi^s_{i,j,k}(x) = 2^{3s/2} \Psi(2^s \bar{x} - (i, j, k)), \tag{2}$$

where we restrict to the dyadic scale 2^s in the actual implementation.

From the two-scale relation [1] between the wavelet and the scaling function an efficient computation scheme can be derived how to compute the continuous inner products [17]. It is performed recursively by applying finite impulse response filters h_k and g_k to the approximation of f at every scale. As a consequence, the resolution of the discrete data samples is decimated by a factor of $\frac{1}{2}$ at each scale.

As for the DFT or the DCT the original function samples can be completely reconstructed from the projected version.

One of the important advantages of the wavelet transform is that it is perfectly suited to localize spatial and frequency domain information. From the multi scale representation features can be extracted and classified according to their local characteristics [5,18]. Thus, since the volume rendering integral can also be solved directly over the decomposed domain [21,27], wavelet transforms can be used as a powerful tool for integrated data analysis and image synthesis. However, different other techniques might be useful and could be implemented. Here, the discrete wavelet transform should be seen as a placeholder for a variety of techniques that can be used. Particularly, the processing order and communication patterns occurring in parallelizing the discrete wavelet transform are similar to a number of other applications. Very often data analysis algorithms based on DFT, DCT, Neural Networks, multi modality Registration and Fusion, or order independent region growing algorithms can be performed based upon the same subdivision and distribution strategies, or at least can be organized in a quite similar way.

3.1 Parallel Wavelet Transform

If the n-dimensional wavelet transform is split into n one dimensional transforms the discrete convolution to perform the decomposition can be successively applied along different axis. While the first approach we present strongly depends on this observation the second one takes advantage of the spatial localization of the wavelet transform.

In the following the abstract target machine is assumed to have p processing units each provided with a certain amount of local memory. The communication bandwidth increases with the size of the blocks transferred. The resolution of the volume to be processed is a power of two in each direction and a multiple of the number of processing units. These conditions should streamline the basic algorithm and lead to optimal performance, but they can also be released without affecting the implementation drastically.

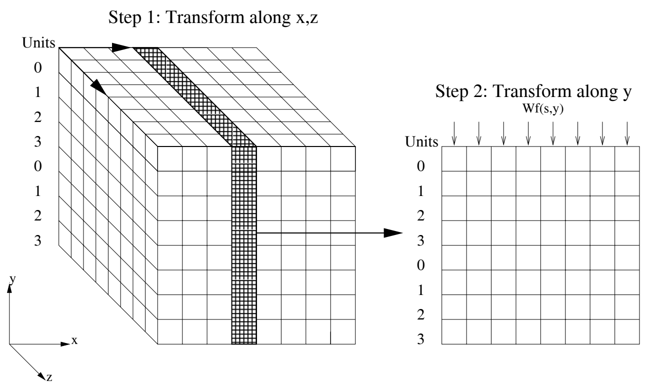


Fig. 2. Schematic representation of the domain subdivision and processing order.

3.2 Slice-wise Parallelization

Assuming a right handed coordinate system, the volume of resolution $(x \times y \times z)$ is divided into y slices each of resolution $(x \times z)$. On each unit p_k, $t = \frac{y}{p}$ slices $k, k+p, k+2p, ..., k+(t-1)p$ are stored. The wavelet transform can be performed along the x and z direction without additional communication and purely within the local memory domain of each unit (see Figure 2). To compute the convolution along the y direction all voxels which are not stored in local memory have to be fetched from other units. Programming this naively results in a huge amount of fine grain communications. Instead, if we try to change the data layout in such a way that the former y-axis is laid along the x- or z-axis, then also the final convolution step can be performed in local memory.

If a slice is cut out of the volume along the (x, y)-direction and is taken as a matrix in which the elements are the voxel values, then a transposition of the matrix organizes the data in the preferable way. As a result of the transposition all voxel values which were originally aligned along the y direction are now aligned along the x direction and vice versa. After applying the wavelet transform along the x-axis, a final transposition organizes the data in its original orientation.

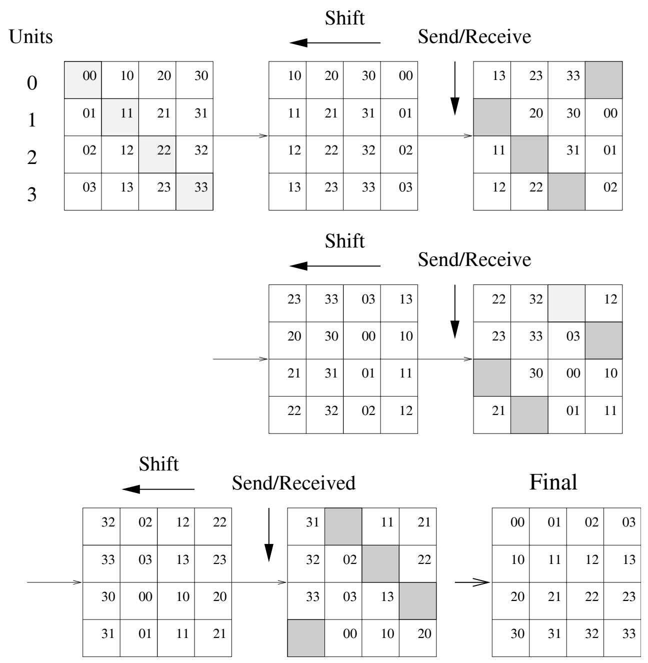


Fig. 3. Schematic representation of the volume transposition in two dimensions. Dark entries are already in place.

The transposition of the volume data can be accomplished by successively shifting the (x, z)-slices in local memory of each processing unit, k, around one position and transferring the shifted versions to the neighboring unit, $(k + 1)$ $mod\, p$. This step has to be repeated p times to obtain the transposed version. A schematic representation is sketched in Figure 3.

If the resolution of the data at a certain level is less than the number of processing units, then rather poor load balancing will be achieved since more and more units become idle.

Here the basic idea is to use a decomposition of the volume into equally spaced blocks which are distributed across the processing units. Due to the spatial localization of the wavelet transform it can be performed on each block separately. A slight communication overhead arises from the support of the used decomposition filters. If the support exceeds one, then additional data samples from neighboring nodes need to be fetched at the block boundaries.

After the transformation has been completed a local wavelet decomposition is generated for each block. Since it should be possible to locally reconstruct arbitrary points from within a certain block we need to store additional coefficients according to the support of the basis functions used (see Figure 4).

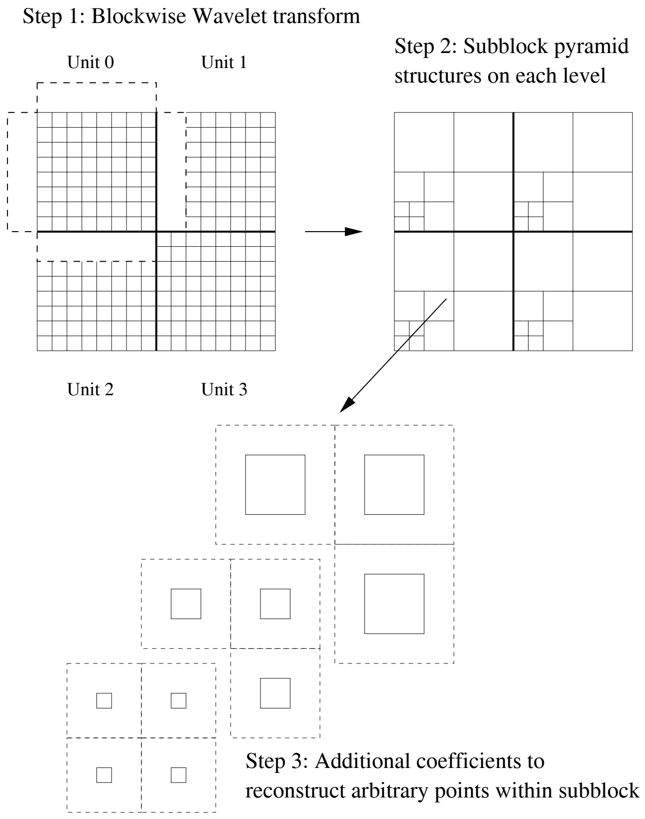

Step 1: Blockwise Wavelet transform
Unit 0
Unit 1
Unit 2
Unit 3
Step 2: Subblock pyramid structures on each level
Step 3: Additional coefficients to reconstruct arbitrary points within subblock

Fig. 4. Parallel wavelet transform based on a block-wise subdivision of the original data. The two dimensional case is supplied to streamline the basic algorithm.

On the other hand, since the support of the involved filters is small in practice, between 4 and 8 seems to be satisfactory, the overall amount of memory just increases insignificantly.

Regardless of the memory overhead each block can now be processed and visualized separately. All kinds of wavelet based feature extraction can be applied locally. Additionally, the wavelet representation can be stored in a compressed format at each node. In case of transmissions between processing nodes the communication load will be minimized. This is particular useful on loosely coupled workstation clusters where the communication load often slows done the overall performance.

Table 1 supplies the timing results for both parallelization strategies which have been implemented on a Connection Machine CM5 with 64 nodes. The former method seems to be better suited for parallelization purposes and probably would have been chosen as the favored method. Particularly, SIMD architectures like the CM2, MasPar MP1, or the Princeton Engine, which enable fast nearest neighbor communications would be well suited to perform this kind of parallelization. On the other hand, once the wavelet transform has been performed data coherence is completely lost, and the layout of wavelet coefficients across the units makes the local reconstruction of data samples on a certain node impossible. The integration of other data analysis techniques or even volume visualization techniques seems to be hard to achieve. In the contrary, the latter approach allows us to efficiently integrate all kinds of data processing techniques based upon a block decomposition of the underlying domain. Although it is the less optimal method in terms of performance it should obviously be used as the preferred one in an integrated environment. Additionally, this data layout offers the possibility to integrate arbitrary hardware options which in practice strictly depend on static block data structures.

In the next section we will show how to integrate the block-wise data layout into a parallel volume visualization algorithm. This algorithm was partly presented in [28]. Consequently only a short description will be given. Instead we will focus the discussion on how to take advantage of the data distribution strategies to integrate data analysis and image synthesis, and how to take advantage of specialized hardware acceleration techniques.

Table 1: Comparison between slice-wise and block-wise parallel wavelet transform

Resolution	32^3	64^3	128^3	256^3	512^3
Slice-wise	0.009	0.068	0.5	3.7	32.0
Block-wise	0.008	0.05	0.56	4.5	39.4

4 Parallel Volume Rendering

In direct volume visualization algorithms the amount of light impinging on the view plane at a certain position is simulated by evaluating the well known volume rendering integral [9,13,4]

$$I(t_0, t_1) = \int_{t_0}^{t_1} q(t) e^{-\int_{t_0}^{t} \alpha(s) ds} dt$$

along each ray of sight (see Figure 5). It sums up the contributions of the volume emission $q(t)$ along the ray, which is scaled by the optical depth according to the volume absorption $\alpha(s)$.

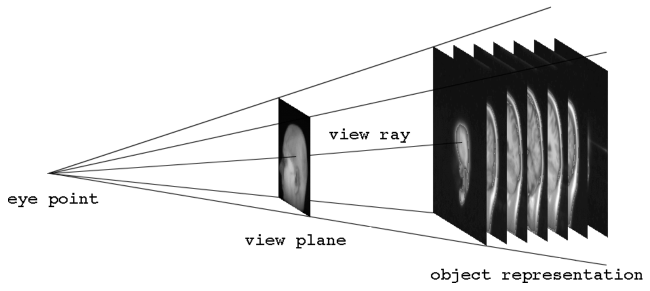


Fig. 5. Volume ray-casting.

Different approaches have been proposed by many authors to parallelize the integral evaluation. Basically, these approaches can be classified in two categories:
– object space driven
– image space driven

The former is based on a volume data transformation to align the volume coordinate axis with the view coordinate axis, while the latter re-samples the volume along the view rays.

Most often, methods have been developed which benefit from special features of the underlying target architecture. A common approach is to accomplish the volume alignment by a sequences of one dimensional shears [6,23,26,30] which can be efficiently mapped onto fast nearest neighbor communication primitives. Unfortunately, additional copies of the data have to be stored and data dependencies are completely lost during the shear steps. Taking this into account, it seems to be impossible to integrate any kind of data processing algorithms into these approaches.

Other approaches replicate the volume along processor clusters [20] which results in limited resolution of the volumes that can be processed, or, as it was proposed by Ma et al. [16], the data is subdivided into smaller blocks which are assigned to different nodes. Since each block is rendered separately neither acceleration techniques can be integrated nor does this strategy allow simulating higher order scattering effects.

The most powerful methods have been designed by Nieh et al. [22] which used a shared memory DASH architecture taking advantage of fast hardware assisted caching, and by Lacroute [11] who parallelized the shear-warp factorization algorithm. To increase the overall performance of both implementations explicit data domain knowledge should be integrated into the caching algorithms. Additionally, portability to other architectures and direct integration of parallel data processing techniques could be improved by defining a priori volume subdivision strategies.

4.1 Algorithm

In [28] a general approach to parallel volume rendering on distributed memory multiprocessor systems was proposed which avoids some of the disadvantages of the previously described methods. It is closely related to the one presented by Camahort [2], who first claimed that block decomposition schemes are the most flexible ones for integrated data analysis and rendering. Since the implementation is rid of any machine dependent features only less effort has to be spent to adapt the implementation onto other architectures. Furthermore, as we retain data coherence where ever possible, global communication during the rendering phase is minimized.

Above all, the demand to efficiently integrate data processing was one of the most dominant goals, and therefore the data layout accounts for this. It is thus the same block decomposition strategy we proposed for the block-wise parallel wavelet transform.

The rendering proceeds in traversing the volume on the coarse resolution which results from the initial volume subdivision. We take advantage of fast parametric clipping to find the intersection points with the blocks. Each time a hit is determined the block has to be fetched from other nodes if it does not exist in local memory. Since we also partitioned the pixel plane and rays belonging to the same partition are located on the same node, the rendering integral across the domain of a block can be performed for all rays in the partition which intersect the block. Finding the intersection points is numerically cheap, thus the overhead only arises from the global communication step. This was less than 10 % of the overall rendering time for all block sizes we tested.

To further reduce the idle times due to congestion a software cache data structure was implemented at each node. If multiple nodes request a block from the same node they probably become idle since the request can not be processed immediately. In this case the requesting block computes hits with the next blocks in advance, requests these blocks, and stores them in a dynamic cache list. Primary to each fetch operation the cache is inspected checking whether the block is stored in local memory. As a consequence, idle times can be minimized and hardware caching mechanisms can be further improved. For example, the use of this strategy on a shared memory system would help to improve the implementations scalability, and would also help to accelerate the rendering process if additional knowledge about the data layout is integrated into the computing order of the view rays.

4.2 Integrated Volume Processing

Block decomposition enables the straight forward integration of data processing techniques into the rendering process, since arbitrary operations can be performed on each block locally. Furthermore, a huge variety of data analysis techniques can be implemented upon a unique block based subdivision scheme. In many cases it suffices to store a limited number of additional slices around each block avoiding expensive global communication operations.

As we described earlier, the parallel wavelet transform was implemented in such a way that the whole wavelet representation of a certain block is stored at a single node. This allows performing a variety of useful but computational intensive procedures locally, such as **thresholding/compression, feature extraction/enhancement, and smoothing/sharpening**.

Thresholding and compression is accomplished by adaptively changing user supplied error tolerances. Filtering the wavelet coefficients according to

$$c_i := \begin{cases} 0 & : \quad |c_i|^2 \leq \epsilon \\ c_i & : \quad |c_i|^2 > \epsilon \end{cases}$$

allows one to directly investigate the correlation between error bounds and image quality. Taking advantage of data structures which store the sparse representations in a way that minimizes the memory requirement enables the rendering of large scale data sets which could have been processed neither. Additionally, the transmission overhead is reduced to some extent, which is particularly useful in distributed environments where remote clients are interconnected with the parallel system.

Since the wavelet coefficients include localized spatial and frequency information they can be used efficiently to improve the extraction of features from within the data [18,5]. As in [14], where the grey-scale gradient was used to detect and highlight region boundaries, the extracted information can be used in the same way, i.e. changing the transfer function accordingly.

Smoothing and sharpening the data is accomplished by discarding wavelet coefficients and by taking into account the difference information at every level. Arbitrary refinement strategies can be chosen and performed on each block individually.

4.3 Hardware Acceleration

Due to the fact that apparently more and more hardware options like signal processors, decompression/compression boards, or specialized accelerators for 2D or 3D graphics have been developed and brought to the market as extensions of available computer systems, one dominant goal should be to integrate these extensions whenever available. Even in a distributed environment where different systems are interconnected across high speed networks the integration of available special purpose architectures should be possible. The proposed decomposition strategy enables the use of a variety of different acceleration techniques which build upon block based data structures.

In our implementation a SGI graphics workstation was connected to a massively parallel system across a HIPPI interface. A parallel version of the marching cubes algorithm [15] to generate polygonal iso-surface representations was implemented. The generated polygon lists are transmitted over the

HIPPI interface to the graphics system were interactive rendering is achieved. Another choice is to transmit the rendered two dimensional images only, taking advantage of the high communication bandwidth. Additionally, we exploited the 3D texture mapping facility of the graphics workstation. Since for hardware assisted rendering of large scale volume data it has to be divided into bricks that fit into texture memory, a block decomposition based approach allows integrating both techniques without changing the general data layout. Finally, one could also take advantage of specialized hardware boards at the parallel site. Installing block structure based compression boards allows for interactive decomposition/composition and will speed up the transformation process to some extent. Even in an environment of a loosely coupled workstation cluster with low bandwidth this is of particular interest.

Basically, the implementation demonstrates the collaboration between numerical number crunchers, broadband communication channels, and special purpose accelerators. The strict block decomposition enables a flexible integration of different algorithms on the parallel system. At the same time it allows taking advantage of special purpose accelerators which are probably located at remote sites, but which are very often justified at the same block data structures.

5 Implementation Details – the Java Interface

The implementation was done on a Connection Machine CM5 with 64 Sparc 2 processors each of which was equipped with 32 MB of memory and 4 additional vector units. Object oriented programming is supported with C++ and the standard CM5 message passing library. Since the vector units can only be accessed in data parallel mode, on each node C* (data parallel C programming language) code segments were implemented to speed up the computation. Particularly the discrete convolution used to perform the data decomposition is well suited to be implemented within this programming model. The use of C* seems to be contradictory to the proposed method of being independent of a certain architecture, but once the basic algorithm has been designed and implemented on an abstract machine independent level we should of course try to adapt different tasks locally to the underlying system. This does not influence the portability to other platforms since in general only short stand alone code segments have to be exchanged. The whole programming layout was designed in a purely block-wise manner with the restriction to only a few standard calls like *CMMD-send-block()* and *CMMD-receive-block()*. In this way the whole program can be easily ported onto other machines without changing the basic layout. We recognize that an implementation based on PVM or MPI would have been the most general choice.

A user interface to control the action of the massively parallel computer has been implemented on the graphics workstation. To ensure architecture independence and easy integration of other tasks along with basic security for the server and the client, the Java programming language was chosen to implement both the client, running on the workstation, and the server, running on the front-end of the parallel machine. The server uses standard TCP/IP sockets to communicate with the parallel computer and the Java socket class to build up connections to the client. The client can upload volumetric data sets to the server for processing and can also use the server as a data pool where different example data sets are located and can be retrieved. In any case, the parallel site is completely responsible for the subdivision of the data set and the distribution of the generated blocks across the processing units. Upon completion of the computation, the server converts the generated data to a platform independent file-format in a space accessible to the client.

Actually, different processing and visualization tools are available. The parallel wavelet transform is generated on demand, thus smoothing, sharpening, compression, and feature enhancement can be applied. The parallel marching cubes algorithm generates polygon structures according to a user defined threshold and a user supplied resolution level. The generated triangle lists are transmitted separately from each node where they have been generated. Additionally, the parallel ray-casting method computes a two dimensional image. These methods can be selected by the client which is also responsible to display the computed and converted results.

Consequently, the only requirement for a workstation to use this prototype system is an internet browser capable of processing Java bytecode. For displaying three dimensional data an external 3D viewer (VRML-Viewer in our application) is required. A snapshot of an interactive session is shown in Figure 6.

6 Conclusion

A prototyped version of a visualization environment for distributed volume data analysis and image synthesis has been proposed. It enables integrating different techniques into each other and it is designed at an abstract, architecture independent level which is completely decoupled from the underlying platform. This allows easy adaption to different platforms, but provides enough flexibility to benefit from special purpose hardware accelerators whenever they are available. The proposed visualization tool incorporates some basic features which should be necessarily integrated into modern visualization environments, and enables on-the-fly data analysis and image synthesis. Different other techniques can be integrated into this environment quite easily. In this context the actual implementations outlines a further step into two basic future directions in scientific visualization: Interactive data analysis and image synthesis, and the connection of arbitrary architectures along with their integration to obtain optimal performance.

The most interesting question that remains to be answered is how to take advantage of specialized hardware options. This allows performing numerically and memory intensive algorithms on multiprocessor solutions which are provided with cheap additional hardware accelerators, for example, multiprocessor PC solutions. Additionally, the question in which way to integrate other grid topologies remains to be open at this time.

7 Acknowledgment

The work was partly done at the German National Institute for Computer Science in St. Augustin, Bonn. The authors would like to thank the TMC application engineer Peter Ossadnik for valuable discussions and help. Special thanks to Peter Kipfer who was responsible for the Java implementation.

References

[1] C. K. Chui. An Introduction to Wavelets. In *Wavelet Analysis and its Application*, Vol.1, Academic Press, 1992.

[2] E. Camahort and I. Chakravarty. Integrating Volume Data Analysis and Rendering on Distributed Memory Architectures. In *ACM Parallel Rendering Symposium*, Comference Proceedings, pp. 89-96, 1993.

[3] I. Daubechies. Ten Lectures on Wavelets. In *CBMS-NSF Series in Applied Mathematics*, Vol. 61, SIAM 1992.

[4] R. Drebin, L. Carpenter and P. Hanrahan. Volume Rendering. In *Computer Graphics*, Vol. 22, No. 4, pp. 65-74, 1988.

[5] B. Guo. A Multiscale Model for Structure Based Volume Rendering. In *IEEE Transactions on Visualization and Computer Graphics*, Vol.1, No. 4, pages 291-301, 1995.

[6] P. Hanrahan. Three-Pass Affine Transforms for Volume Rendering. In *Computer Graphics*, Vol. 24. No. 5, 1990.

[7] W.M. Hsu. Segmented Ray Casting for Data Parallel Volume Rendering. In *ACM Parallel Rendering Symposium 1993, Conference Proceedings*, pages 7-14, 1993. In *Wavelets: mathematics and applications*, edited by J. Benedetto and M. Frazier, pages 467-505, 1993.

[8] B. Jawerth and W. Sweldens. An Overview of Wavelet Based Multiresolution Analysis. In *SIAM Review*, Vol.36, No.3, 1994.

[9] W. Krueger. The Application of Transport Theory To Visualization of 3D Scalar Data Fields. In *IEEE Visualization 1990, Conference Proceedings*, pages 12–15, 1990.

[10] P. Lacroute and M. Levoy. Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation. In *ACM SIGGRAPH'94 Conference Proceedings*, pages 451-457, 1995.

[11] P. Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. In *ACM Parallel Rendering Symposium 1995, Conference Proceedings*, pages 15-22, 1995.

[12] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm. In *Computer Graphics*, Vol.25, No.4, pages 12–15, 1990.

[13] M. Levoy. Gaze-Directed Volume Rendering. In *Computer Graphics*, Vol.24. No.2, pages 285-288, 1991.

[14] M. Levoy. Display of Surfaces from Volume Data. In *IEEE Comput er Graphics and Applications*, Vol.8, No.3, pages 29-37, 1990.

[15] W.E. Lorensen and H.E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Computer Graphics*, Vol.21, No.4, pages 163-169, 1987.

[16] L-K. Ma, J. Painter, C. Hansen and M. Krogh. A Data Distributed, Parallel Algorithm for Ray-Traced Volume Rendering. In *ACM Parallel Rendering Symposium 1993, Conference Proceedings*, pages 15-22, 1993.

[17] S. Mallat. A Theory for Multiresolution Signal Decomposition: The Wavelet Representation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.11, No.6, pages 674-693, 1989.

[18] S. Mallat and S. Zhong. Characterization of Signals from Multiscale Edges. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol.14, No.7, pages 710-730, 1992.

[19] T. Malzbender. Fourier Domain Volume Rendering. In *ACM Transactions on Graphics*, Vol. 12, No. 3, pages 233-250, 1993.

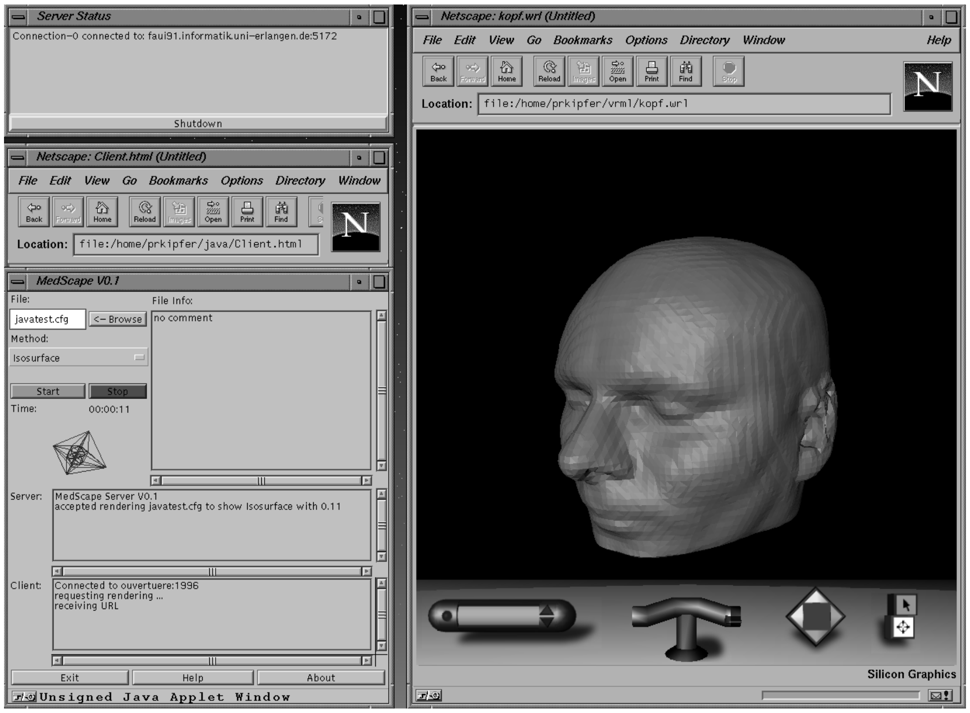
[20] C. Montani, R. Perego and R. Scopignio. Parallel Volume Visualization on a Hypercube Architecture. In *ACM Workshop on Volume Visualization 1992, Conference Proceedings*, pages 17-24, 1992.

[21] S. Muraki. Approximation and Rendering of Volume Data Using Wavelet Transforms. In *IEEE Computer Graphics and Applications*, Vol.13, No.4, pages 50-56, 1993.

[22] J. Nieh and M. Levoy. Volume Rendering on Scalable Shared-Memory MIMD Architectures. In *ACM Workshop on Volume Visualization 1992, Conference Proceedings*, pages 17-24, 1992.

[23] P. Schroeder and J.B. Salem. Fast Rotation of Volume Data on Data Parallel Architectures. In *IEEE Visualization 1991, Conference Proceedings*, pages 87-102, 1991.

[24] P. Schroeder and G. Stoll. Data Parallel Volume Rendering as Line Drawing. In *ACM Workshop on Volume Visualization 1992, Conference Proceedings*, pages 25-32, 1992.


File Edit View Go Bookmarks Options Directory Window Help
Back Forward Home Reload Images Open Print Find Stop
Shutdown
<- Browse
Start Stop
Exit Help About
Silicon Graphics

Server Status

Connection–0 connected to: faui91.informatik.uni–erlangen.de:5172

Netscape: Client.html (Untitled)

Location: file:/home/prkipfer/java/Client.html

MedScape V0.1

File: javatest.cfg

Method: Isosurface

Time: 00:00:11

File Info: no comment

Server: MedScape Server V0.1
accepted rendering javatest.cfg to show Isosurface with 0.11

Client: Connected to ouverture:1996
requesting rendering ...
receiving URL

Unsigned Java Applet Window

Netscape: kopf.wrl (Untitled)

Location: file:/home/prkipfer/vrml/kopf.wrl

Fig. 6. Java based interface and remote control. Top left corner shows the Java server that is connected to the client Java applet as shown below. The VRML viewer that is plugged in Netscape is shown on the right.

[25] T. Totsuka and M. Levoy. Frequency Domain Volume Rendering. In *Computer Graphics Proceedings, Annual Conference Series*, pp. 271- 278, 1993.

[26] G. Vezina, P. Fletcher and P. Robertson. Volume Rendering on the MasPar MP-1. In *ACM Workshop on Volume Visualization 1992, Conference Proceedings*, pages 3-8, 1992.

[27] R. Westermann. A Multiresolution Framework for Volume Rendering. In *ACM Workshop on Volume Visualization 1992, Conference Proceedings*, pages 51-58, 1994.

[28] R. Westermann. Parallel Volume Rendering. In *IEEE 9th International Parallel Processing Symposium, Conference Proceedings*, pages 693-700, 1995.

[29] L. Westover. Footprint Evaluation for Volume Rendering, In *Computer Graphics Proceedings, Annual Conference Series*, pages 185–188, 1990.

[30] C. Wittenbrink and A. Somani. Time and Space Optimal Data Parallel Volume Rendering Using Permutation Warping. To appear in *Journal of Parallel and Distributed Computing*, 1996.