Communication Costs of Parallel Volume Rendering of Large Images on a Network Of Workstations

Oscar Figueiredo, Benoit Gennart, Roger Hersch

Swiss Federal Institute of Technology (EPFL)
Peripheral Systems Laboratory (LSP)
CH-1015 Lausanne

Abstract. Direct Volume Rendering is a popular technique for visualization of 3D datasets that offers many advantages over other algorithms but requires important computing power. Efficient parallelization is therefore an essential need. Taking advantage of an existing LAN instead of using a dedicated parallel architecture can be a cost-effective solution. However this requires special attention in algorithm design to the specific aspects of a LAN as a parallel computing system (long latency, slow throughput). Besides, due to their large volume (up to 1GB), 3D datasets as those produced by modern medical scanners can not be made entirely resident into main memory. We present in this paper the study of an algorithm for direct volume rendering of large images on a local network of workstations which does not assume the presence of the whole dataset in main memory. A method for handling large distributed 3D image datasets is proposed and three parallelization strategies are compared based on communication performance measurements of a LAN.

1 Introduction

Algorithms for three dimensional scientific image rendering are numerous [SFF, CB90, Elv92]. Among them, direct volume rendering has grown in popularity as an image rendering technique over the last few years. The advantages of this method over surface based techniques are well known [Lev88] : elimination of the preprocessing stage needed to extract surfaces from the data, generation of realistic high-quality images, improved display of weak or fuzzy surfaces that are difficult to classify, semi-transparent visualization of successive layers in the object. The cost of these features is the need to traverse the whole dataset each time an image is rendered, which makes volume rendering a computationally expensive process. Therefore, important efforts have been made towards specialized architectures [KB88, SFF] in order to accelerate rendering by using parallelization and *ad hoc* hardware. Several attempts at parallelizing algorithms on multiprocessor machines have also been made [NL92, Cha92, SS92, SGL]. An

Worstations und ihre Anwendungen, Proceedings of the SIWORK'96 Workshop, Clemens Cap, VDF, Zürich, May 14-15 1996, pp 185-194 alternative to this approach consists in taking advantage of the characteristics of a local area network (LAN). This was considered by Giertsen and Petersen in [GP93] though they did not actually use a direct volume rendering algorithm.

Issues in designing a parallel volume rendering algorithm for a LAN-based parallel machine are very different due to the much lower bandwidth of such a system. Furthermore dealing with large datasets such as those produced by 3D medical scanning systems requires specific attention to memory management if one considers that the whole image cannot entirely fit in main memory. Common medical scanners resolution is today of 1000x1000 pixels for each slice, angiography examinations can produce 1000x1000 pixels shots at a rate of 15 to 30 frames per second (remember that a high resolution 1024x1024x1024 voxels 8 bit/voxel scan represents 1 GB of data). Therefore we propose in this paper a preliminary study of the implementation of a parallel direct volume rendering algorithm for large images striped on a network of workstations : we evaluated the communication performance of a network of workstations (latency and throughput) and used the measured parameters in a simple model to determine the communication requirements of three different parallelization strategies. Section 2 describes the computing environment and the general algorithm design (use of a partitioned 3D data volume and direct volume rendering algorithm). Section 3 focuses on a model estimating the compared communication performance of possible parallelization strategies.

2 Computing Environment and Algorithm Design

We consider here an ordinary LAN composed of general-purpose workstations. We make no assumptions about the network technology (Ethernet, FDDI,...) since additional software such as PVM provides insulation from these lower level layers. Thus the computational model is a master-slave model using message passing between master and slaves : the master distributes the elementary jobs to the slaves that actually do the computation work and return the result to the master. Memory is physically distributed between the slaves, there is no shared memory.

2.1 Improving Storage and Access to Large Images

As shown by Hersch in [Her93] access times and overall retrieving time of 2D images can be greatly improved by partitioning the images into sub-tiles of appropriate size and retrieving those tiles, also called *extents*, in parallel. This is particularly efficient if the extents are well distributed among the storage nodes. This is still true for 3D images and we will adopt such a storage/retrieval scheme for the original 3D dataset (Fig.1).

2.2 Parallel Direct Volume Rendering

Our algorithm is based on the classical volume rendering algorithm described by Levoy in [Lev90] and illustrated in Fig.2. Volume rendering is an imaging

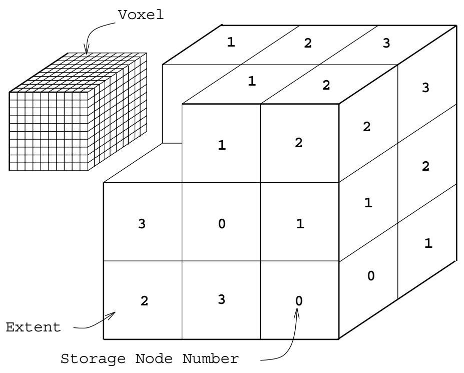

Fig. 1. A 3D Dataset Divided Into Extents

technique for creating 2D views out of a 3D volume. Unlike other techniques that try to fit geometric primitives to the objects in the volume, volume rendering uses a direct ray casting approach through the 3D volume.

As a preliminary step an opacity value is computed for each voxel of the 3D data which determines the parts that will be visible through the rendering. Shading is also performed based on view and lighting information at each voxel. These two operations can be considered as a preprocessing stage and we are not concerned about them. The actual rendering is done using ray casting, which allows compositing of color and opacity values of the voxels in the 3D volume for each pixel of the image. It consists in casting a discrete ray from each pixel of the viewing grid towards the 3D data. Color and opacity values of the voxels of the 3D dataset that the discrete ray encounters along the ray are composited in the sense of [PD84] according to the following transparency formulas :

$$C_{\text{out}} = \frac{\alpha_{\text{in}} C_{\text{in}} + \alpha\,(1 - \alpha_{\text{in}})\,C}{\alpha_{\text{out}}} \quad \text{and} \quad \alpha_{out} = \alpha_{in} + \alpha\,(1 - \alpha_{in})$$

where C_{in}, α_{in}, C_{out} and α_{out} are the colors and opacities of the ray as it enters and exits the visited voxel and C and α are the color and opacity value of the visited voxel.

Let us consider a 3D volume data divided into extents as explained in Sec.2.1. At ray traversal time each ray traverses several extents of 3D data. These extents are stored on the disks attached to different computation nodes.

Parallelization occurs at the ray level. All the relevant information to the computation of a ray (current visited voxel in the 3D dataset, current value of accumulated color and opacity,...) is stored in what we call *ray data packet*. For each ray (*ie* for each pixel of the final image) a ray data packet is created by the master and handed to the slave that stores the first extent of the 3D volume hit by the ray. The slave moves the ray data packet through the extent updating its values at each traversed voxel. When the ray data packet reaches an extent

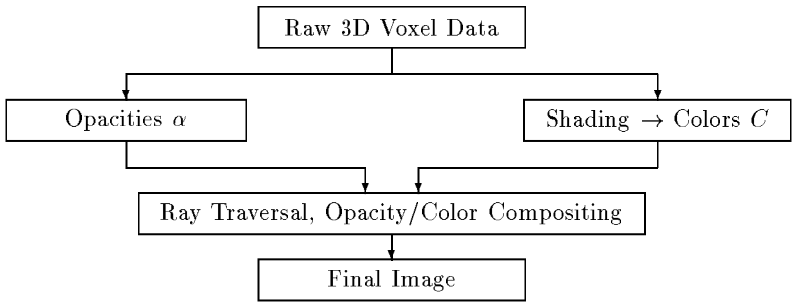
Raw 3D Voxel Data
Opacities α
Shading $\rightarrow$ Colors C
Ray Traversal, Opacity/Color Compositing
Final Image

Fig. 2. Rendering Algorithm Overview

boundary, it is transferred to the slave storing the adjacent extent. After the last extent has been traversed, the ray data packet is returned to the master.

Since we are working in an environment that provides very slow communications, one of our major concerns is to reduce both the number and the size of data transfers between the computing nodes. We consider three parallelization strategies.

Algorithm with Partial Accumulation and Ray Transfers A first solution is to leave the extent distribution between the nodes unchanged. Each node is given the calculation of a ray as a job. When during ray traversal an extent that is not stored locally is entered, the slave node sends the ray data packet to the node that stores that extent. Since ray data packets are small (8 to 16 bytes) this appears as a good solution for minimizing the global communications volume. However the number of individual communication operations is large since ray data packets are transfered several times during traversal of each ray.

Algorithm with Partial Accumulation and Transfer of Sets Of Rays The previous solution is simple to implement but requires a large number of individual communication operations. Instead of transfering ray data packets one by one when needed, an alternative is to group ray data packets that should be sent to the same node together and send them in a single communication operation. Rays leave a traversed extent through at most three sides. They can therefore be grouped into three sets and be transferred in only three communication operations.

Algorithm with Preliminary Extent Shuffle The previous algorithms are not optimal in respect to the number of communications. In order to ensure that the calculation for a ray can be entirely carried on the same node without the need for data transfers (either extents or ray data packets) during ray traversal, an alternate solution is to perform a preliminary extent shuffle between all nodes depending on the view direction. This concept is similar to shear-warp algorithms [LL94, COF95] where data is appropriately realigned before actual ray-casting. Unlike the previous algorithm this one requires transfer of large quantities of data but this step is done once for each rendering operation and requires only a small number of individual communication operations.

2.3 Data Flow

We give here a summary of the data flow occuring in these algorithms. Volume data rendering consists of the following steps :

- the master sends a set of rays (a single ray in the first algorithm) to a slave
- the slave reads the extent traversed by the set of rays from the disk
- the slave proceeds with the calculation of the rays in the ray data packet as they traverse the data extent
- the slave groups all the rays that entered the extent in three sets for the three sides through which the rays leave the extent
- the slave sends to the appropriate nodes the ray data packet sets.

Note: Of course the last two steps do not apply for the third algorithm where ray data packet transfers between the slaves do not happen.

Reading from disks, data transfers and computation can all be pipelined if one considers the system as a whole : some workstations may communicate while the others are accessing data on disks or computing. The pipelining of communication and data transfers on an individual workstation depends on the quality of the communication hardware. Some communication hardware performs the packetization of messages and thus allows simultaneous computation and communication on a single workstation. Regardless of the possible degree of pipelining it is still important to know the ratio between computation and communication.

3 Performance Issues

3.1 Communication Performance of FDDI and Ethernet LANs

In order to have a realistic estimation of the communication cost of each algorithm we first ran a series of tests on a local network to determine actual latency and throughput values in an experimental setup. Measures were carried on a 100 Mb/s FDDI network of DEC Alpha workstations running a parallel program using the PVM communication library. Figure 3 shows the delay (ie. the elapsed time between the emission of a message and the end of the reception of the acknowledgment message from the recipient) against the size of the messages exchanged between two processes. The slope of the linearization of the experimental curve represents the throughput and the extrapolated delay for a message size of zero represents the network latency (ie the elapsed time between a transfer request by the user and the actual emission of the first byte).

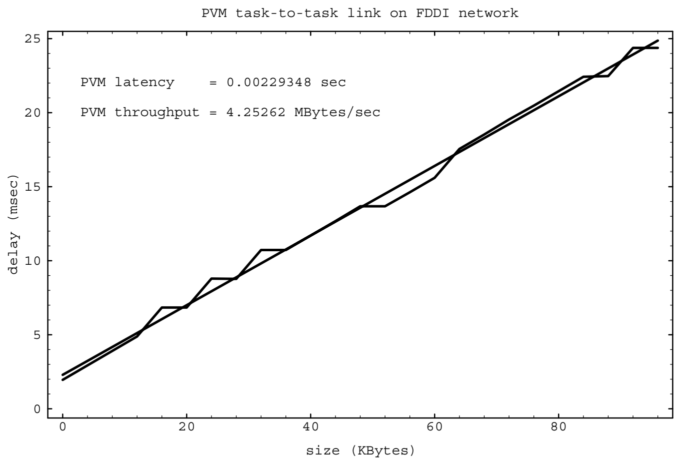

Fig. 3. Latency and Throughput of a PVM Link on DEC Alpha Workstations connected by a FDDI network

We denote δ this average latency time on the network. τ represents the average sustained throughput on the network once the communication was established. In the case studied, the values for the latency and the throughput are $\delta = 2$ ms and $\tau = 4$ MB/s. Better results for the throughput are certainly possible using different parameters and more favorable load conditions but these are the values obtained under real-world conditions on a network shared among several applications and we assume for the calculations below that the effective throughput is limited to $\tau = 4$ MB/s. For comparison purposes we also made the same measurements using the same methodology on a very lightly loaded Ethernet network. The following values were obtained : $\delta = 1\,\text{ms}$ and $\tau = 900\,\text{KB/s}$. The lower values of these two parameters for an Ethernet network allow us to analyze their respective effects on the proposed parallelization strategies.

3.2 Performance Estimation of the Proposed Algorithms

In order to determine which of the three algorithms provides better performance we give here a rough estimate of the communications cost of all three algorithms as well as an estimate of their computation times for reference.

Cost of Ray Data Packet Transfers Let N_{R} be the number of rays to cast (this is equal to the number of pixels in the final image). Let l_{R} be the average number of extents of the 3D dataset crossed by one ray. Each ray requires the transfer of l_{R} ray data packets on the average. Assuming a volume of $a \times b \times c$ voxels divided into $A \times B \times C$ extents, l_{R} is of the order of A, B or C. The cost of each transfer is estimated to be $\delta + \frac{S_{\rm R}}{\tau}$ where $S_{\rm R}$ is the size in bytes of the ray data packet. Global communication time for all rays is then estimated to be :

$$t_{\rm transfer} = N_{\rm R}\ l_{\rm R}\ \left(\delta + \frac{S_{\rm R}}{\tau}\right)$$

Cost of Transfer of Ray Data Packet Sets Let $E_{\rm X}$, $E_{\rm Y}$, $E_{\rm Z}$ be the width, height and depth in pixels of the extents. The number of rays that cross an extent is of the order of $\frac{1}{2}(E_{\rm X}E_{\rm Y} + E_{\rm X}E_{\rm Z} + E_{\rm Y}E_{\rm Z})$ which is the mean value of the extremum values that can be obtained for specific view directions : $E_{\rm X}E_{\rm Y}$, $E_{\rm X}E_{\rm Z}$ and $E_{\rm Y}E_{\rm Z}$ are the number of rays that cross an extent when viewing respectively along the Z, Y and X direction, $E_{\rm X}E_{\rm Y} + E_{\rm X}E_{\rm Z} + E_{\rm Y}E_{\rm Z}$ is the number of rays that cross an extent when viewing along a direction at 45° from the three main axes. In the most general case these rays leave the extent through one of three possible sides. The ray data packets can thus be grouped in three separate sets of size $\frac{1}{6}(E_{\rm X}E_{\rm Y} + E_{\rm X}E_{\rm Z} + E_{\rm Y}E_{\rm Z})S_{\rm R}$ and be transmitted with a single communication operation for each. To simplify we consider cubic extents such as $E = E_{\rm X} = E_{\rm Y} = E_{\rm Z}$. The total communication cost is in this case :

$$t_{\rm transferSet} = \frac{2\ N_{\rm R}\ l_{\rm R}}{E^2}\ \left(\delta + \frac{E^2 S_{\rm R}}{2\tau}\right)$$

Cost of Preliminary Extent Shuffle Let $N_{\rm E}$ be the total number of extents in the 3D dataset and $S_{\rm E}$ be the size in bytes of one extent. If S is the number of storage nodes and the extents are regularly distributed among the nodes and within the dataset then we can consider that on the average only a fraction $\left(1 - \frac{1}{S}\right)$ of the total number of extents needs to be moved from one node to another. Furthermore it has been reported that in typical classified volumes 70 to 95% of the voxels are transparent [LL94], we estimate the ratio of completely transparent extents to be around 50%. These extents do naturally not need to be transferred. Therefore the total communication time for the extent shuffle can be estimated to be :

$$t_{\rm shuffle} = \frac{1}{2}\ N_{\rm E}\ \left(1 - \frac{1}{S}\right)\ \left(\delta + \frac{S_{\rm E}}{\tau}\right)$$

Computation and Disk Reading Times Let $\tau_{\rm comp}$ be the computation throughput. The number of traversed extents is of the order of $\frac{2}{3}N_{\rm R}l_{\rm R}E$ and the average computation time is then given by :

$$t_{\rm comp} = \frac{2\ N_{\rm R}\ l_{\rm R}\ E}{3\ \tau_{\rm comp}}$$

Disk reading times depend on the chosen parallelization strategy. For example, in the case of the algorithm using transfers of ray data packets sets, the total disk reading time is :

$$t_{\rm disk} = \frac{2\ N_{\rm R}\ l_{\rm R}}{3\ E^2}\ \left(\delta_{\rm disk} + \frac{S_{\rm E}}{\tau_{\rm disk}}\right)$$

3.3 Summary of Numerical Results

We give here numerical results for different values of the parameters. We consider two 3D dataset sizes of 512^3 voxels (128 MB) or 1024^3 voxels (1GB) divided into extents of dimensions 32^3 voxels (32 KB) or into extents of dimensions 64^3 voxels (256 KB). The final image is always 512^2 pixels. Parallelization is done on 4 workstations. Ray data packet size is 8 bytes. Computation throughput is estimated at 4 MB/s for each machine. This is based on [LL94] considering also opacity and shading computation times.

Dataset Size	Extent Size	FDDI			Ethernet			Comput. Time*
		t_{trans}	t_{trSet}	t_{shuf}	t_{trans}	t_{trSet}	t_{shuf}	
512^3	E=32	5200 s	15.2 s	15 s	2600 s	28 s	56 s	5 s
	E=64	2620 s	3.8 s	12.4 s	1320 s	12 s	55 s	5 s
1024^3	E=32	10500 s	30.5 s	120 s	5300 s	56 s	450 s	10 s
	E=64	5200 s	7.6 s	99 s	2600 s	24 s	440 s	10 s

**per workstation*

From this table we can see that the two network parameters δ and τ have different influences on the three algorithms : the first one requires an important number of communication operations and thus behaves better on the Ethernet network as latency δ is smaller. On the opposite, the algorithm using extent shuffle is more sensitive to the network throughput and thus achieves better performance on a FDDI network. The second algorithm which is a compromise solution is more efficient on a fast network but shows less performance degradation on a slow Ethernet network than the extent shuffle algorithm. The second parallelization strategy which provides the better balance between global communication volume and number of individual communication operations gets the better results in most cases, the only exception being on a FDDI network with a relatively small image and small extents. This shows that the extent size is also an important parameter and must be chosen carefully. 64^3 voxels seems to be an optimum size as bigger extent sizes would reduce parallelization granularity and lessen the effect of some optimizations (for instance the ratio of empty extents would become much smaller). It can be noticed also that the algorithm using extent shuffle scales poorly in the case of big datasets from which relatively small images are extracted. This can be explained by the fact we considered brute force shuffle of all extents (except the empty ones) in all cases. Naturally one should try to shuffle only those extents that are actually traversed by the rays.

The product $\tau\delta$ represents the number of bytes that could be transmitted at rate τ during the latency time δ. This is an important parameter in the choice of an algorithm over the others : the bigger it is and the more favorable becomes the algorithm using extent shuffle because it minimizes the number of communication operations.

	τ	δ	$\tau\delta$
FDDI	4 MB/s	2 ms	8 KB
Ethernet	900 KB/s	1 ms	900 Bytes

For reference, measurements have reported a throughput of 4 MB/s with a 10 ms latency for usual hard disk drives, this results in an effective throughput of 3.4 MB/s when reading 256 KB blocks from the disk. We give here a short summary of the costs of the different operations along the data flow for the rendering of a 1024^3 voxels set divided into 64^3 voxels extents using the second parallelization strategy on a FDDI network. This table distinguishes between *raw* throughput which is the hardware theoretical throughput, *measured* throughput (τ) which is the experimentally measured (or expected) throughput and *effective* throughput which takes into account the size of read/transmitted data blocks and the latency.

	Latency	Throughput			Time
	δ	Raw	Measured (τ)	Effective	t
Disk Reading*	10 ms	4 MB/s	4 MB/s	3.4 MB/s	7.7 s
Computation*	-	-	4 MB/s	4 MB/s	6.7 s
Data Transfer	2 ms	12 MB/s	4 MB/s	3.4 MB/s	7.6 s

**per workstation*

We can see that when running on 4 workstations, disk access and computation times are of the same order as communication costs. Therefore, assuming adequate pipelining of communication and computation we can hope to get an optimum speedup using 4 processors to parallelize digital volume rendering. Reducing the computation time by adding more processors would only improve marginally the overall execution time as the communication costs remains fixed. In addition these figures show that communication times are hidden by computation times in the pipeline, so duplicating the 3D dataset on each computation node would bring no benefit. This is particularly interesting since we deal with large 3D volumes.

4 Conclusion

In this paper we have shown that volume rendering of large images on a network of workstations requires specific algorithm design. We propose a method to handle large 3D image datasets that cannot entirely fit into memory. We also propose three parallelization strategies and a simple model to estimate their individual communication costs. With this model and communication performance measurements made on a FDDI network of DEC workstations we have shown that in this case minimizing the total number of communication operations is much more efficient than minimizing the global volume of transferred data. A good balance between these two parameters is necessary and we presented an algorithm that satisfies this requirement. Estimation of computation times let us expect good parallelization on a small number of workstations (4 to 8). Our study also shows that duplicating the 3D dataset is useless and brings no improvement on the global time required for the rendering. Future research directions include optimized extent shuffle where only traversed extents would be shuffled which requires preliminary knowledge of the depth that rays are expected to traverse through the volume in the case where early termination is considered.

References

[CB90] Jean-Louis Coatrieux and Christian Barillot. *3D Imaging in Medicine*, volume F60 of *NATO ASI Series*, chapter A Survey of 3D Display Techniques to Render Medical Data, pages 175–195. Springer-Verlag, 1990.

[Cha92] Judy Challinger. Parallel volume rendering on a shared-memory multiprocessor. Technical report, University of California at Santa Cruz, Santa Cruz, CA 95064, March 1992.

[COF95] Daniel Cohen-Or and Shachar Fleishman. An incremental alignment alignment algorithm for parallel volume rendering. In Frits Post and Martin Göbel, editors, *Eurographics '95*, volume 14, pages C–123/C–133. Eurographics, Blackwell Publishers, 1995.

[Elv92] T. Todd Elvins. A survey of algorithms for volume visualization. *Computer Graphics*, 26(3):194–201, August 1992.

[GP93] Christopher Giertsen and Johnny Petersen. Parallel volume rendering on a network of workstations. *IEEE Computer Graphics and Applications*, 13(6):16–23, Nov 1993.

[Her93] Roger D. Hersch. Benefits of an image oriented parallel file system. In *Proceedings Conf. Storage and Retrieval for Image and Video Databases*, 1993.

[KB88] Arie Kaufman and R. Bakalash. Memory and processing architectures for 3d voxel-based imagery. *IEEE Computer Graphics and Applications*, 8(11):10–23, November 1988.

[Lev88] Marc Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8:29–37, may 1988.

[Lev90] Marc Levoy. Efficient ray tracing of volume data. *ACM transactions on Graphics*, 9(3):245–261, July 1990.

[LL94] Philippe Lacroute and Marc Levoy. Fast volume rendering using a shear-warp factorization of the viewing transformation. In *SIGGRAPH 94 : Computer Graphics Proceedings*, Annual Conferences Series, pages 451–458, Orlando, FL, July 24-29 1994. ACM.

[NL92] Jason Nieh and Marc Levoy. Volume rendering on scalable shared-memory mimd architectures. In *Workshop on Volume Visualization*, pages 17–24, Boston, October 1992. ACM, ACM.

[PD84] Thomas Porter and Tom Duff. Compositing digital images. In Hank Christiansen, editor, *Computer Graphics*, volume 18, pages 253–259. ACM SIGGRAPH, ACM, July 1984.

[SFF] M. R. Stytz, G. Frieder, and O. Frieder. Three-dimensional medical imaging : Algorithms and computer systems.

[SGL] Jaswinder Pal Singh, Anoop Gupta, and Marc Levoy. Parallel visualization algorithms : Performance and architectural implications.

[SS92] Peter Schröder and Gordon Stoll. Data parallel volume rendering as line drawing. In *Workshop on Volume Visualization*, pages 25–32, Boston, MA, October 1992. ACM, ACM.

This article was processed using the LaTeX macro package with LLNCS style