PARALLEL VOLUME RENDERING FOR OCEAN VISUALIZATION IN A CLUSTER OF PCS

Alexandre Coelho,[1] Marcio Nascimento,[1] Cristiana Bentes,[1]
Maria Clicia Stelling de Castro,[2] and Ricardo Farias[3]

[1]*PGEC-Geomatica, Universidade do Estado do Rio de Janeiro (UERJ),*
Rua Sao Francisco Xavier 524, 5o andar, Bloco D, RJ, Brasil
{coelho,msnasc,cris}@eng.uerj.br

[2]*Instituto de Matematica e Estatistica, Universidade do Estado do Rio de Janeiro (UERJ),*
Rua Sao Francisco Xavier 524, 6o andar, Bloco D, RJ, Brasil
clicia@ime.uerj.br

[3]*COPPE-Sistemas, Universidade Federal do Rio de Janeiro (UFRJ),*
Cidade Universitaria, Centro de Tecnologia, Bloco H, RJ, Brasil
rfarias@cos.ufrj.br

Abstract

Volume rendering techniques can be very useful in geographical information systems to provide meaningful and visual information about the surface and the interior of 3D datasets. For ocean visualization, in particular, volume rendering techniques improve the analysis of the ocean inner structure, by generating visual information about, e.g., its temperature, salinity, velocity and mass. The rendering of huge datasets, however, is a computationally intensive task and, in order to achieve interactive visualization times, a high-performance computational system is fundamental. Although parallel machines have been successful in providing interactive times, most recent efforts have been directed towards a more cost-effective solution: implementing volume rendering algorithms on clusters of PCs. This platform has low-cost and can be easily upgraded. Parallel rendering applications, however, usually suffer from high load imbalance during the execution. In this paper, we propose a low-cost and high-performance system for ocean visualization in a cluster of PCs, DPZSweep. Our solution spreads the computation over the cluster and provides dynamic load balancing with a low overhead. Our experimental results show that when we included the load balancing algorithms, DPZSweep obtained up to 95% of parallel efficiency in 16 processors.

Keywords: Ocean Visualization, Parallel Rendering, Distributed Systems

1. Introduction

One of the most important fields of scientific visualization is volume visualization. Volume visualization is the process of generating meaningful and visual information onto a two-dimensional image plane from tri-dimensional datasets. It has been increasingly important in geographical information systems to improve the ocean modeling [Djurcilov et al., 2002, Gonzato and Saec, 2000], the monitoration of atmospheric pollution [de Oliveira and Ferreira, 1997], the visualization of meteorological data [Santos et al., 1996], the terrain modeling [Prakash and Kaufman, 1997], and the understanding of some natural phenomena like tropical cyclones [Watson et al., 2002].

However, most of traditional tools and techniques for volume visualization in geographical information systems allow researchers to explore only the surface of 3D datasets [Koller et al., 1995]. Direct volume rendering techniques, on the other hand, convey more information than surface rendering methods, enabling the viewer to fully reveal the internal structure of 3D data.

For the ocean modeling application, direct volume rendering techniques improve the analysis of the ocean inner structure. They allow the generation of visual information about, e.g., the ocean temperature, salinity, velocity and mass, providing more information than a set of cross-sections and planar maps. Nevertheless, direct volume rendering is notoriously a memory and computationally intensive task and, in order to achieve interactive visualization times, a high-performance computational system is fundamental.

Several parallel volume rendering algorithms were proposed in the literature, e.g. [Challinger, 1993, Hofsetz and Ma, 2000, Hong and Kaufman, 1998, Ma, 1995, Ma and Crockett, 1997], and they achieve quite good performance, running on expensive parallel machines like SGI Power Challenge, IBM SP2, or SGI Origin 2000. Recently, the decreasing cost and high availability of commodity PCs and network technologies turn clusters of PCs a low-cost alternative for running parallel rendering [Muraki et al., 2003, Samanta et al., 1999, Meiıner et al., 1998, Samanta et al., 2000].

In this paper, we propose a low-cost and high performance system for ocean visualization and realistic rendering. Our system was designed to run on a cluster of PCs, and to overcome the main problem this architecture imposes: the high communication overhead.

Our solution spreads the computation over the cluster of PCs and it provides dynamic load balancing with a low overhead. Our idea is to take advantage of some well-known distributed information diffusion algorithms to inform the PCs about the system load without using broadcast messages. In this way, we avoid the overload of load balancing messages.

Our parallel system, DPZSweep, is an all-software distributed version of the PZSweep system [Farias and Silva, 2001] that provides: efficiency, scalability,

portability and out-of-core execution. Our experimental results show that when we included the load balancing algorithms, we obtained up to 95% of parallel efficiency in 16 processors for the most overloaded dataset.

The remainder of this paper is organized as follows. The next section explains in more details the two different approaches to volume visualization: surface rendering and volume rendering. Section 3 shows the importance of visualizing oceanographic datasets. Section 4 describes our parallel volume rendering system. Section 5 presents the results of our most important experiments. Section 6 relates our work to recent developments in the field of ocean visualization tools. Finally, in section 7 we present our conclusions and the proposals for future work.

2. Volume Visualization

There are two main approaches for the visualization of volumetric dataset: surface rendering and volume rendering. Surface rendering is a technique in which volumetric data is converted into polygons representing the outer surface of the object. Rays of light are tested against all objects in the scene to determine if they intersect any visible surface. Each pixel in the final image is assigned the characteristics of the the closest intersection between the ray and the polygons. It is useful for extracting surfaces from volume data.

Volume rendering, on the other hand, is a technique in which the object of interest is represented by cubic or tetrahedral blocks called voxels. Each voxel has associated with it one or more values quantifying some measured or calculated property of the original object, such as transparency, luminosity, density, or flow velocity. The final color of a pixel in the image comes from the sum of the contributions from each voxel. In volume rendering, the faces are considered to be semi-transparent, allowing the ray of light to pass from voxel to voxel, instead of stopping at its surface.

Compared with surface rendering, volume rendering manipulates much more data, requiring greater computing power. However, it has the ability to preserve the integrity of the original data throughout the visualization process. Interesting features of volumetric data could be lost in surface rendering, embedded in the middle, hidden by outer opaque surfaces. Since the entire dataset is preserved in volume rendering, any part may be viewed, including the internal structures and details.

3. Ocean Visualization

The field of ocean sciences benefits extraordinary from scientific visualization improvements. These improvements allow oceanographers to study the natural system with an unprecedented degree of realism: enhancing observational capability, and linking observations with models. The impact of these

developments on understanding the oceanic processes have many economic and social benefits.

The visualization of the inner structure of oceans enables a more comprehensive and dynamic exploration of their characteristics such as temperature, salinity, velocity and mass. While visualization of individual depth layers is useful, there are phenomena that may be better understood by viewing all depth layers at once, as for example, the Mediterranean outflow. In the Mediterranean sea, evaporation produces a very dense and salty water, heavier than the less salty water of the Atlantic ocean. Therefore, the water flows through the Strait of Gibraltar forming a distinct water mass, which can only be fully analyzed with volume rendering techniques [McPherson and Maltrud, 1998].

The visualization of the interior of ocean datasets can have valuable contributions for many areas, as for example:

- **Climate Research:**

  The understanding of the ocean general circulation is critical to diagnose and predict climate changes and their effects. The oceans control the Earth's weather, because they heat and cool, humidify and dry the air and control wind speed and direction. Visualizing and modeling changes in the distribution of heat in the ocean allow researchers to have ability of projecting future climate and the effect of it in human activities.

- **Offshore Industries:** Visualization of ocean circulation helps cable-laying vessels and offshore oil operations, avoiding and minimizing the impacts of strong currents. In addition, volume rendering of deep ocean assists offshore engineering projects worldwide.

- **Fishing and Mammals Management:** Volume rendering techniques can be used to determine ocean properties that may explain fish and mammal behavior. They help the investigation of ocean habitats and resources. For example, a strong cold pool or high area of salinity may aggregate prey (invertebrates and planktonic fish), providing feeding areas for marine mammals.

4. The Parallel Rendering System

The parallel rendering system we used to visualize ocean dataset is called DPZSweep, and is based on the PZSweep system [Farias and Silva, 2001]. PZSweep is an out-of-core parallel rendering system for irregular datasets, originally developed to run on a shared-memory programming model. Although PZSweep has proven to be very efficient for distributed-shared memory machines (about 85% of parallel efficiency), its task queue programming model is not well-suited for a distributed-memory machine as a cluster of PCs.

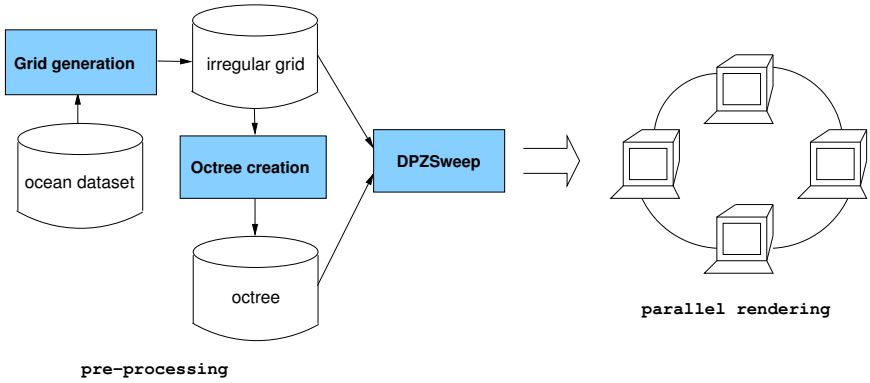

Figure 1. The pre-processing and rendering phases of DPZSweep.

DPZSweep system is a distributed version of the PZSweep, that runs on a cluster of PCs. DPZSweep allows ocean researchers to interactively visualize large volumes of 3D data, providing the following benefits:

- **Efficiency and scalability:** the system remains executing effectively as the problem size increases.
- **Low-cost:** its target architecture, cluster of PCs, can be built with off-the-shelf components.
- **All software implementation:** does not require any hardware graphics on each PC.
- **Portability:** it was implemented on top of C++, MPI and Linux.
- **Free software:** its last version is not ready for distribution, but soon it will be available for ocean researchers to visualize their large volume of datasets.
- **Out-of-core execution:** can render ocean datasets that are too big to fit in main memory.

Pre-Processing

Before starting the parallel rendering process, it is necessary to convert the ocean dataset into the dataset format suitable for the rendering algorithm. This conversion is done by DPZSweep pre-processing phase that has two steps: **grid generation** and **octree creation**. Figure 1 shows a schematic view of DPZSweep, with the pre-processing and parallel rendering phases.

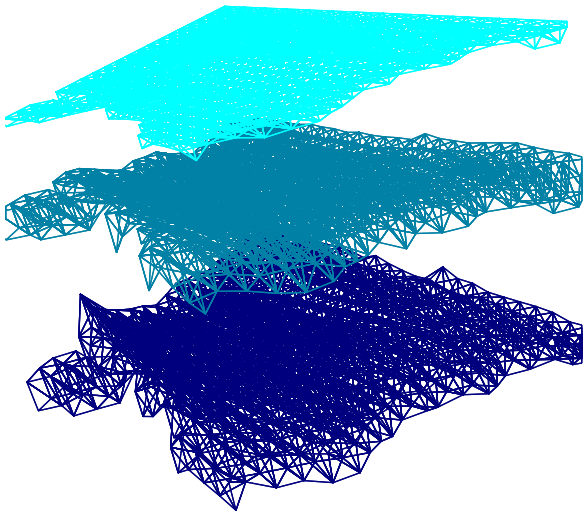

Figure 2. Three layers of the tetrahedral grid generated for the ocean dataset used in our experiments.

Grid Generation. Ocean datasets are usually composed by a regularly spaced longitude, latitude grid with different depth layers. Each point in the grid has a scalar parameter associated, such as temperature, salinity, velocity, and mass. In order to comply with the DPZSweep internal data representation, it was necessary to develop a grid generation module.

The grid generation module first connects each (x,y) point with its neighbors forming rectangles in the depth levels. Each rectangle in the depth level d is connected with the corresponding rectangle in the depth level $d + 1$, forming a parallelepiped. After that, each parallelepiped is divided into two prisms, that are then divided into 3 tetrahedra. In this way, we can create an irregular grid of tetrahedra needed for the DPZSweep parallel rendering algorithm. This irregular grid is stored on a disk file, called grid data file. Figure 2 shows three displaced layers of the ocean tetrahedral grid used in our experiments. The grid where these layers came from was created by our grid generation module.

Octree Creation. After the grid data file is created, it is necessary to create a hierarchical representation of the data, in order to allow the out-of-core rendering of the irregular grid. An octree is built by partitioning and restructuring the grid data. Each octree leaf contains a pointer to a small region of the grid data.

The octree structure is stored on a disk and used in the rendering algorithm to load the grid data regions corresponding to the octree leaves. Only a very small amount of data are brought into the main memory on demand.

It is important to note that the pre-processing phase is done only once per dataset, which means that the same grid and octree can be used for visualizing the dataset from different angles.

Parallel Rendering Algorithm

Our parallel rendering algorithm uses an image-space task subdivision. The screen is broken into small-sized rectangular pieces, called *tiles*. Each tile represents a computational unit of work. A processor processor knows which part of the dataset has to be rendered by the tile id.

Initially, all processors grab a group of non-empty tiles[1] to compute. The sequential rendering of one tile is based on the ZSweep algorithm [Farias et al., 2000]. ZSweep sweeps the dataset vertices, in depth order, with a plane perpendicular to the viewing direction, and projects the faces of cells incident on each vertex.

The group of tiles that is assigned to each processor is determined by the static assignment strategy used. DPZSweep has three different static assignment strategies: contiguous, interleaved and random. Nevertheless, in this work we consider only the random strategy, that distributes the tiles on a random fashion, since it provides the best initial distribution results [Coelho et al., 2005].

Load Balancing

After each group of tiles is assigned to a processor, if the initial assignment generates load imbalance, DPZSweep uses a dynamic load balancing scheme to rebalance the work. Our idea is to take advantage of some well-known distributed information diffusion algorithms to inform the PCs about the system load without using broadcast messages. DPZSweep has three different load balancing algorithms: Nearest Neighbor, Longest Queue, and Circular Distribution. The first algorithm is based on a simple nearest neighbor exchange, while the other two are based on the well-known distributed token-ring algorithm.

Nearest Neighbor. In the Nearest Neighbor (NN) algorithm, processor p searches for unprocessed tiles only on processor $p + 1$[2]. This algorithm avoids excessive message exchanging as each processor just asks for its nearest neighbor. It is based on the idea that if the initial assignment is quite a good distribution, then the dynamic load redistribution can be restricted to some neighbors adjustment.

Longest Queue. The idea of the Longest Queue (LQ) algorithm is to ask the most overloaded processor for work. However, as there is no such global information on a cluster, the algorithm circulates a token, containing the load of each processor. Every time the token reaches a processor, it updates the token with its current load. When a processor is idle, using the token information, it asks for work to the most loaded processor. The one with the longest task queue.

Circular Distribution. The load balancing concept in the Circular Distribution (CD) algorithm is different. CD assembles the assignment and balancing tasks in the same algorithm. The algorithm distributes the work dynamically among processors. CD circulates a token containing the list of work to be done. In fact, this list contains the ids of the tiles that have not been rendered. An idle processor which receives the token, takes out from the token some of the tiles [3] to compute. A busy processor simply passes the token along to its neighbor. The algorithm finishes when the token is empty.

5. Experimental Results

Our experimental environment consists of a cluster composed by 16 processors Intel Pentium III with 800MHz. The nodes are connected by Fast Ethernet (100Mbits/sec). All the nodes run Linux kernel 2.4.20, and the communication is handled using MPI 1.2.5 [Snir et al., 1996].

We decided to use Linux operating system and MPI message passing library for two main reasons: portability and free software usage. In this way, our system allows free sharing of ideas and information, and can be used on a variety of parallel cluster architectures.

Ocean Dataset

We have used in our experiments a sample of the Gulf of Mexico ocean data from Navy Research Laboratory (NRL). The sample has a resolution of 1 degree in latitude and longitude and carries information of 6 depth levels. Since the main focus of this work is in the evaluation of DPZSweep as an efficient and low-cost visualization system for volumetric ocean data, we decided to evaluate only a single time-step and a single scalar value: velocity. We did not include the images generated from our ocean datasets, because, in grey scale, it is not possible to distinguish their details. These images can be found at http://www.lcg.ufrj.br/ rfarias/oceanimages.

The tetrahedralized version of this dataset was locally generated by our grid generation module. In fact, we used three different tetrahedralized versions of this dataset, that we called ocean, ocean1 and ocean2. These versions differ in terms of the number of tetrahedra generated, and represent the same dataset

Table 1. Different levels of tetrahedralized versions of ocean datasets used in our experiments

Datasets Information	
Dataset	# of tetrahedra
ocean	44K
ocean1	356K
ocean2	2854K

with different sizes and different amount of work to be done. The numbers of tetrahedra of each version are listed in Table 1. We used in our experiments an image size of 1024 $\times$ 1024 pixels and a 32-by-32 tile decomposition.

Performance Analysis

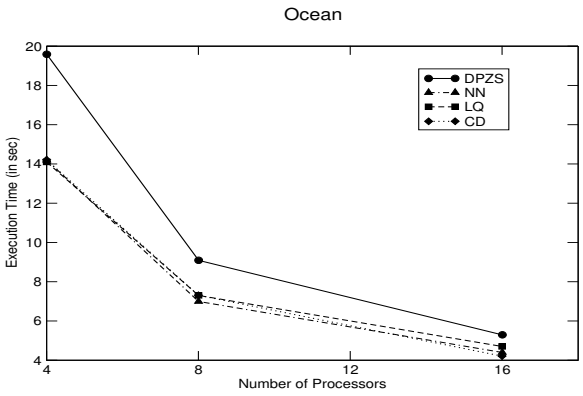

Figure 3. Execution times of DPZS, NN, LQ and CD for ocean.

Figures 3, 4 and 5 show the execution times in seconds for ocean, ocean1 and ocean2, respectively. They were run under 4, 8 and 16 processors for DPZSweep without load balancing (DPZS), and DPZSweep with the proposed load balancing strategies: Nearest Neighbor (NN), Longest Queue (LQ) and Circular Distribution (CD). Table 2 presents the precise values we used to generate these graphics. As we can observe in these results, the version of DPZSweep without load balancing (DPZS) presents the greatest execution times for each of the datasets, because the load imbalance inlaid in the static distribution reduces the system performance. All the systems with load balancing mechanisms perform satisfactorily in the cluster, obtaining up to 95% of parallel efficiency for ocean2 dataset. The reductions in execution time, how-

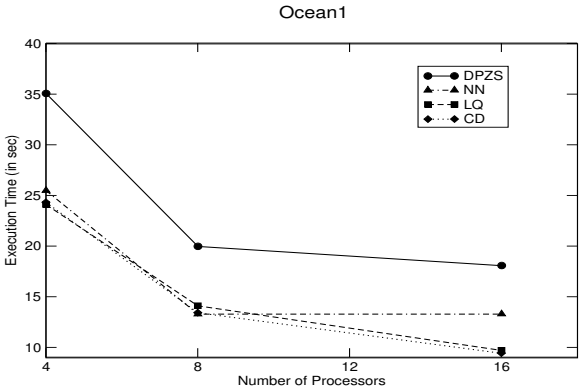

Figure 4. Execution times of DPZS, NN, LQ and CD for ocean1.

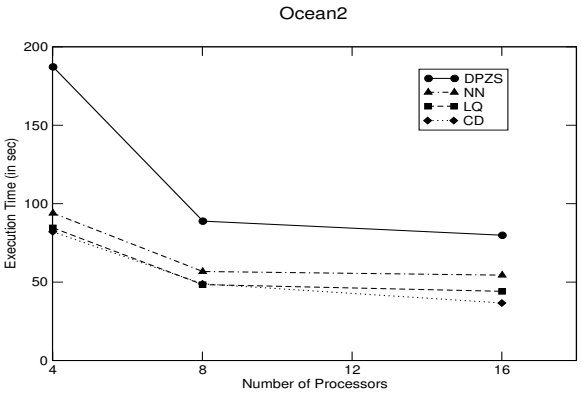

Figure 5. Execution times of DPZS, NN, LQ and CD for ocean2.

ever, are smaller from 8 to 16 processors, as these datasets are not big enough to turn the messaging overhead negligible.

Comparing the performance of the three different load balancing strategies, we can observe that, although the strategies are completely different, they provide almost the same execution time results, and, consequently, present almost the same percentage of load imbalance. We can observe, however, that CD achieves the smallest execution time, and NN not always can rebalance the load only with neighbor exchange.

As can be observed, our distributed load balancing algorithms are efficient in redistributing the load. The performance of our techniques used to spread information to the processors are satisfactory. In addition, although these techniques were evaluated on top of DPZSweep system, they are general enough

Table 2. Execution times results in seconds for DPZS, NN, LQ and CD for ocean, ocean1 and ocean2 datasets, running under 4, 8 and 16 processors

Ocean				
# of proc	DPZS	NN	LQ	CD
4	19.6	14.2	14.1	14.2
8	9.1	7.0	7.3	7.3
16	5.3	4.4	4.7	4.2
Ocean1				
# of proc	DPZS	NN	LQ	CD
4	35.1	25.5	24.1	24.3
8	20.0	13.4	14.1	13.4
16	18.1	13.2	9.7	9.4
Ocean2				
# of proc	DPZS	NN	LQ	CD
4	187.1	93.9	84.6	82.1
8	88.8	56.7	48.4	48.9
16	79.8	54.4	44.1	36.7

to be applied to other tile-based parallel rendering system to obtain interactive visualization times for ocean datasets.

6. Related Work

POPTEX [McPherson and Maltrud, 1998] is a visualization system for the POP ocean model developed at Los Alamos. They achieved interactive rendering times using a full combination of hardware features available on the SGI Origin 2000. Furthermore, they do not provide volume rendering, but only surface rendering.

The ParVox system at JPL[Peggy et al., 1997] is a general purpose parallel volume rendering system that is capable of visualizing large volumes of 4-D simulation/modeling database. ParVox was used to visualize ocean dataset and achieved rendering rates of one frame per second using 256 processing elements of a Cray T3D. Although ParVox was originally developed to run on a Cray T3D, its most recent version runs on Beowulf clusters. However, they do not include any kind of load balancing strategy. Moreover, the system is based on a different rendering algorithm, called splatting.

Vis5D [Hibbard and Santek, 1990] is a freely available system for interactive visualization of large 5-D gridded datasets such as those produced by numerical weather models. Volume rendering was added to Vis5D using polygon approximation but without out-of-core execution. Instead, it uses a compression mechanism to store the entire data file in the main memory.

Jimenez *et al.*[Jimnez et al., 2003] describe a set of 3D and 4D visualization tools and techniques for CORIE, a complex environmental observation and forecasting system (EOFS) for the Columbia River. The volume rendering functionality is based on the VTK (Visualization Toolkit) system. VTK employs parallel rendering, but uses a master-slave approach, restricting its scalability. It can run on a cluster of PCs, however, it does not include any load balancing scheme.

7. Conclusions

In this paper we presented a portable, scalable and low-cost parallel volume rendering system for cluster of PCs. It supports distributed visualization needs demanded by ocean applications. Our system, called DPZSweep, allows ocean researchers to interactively visualize large volumes of 3D data, revealing its internal structure. Our approach was designed to:

- Overcome the high load imbalance imposed by the parallel rendering application, by providing dynamic load balancing with low overhead;
- Provide out-of-core execution, for coping with datasets that are too large to fit in main memory; and
- Use portable and free software infrastructure, as Linux operating system and MPI message passing library.

We made some experiments running ocean data on top of DPZSweep system, and obtained up to 95% of parallel efficiency. The great reductions in execution times were due to the use of our load balancing algorithms. These results prove the efficiency of our techniques to inform the processors about the system load. Moreover, the techniques are general, and can be applied to other tile-based parallel rendering systems to obtain interactive visualization times for ocean datasets.

Acknowledgments

We thank Navy Research Laboratory at Mississippi, USA for providing the ocean dataset, Prof. Robert Moorhead II from Engineering Research Center (ERC) at Mississippi State University for helpful comments and discussions, Jose Oscar Mendonza Latorre for the text revision, and Renato Silva and Ricardo Amorim from LNCC-Brazil for their support and patience to set up the Carcara cluster. This project was developed with the support of the Carcara Project on Parallel Computation, by means of making available their cluster for our experiments.

Notes

1. A non-empty tile is a tile whose shaft actually intersects with cells and vertices of the dataset. Empty tiles have no work to be done, and can be discarded.

2. Note that, p+1 neighbors are considered in a circular fashion, so that the p+1 neighbor of processor (n-1) is processor 0

3. In our experiments we are taking just one tile.

References

[Challinger, 1993] Challinger, J. (1993). Scalable parallel volume raycasting for nonrectilinear computational grids. In *ACM SIGGRAPH Symposium on Parallel Rendering*, pages 81 – 88.

[Coelho et al., 2005] Coelho, A., Bentes, C., and Farias, R. (2005). Distributed load balancing algorithms for parallel volume rendering on cluster of pcs. In *Proc. of the Visualization and Data Analysis Conference*. To appear.

[de Oliveira and Ferreira, 1997] de Oliveira, E. A. and Ferreira, M. A. (1997). Visualizacao da dispersao de efluentes na atmosfera. In *X Simposio Brasileiro de Computacao Grafica e Processamento de Imagens*.

[Djurcilov et al., 2002] Djurcilov, S., Kim, K., Lermusiaux, P.F.J., and Pang, A. (2002). Visualizing scalar volumetric data with uncertainty. *Computers and Graphics*, 2(26):239–248.

[Farias et al., 2000] Farias, R., Mitchell, J., and Silva, C. (2000). Zsweep: An efficient and exact projection algorithm for unstructured volume rendering. In *2000 Volume Visualization Symposium*, pages 91 – 99.

[Farias and Silva, 2001] Farias, R. and Silva, C. (2001). Parallelizing the zsweep algorithm for distributed-shared memory architectures. In *International Workshop on Volume Graphics*, pages 91 – 99.

[Gonzato and Saec, 2000] Gonzato, J. C. and Saec, B. Le. (2000). On modeling and rendering ocean scenes. *Journal of Visualisation and Computer Simulation*, 11(1):27–37.

[Hibbard and Santek, 1990] Hibbard, W. and Santek, D. (1990). The Vis5D system for easy interactive visualization. In *Proc. of the 2nd IEEE Visualization Conference (Vis1990)*, pages 28 – 35.

[Hofsetz and Ma, 2000] Hofsetz, C. and Ma, K.L. (2000). Multi-threaded rendering unstructured-grid volume data on the SGI Origin 2000. In *Third Eurographics Workshop on Parallel Graphics and Visualization*, pages 91 – 99.

[Hong and Kaufman, 1998] Hong, L. and Kaufman, A. (1998). Accelerated ray-casting for curvilinear volumes. In *Proc. of the 9th IEEE Visualization Conference (Vis1998)*, pages 247 – 254.

[Jimnez et al., 2003] Jimnez, W. H., andA. Baptista, W. T. Correa, and Silva, C. (2003). Visualizing spatial and temporal variability in coastal observatories. In *Proc. of the 14th IEEE Visualization Conference (Vis2003)*, pages 75 – 80.

[Koller et al., 1995] Koller, D., Lindstrom, P., Ribarsky, W., Hodges, L. F., , Faust, N., and Turner, G. (1995). Virtual GIS: A real-time 3D geographic information system. In *Proc. of the 6th Conference on Visualization'95*.

[Ma, 1995] Ma, K. L. (1995). Parallel volume ray-casting for unstructured-grid data on distributed-mmory architectures. In *IEEE Parallel Rendering Symposium*, pages 23 – 30.

[Ma and Crockett, 1997] Ma, K.L. and Crockett, T. (1997). A scalable parallel cell-projection volume rendering algorithm for three-dimensional unstructured data. In *IEEE Parallel Rendering Symposium*, pages 95 – 104.

[McPherson and Maltrud, 1998] McPherson, A. and Maltrud, M. (1998). Poptex: Interactive ocean model visualization using texture mapping hardware. In *Proc. of the 9th IEEE Visualization Conference (Vis1998)*, pages 471 – 474.

[Meiıner et al., 1998] Meiıner, M., Huttner, T., Blochinger, W., and Weber, A. (1998). Parallel direct volume rendering on PC networks. In *Proc. of the Intl. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'98)*.

[Muraki et al., 2003] Muraki, S., Lum, E., Ma, K., Ogata, M., and Liu, X. (2003). A PC cluster system for simultaneous interactive volumetric modeling and visualization. In *Proc. of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics*.

[Peggy et al., 1997] Peggy, P., Whitman, S., Mendonza, R., and Tsiao, J. (1997). ParVox – A parallel splatting volume rendering system for distributed visualization. In *James Painter, Gordon Stoll and Kwan-Liu Ma, editors, IEEE Parallel Rendering Symposium*, pages 7 – 14.

[Prakash and Kaufman, 1997] Prakash, C. E. and Kaufman, A. E. (1997). Volume terrain modeling. Technical report, SUNYSB Technical Report.

[Samanta et al., 2000] Samanta, R., Funkhouser, T., Li, K., and Singh, J. P. (2000). Hybrid sort-first and sort-last parallel rendering with a cluster of PCs. In *Proc. of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*.

[Samanta et al., 1999] Samanta, R., J.Zheng, Funkhouser, T., Li, K., and Singh, J. P. (1999). Load balancing for multi-projector rendering systems. In *Proc. of the SIGGRAPH/Eurographics Workshop on Graphics Hardware*, pages 107 –116.

[Santos et al., 1996] Santos, S. R., Valadao, C. E. A., and Dreux, M. (1996). Visualizacao e acompanhamento automatico de sistemas de nuvens. In *IX Simposio Brasileiro de Computacao Grafica e Processamento de Imagens*.

[Snir et al., 1996] Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., and Dongarra, J. (1996). *MPI - The Complete Reference*. The MIT Press, Cambridge, Massachusetts, London, England.

[Watson et al., 2002] Watson, Andrew I., Lerico, T. P., Fournier, J. D., and Szoke, E. J. (2002). The use of d3d when examining tropical cyclones. In *Interactive Symposium on AWIPS*, pages 131–135.