Communication Costs for Parallel Volume-Rendering Algorithms

Ulrich Neumann

neumann@cs.unc.edu
Department of Computer Science
University of North Carolina at Chapel Hill

Abstract

This paper examines the many ways to structure parallel volume rendering algorithms and analyzes the communication costs associated with them. Parallel volume rendering algorithms are enumerated through a taxonomy which sorts them into two main classes that exhibit similar communication costs: *image* and *object* partitions. The intrinsic communication costs for algorithms in these classes are analyzed independent of an implementation. Given a network model for a target system, an algorithm's intrinsic communication cost can be used to estimate the time consumed by communication and the effect upon communication time as the system size and data size are varied. Communication cost and time are measured on the Intel Touchstone Delta to verify the predicted scaling behavior. The results show that, for a fixed screen size, systems with mesh networks scale well for object partition algorithms − the time required for communication decreases as the data and system sizes increase.

1. Introduction

The computational expense of volume rendering motivates the development of parallel implementations on multicomputers. Through parallelism, higher frame rates are achieved which provide more natural viewing control and enhanced comprehension of three dimensional structure. Many parallel implementations have been reported, but no framework has been established to allow comparisons of their relative merits independent of their host hardware. This paper enumerates and classifies parallel volume rendering algorithms suitable for multicomputers with distributed memory and a communication network. Communication costs are determined for classes of parallel algorithms by considering their inherent communication requirements. This study of algorithms and their communication costs should be useful to designers and implementers of parallel volume rendering hardware and software systems.

1.1. Algorithms and Rendering Methods

There is a distinction between a *parallel volume rendering algorithm* and a volume rendering method like ray casting or splatting. A parallel algorithm describes how data and computation is distributed among the resources of a system. In such a description, the rendering method is not an issue and may be unspecified. For example, a simple parallel algorithm for a system with *n* nodes divides the screen into *n* regions and assigns each node a separate region to render. This parallel algorithm does not specify what rendering method is used by each node to render its region. By considering parallel algorithms and rendering methods independently, the performance ramifications of each issue are separately more clearly.

1.2. Redistribution

Communication costs are an important issue for parallel system and software designers to consider. The selection of a parallel algorithm has a major impact on the communication requirement between nodes. Unless all nodes have a local copy of the data, or viewing positions are severely restricted, a parallel volume rendering algorithm intrinsically requires communication between compute nodes. The transfer between nodes of volume or image data necessitated by a parallel algorithm is defined here as *redistribution*. Redistribution costs are measured as the quantity of data transferred (redistribution size) and the time consumed by moving it over the network (redistribution time). Replication of data at each node is wasteful for large numbers of nodes and impossible when data size exceeds local memory size. Restricting the viewing positions limits one's ability to explore the data. Therefore, in most practical cases, redistribution must occur.

The upper bound of redistribution size is independent of the rendering method. The choice of rendering method may reduce the actual requirement. For example, nodes that render by ray casting may adaptively terminate rays and therefore not access portions of the data that would otherwise be needed. Such efficiencies are data dependent but often significant. In this analysis, the peak communication requirement is derived as an upper bound with the understanding that rendering efficiencies may reduce this by some factor.

1.3. Mesh Networks

Communication between nodes in multicomputers is frequently through two and three-dimensional mesh-connected networks. (E.g.: Stanford Dash, Intel Delta and Paragon, MIT J-Machine, Caltech Mosaic.) The performance of these communication networks with parallel volume rendering algorithms is one focus of this paper. Mesh networks scale easily so they are a practical choice for

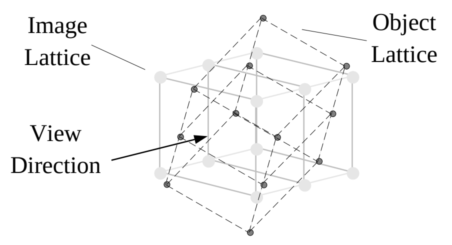

Fig. 1 - Image and object lattice volumes

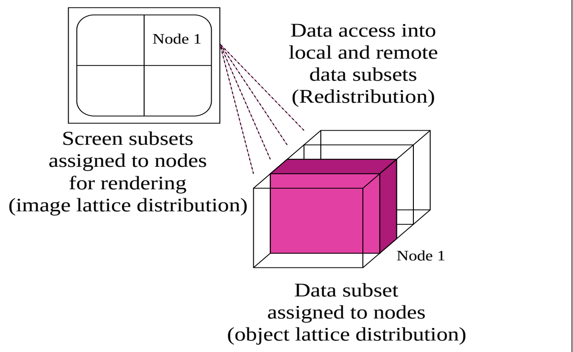

Fig. 2 - Image partition algorithms

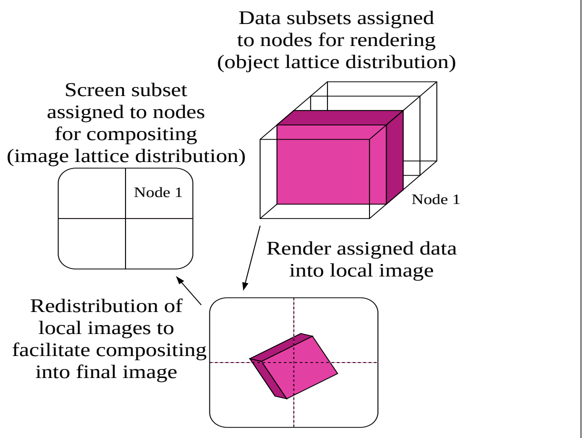

Fig. 3 - Object partition algorithms

systems ranging from tens to thousands of nodes. This paper provides models for predicting the redistribution costs incurred by different parallel algorithms on a range of mesh system sizes. The models predict that, for a fixed image size, the class of object partition algorithms requires decreasing communication time as the data size and number of nodes increases. This scaling behavior makes highly parallel systems feasible with thousands of nodes connected by a modest 2D mesh network without loss of performance due to communication .

2. Parallel Algorithms

In parallel volume rendering algorithms subsets of two volumes must be distributed over the nodes of a system (Fig. 1). The data to be visualized is one volume and referred to as the *object lattice*. The other volume is the set of points whose values are computed to produce an image. Points in

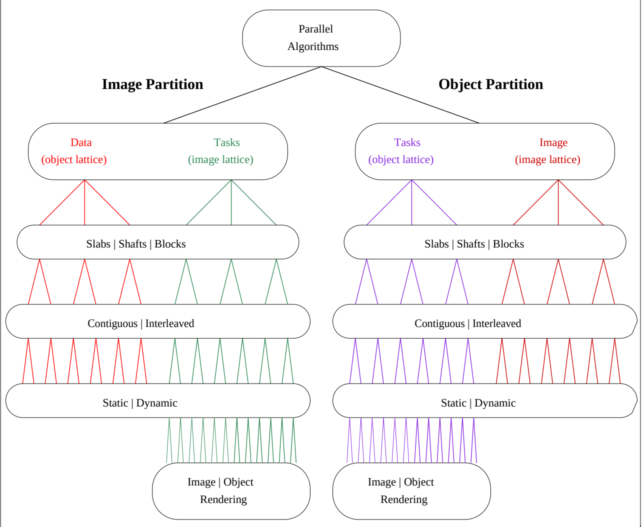

Fig. 4 - Taxonomy of
parallel volume rendering algorithms

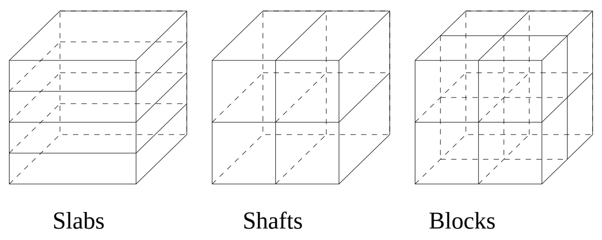

Fig. 5. - Data subset shapes

this volume are aligned behind the image pixels along the view direction and are referred to as the *image lattice*. The work assigned to a node is based on either its assigned object or image lattice subset . This task assignment distinction creates two main classes of parallel algorithms, *image partitions* and *object partitions*. In an image partition (Fig. 2), nodes are assigned volumes of image lattice points to compute. Redistribution occurs as volume data moves between nodes to facilitate interpolation of the assigned points. In an object partition (Fig. 3), each node renders a local color and opacity image of its assigned data subset. Redistribution occurs as local images are moved to facilitate their combining into a complete image. Member algorithms in each class differ in the shapes of the data and image subsets, the subset's static or dynamic nature over time, and the spatial relationship of the subsets to each other [Neum93]. A taxonomy (Fig. 4) enumerates the possible algorithms graphically. Note that the choice of image or object order rendering methods is also a variable.

2.1. Lattice Subsets

Subsets of the object or image lattices may be distributed among nodes in three shapes: *slabs*, *shafts*, and *blocks* (Fig. 5). When data is redistributed, the subset size is the granularity of the transfer. To control transfer size there may be more data subsets than nodes; a node may store multiple subsets in its local memory. If these multiple subsets are spatially adjacent, (e.g., multiple slices forming a slab) they are classified as *contiguous*. Any non-adjacent arrangement is classified as *interleaved*. If the distribution of subsets varies between frames, the distribution is *dynamic*. An unchanging distribution is *static*.

Because their redistribution costs differ, image partitions are subdivided into two different subclasses, one with static data distributions, and the other with dynamic data distributions. This distinction is not made for object partitions since static and dynamic data distributions exhibit the same redistribution costs. The analysis of the redistribution costs for three classes of algorithms is sufficient to cover all the approaches shown in the taxonomy.

3. Network Model

A network model is needed to estimate the redistribution time for a particular system once the redistribution size for an algorithm is known. This section develops a model for mesh and toroidal networks commonly used in multicomputers. Current generation mesh and toroidal networks employ *virtual cut-through*, *oblivious*, *wormhole* routing techniques (e.g.: Intel Delta and Paragon). This terminology and the characteristics of these networks are reviewed below.

Virtual cut-through refers to the way messages pass through intermediate network nodes between the source and destination nodes. Routing logic on intermediate nodes detects the message destination encoded into the message header, and forwards the message to a neighboring node without interrupting the intermediate node's processor.

A network that has fixed, deterministic message routing paths for any source-destination node pair, is referred to as *oblivious*. In contrast, an *adaptive* network routes a message based on the utilization of local paths.

A *wormhole* routing network establishes a connection between the source and destination nodes through which the message flows. If a needed path is already occupied, progress toward establishing the connection is blocked until the needed path is relinquished. Once a connection is established, the message (or packet) flows without interruption. A partially-routed blocked message occupies paths that may in turn block other messages.

Glossary of abbreviations:

a	dimension of a mesh network
b	bisection bandwidth
c	channel bandwidth for one link in a network
d	volume data size - number of samples
h	average cache hit ratio
k	edge length of a network
m	redistribution size - amount of data moved per frame
n	number of nodes
r	replication factor - number of copies of the volume data stored in the system
p	number of pixels in an image
q	network injection bandwidth at a node
t	time consumed per frame

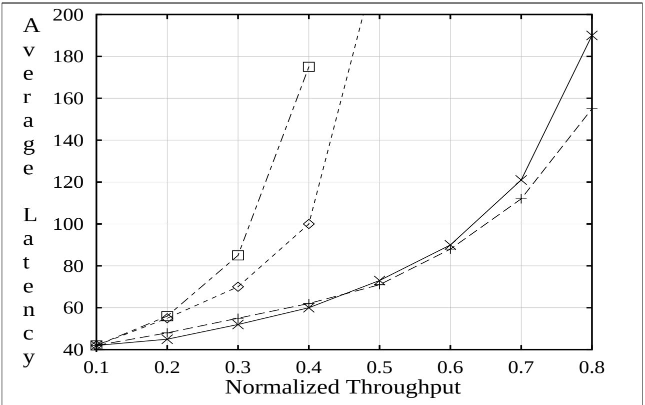

Fig. 6 - Average latency vs. normalized throughput (adapted from [Ngai89])

John Ngai [Ngai89] characterized these networks while proposing adaptive enhancements. Some of Ngai's test results for 2D and 3D mesh and torus topologies are reproduced in figure 6. The test conditions of uniformly-random message destinations and fixed-length single-packet messages are reasonable simplifications of the conditions encountered in some of the parallel algorithms considered here. The major performance aspects of these networks are the *throughput* and average *latency* of messages as a function of *applied load* and *bisection bandwidth*.

Throughput is a measure of aggregate network message delivery bandwidth.

Latency is the delay from a source node's injection of a message header into the network until the complete message exits the network at the receiving node.

Applied load is the aggregate message injection bandwidth into the network.

Bisection bandwidth is the aggregate peak bandwidth through the minimal set of routing channels that, when removed, splits the network into two equal and disjoint parts.

For a network with *n* nodes, let $n = k^a$, where *k* is even and *a* is the dimension of the mesh. The bisection width of a mesh is n / k channels. The bisection bandwidth of a mesh and torus is

$$b_{\text{mesh}} = c\, n / k \qquad (1)$$

$$b_{torus} = 2\, c\, n \,/\, k \qquad (2)$$

where c is the bandwidth of a single communication channel. Toroidal topologies have additional wrap-around connections that double the mesh bisection for a given k and n.

Under steady state conditions, network throughput equals the applied load. As the applied load increases beyond what the network can deliver, messages are queued at the source and delayed without bound; this *source queueing* time is separate from the network latency measure. Throughput in figure 6 is normalized to the maximum load that saturates the bisection bandwidth. All nodes inject fixed-length messages into the network at a uniform rate and to uniformly distributed destinations. The network is bidirectional with separate paths for message flow in opposite directions. Nodes on each side of the bisection send one-half of their messages across the bisection. An *injection bandwidth q* at each node saturates the bisection paths when

$$q_{mesh} = 4\, c \,/\, k \qquad (3)$$
$$q_{torus} = 8\, c \,/\, k \qquad (4)$$

Since a torus has twice the bisection bandwidth of a mesh with identical dimensions, the injection bandwidth required to saturate the bisection is also doubled. At this *saturation load*, the aggregate bandwidth injected into the network is n q, which represents a normalized load of 1.0. The normalized load and normalized throughput are a fraction of the saturation load.

Communication times are estimated in this paper under the assumption that c is sufficient to keep the normalized load and throughput ≤ 0.3 for meshes and ≤ 0.6 for tori. Under these conditions the average latency is roughly equal in either network of size n.

4. Parallel Algorithm Performance

Three classes of parallel algorithms are considered because of their intrinsically different redistribution costs: image partitions with static data, image partitions with dynamic data, and object partitions.

4.1. Image Partition with Static Data Distribution

In this class of algorithms, nodes are assigned one or more subsets of image lattice points to compute. Often shafts subsets are used which equates to assigning screen regions to nodes [Chal91] [Corr92] [Mont92] [Nieh92] [Vézi92] [Yoo91]. Data subsets are distributed among the nodes in a static distribution — a specified data point is always stored in the same node's local memory. To render their region(s), nodes access remote or local data as necessary (Fig. 2) based on the current view transformation. Interleaved static data distributions produce redistribution routing patterns that approximate the random distribution used to characterize network performance. A fine-grain randomly interleaved block data distribution achieves this and makes the redistribution size view-independent [Nieh92]. This data distribution is the context for the remainder of section 4.

4.1.1 Redistribution Costs

Redistribution size is affected by replication of the data set. Define a data size d and a replication factor r $(1 \leq r \leq n)$ where r is the number of copies of the data stored in the system. Each node needs about 1/n'th of the data to render its assigned region. Nodes have (r d/n) randomly located data points in their local memory, and of those, (r d/n2) points are needed for rendering their assigned region. Redistribution size is

mredist d r d/n (5)

If r = n, every node has a complete copy of the data and the redistribution size is zero. If only one copy of the data resides in the system r = 1 and the redistribution size is d d/n. The redistribution time on a 2D mesh under a normalized load of 0.3 is

```
t2Dredist     mredist / (0.3 n qmesh)
              (d  r d/n) / (1.2 n c/k)
              (d  r d/n) / (1.2 n1/2 c)        (6)
```

Network throughput is O(n1/2), so if n is scaled in proportion to d, throughput increases too slowly to maintain constant redistribution time. Toroidal 2D topologies exhibit the same behavior except for a factor of four in their throughput. This is the expected behavior of mesh networks the average injection bandwidth approaches zero as the mesh size increases.

The throughput of a 3D mesh of n nodes is n1/6 greater than a 2D mesh for the same latency, so

t3Dredist (d r d/n) / (1.2 n2/3 c) (7)

Equation 7 shows that 3D topologies scale only slightly better than their 2D counterparts (Eq. 6) for this class of algorithms.

4.1.2. Observations

The scaling of redistribution time for image partition algorithms with static data distributions on mesh networks is summarized as

a) when d and n are increased proportionally the redistribution time increases, and

b) for constant d and increasing n the redistribution time decreases.

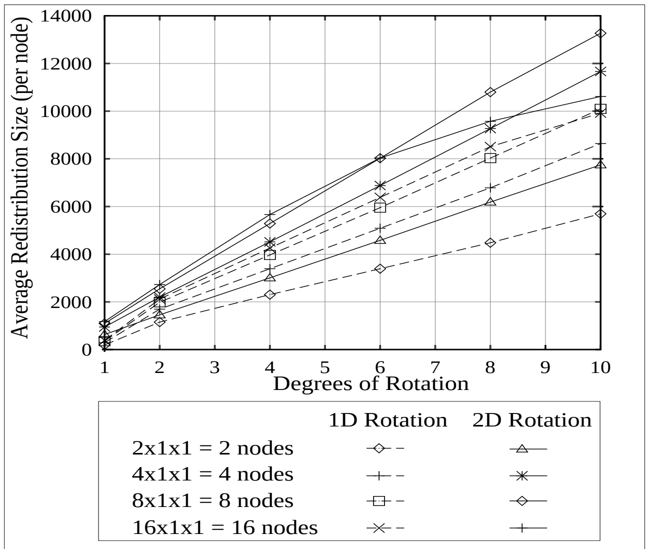

Fig. 7 - Redistribution with slabs on 1D network

Computation time for rendering has not been addressed, but regardless of the rendering method, the growth of redistribution time as *d* and *n* increase together will eventually limit overall system performance. Alternatively, a faster (more expensive) network must be provided as the system size is increased.

Section 1 described how rendering efficiencies can reduce redistribution size. It may also be lowered by using large caches to take advantage of image and temporal coherence [Corr92]. Cached values from the previous frame are likely to be needed for the current frame. With caches the redistribution costs in Eqs. 5 - 7 are modified by setting $r = nh$, where h is the average hit ratio of the caches.

4.2. Image Partition with Dynamic Data Distribution

This class of parallel algorithms differs from all others in that data migrates among nodes in response to view

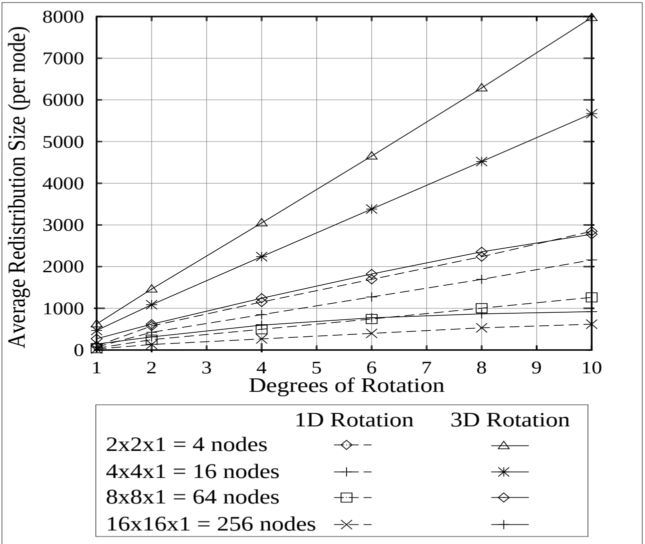

Fig. 8 - Redistribution with shafts on 2D network

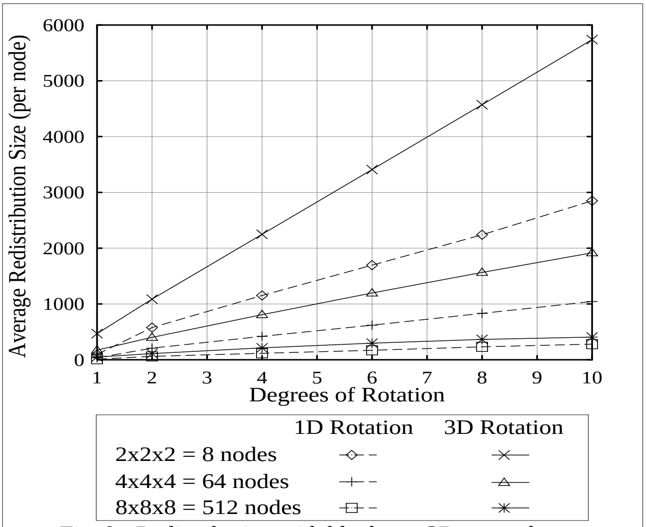

Fig. 9 - Redistribution with blocks on 3D network

changes. This differs from the use of caches with a static data distribution in that there is no assigned node that will always have a particular data value. For a dynamic data algorithm all of a node's local data memory is treated as cache, and access to a particular data point is made to the node(s) whose cache had it last frame. No implementations of this class of algorithms have been reported.

The main advantage of a dynamic distribution over a static one is that the injection bandwidth supported for each node remains constant for all system sizes. By matching the network and partition dimensions, and mapping neighboring image lattice subsets to neighboring nodes on the network, communication can be limited to *adjacent* nodes only. Adjacent nodes are defined as having a routing distance of one or zero along each dimension of the network between them. Network throughput between adjacent nodes is within a constant factor of nearest-neighbor throughput due to the bounded distance between nodes. Network throughput for adjacent-node communication is proportional to *n*.

4.2.1. Redistribution Costs

View changes must be bounded to ensure that data subsets migrate no farther than adjacent nodes. Figures 7 - 9 show experimentally measured redistribution sizes as a function of an incremental rotation about one or more axes. A 64^3 data set is transformed by the rotation angle given on the abscissa. Transformed data points that cross image lattice subset boundaries are counted towards redistribution. Figures 7 - 9 have best-case and worst-case rotations for slab, shaft, and block subsets on 1D, 2D, and 3D mesh topologies, respectively. Average node redistribution size is plotted, but the position of a node's image lattice subset relative to the axis of rotation affects the redistribution size at that node.

Slab distributions (Fig. 7) show an approximate doubling of

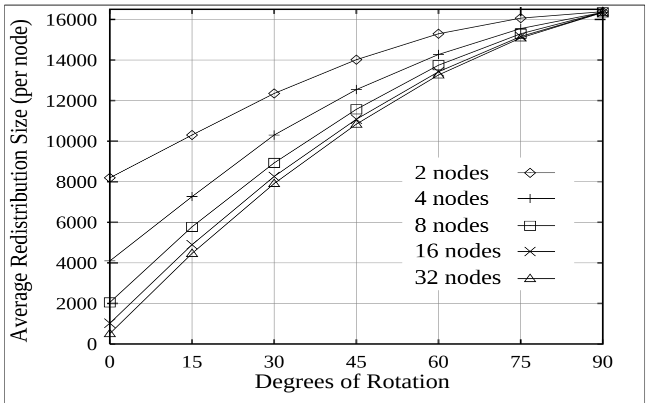

Fig. 10 - Redistribution with slab data subsets

average redistribution size between two and sixteen nodes. This is due to the fact that for *n* nodes, there are *n*-1 boundaries for data to migrate across. For large *n*, the average redistribution size remains constant. The downward curve in the sixteen node case is caused by a rotation angle large enough to cause data to migrate beyond adjacent regions. With slabs cut perpendicular to a single axis of rotation there is no redistribution. The 1D rotation data in figure 7 corresponds to rotation about an axis lying in plane of the slabs. The 2D rotation data in figure 7 is equivalent to 3D rotation and represents the worst-case redistribution size for a given angle applied successively about each axis.

Shaft (Fig. 8) and block (Fig. 9) distributions show a decrease in average redistribution size as *n* increases. In figures 8 and 9 the 1D and 3D rotations cause minimal and maximal redistribution size, respectively.

4.2.2. Observations

The redistribution size for a given rotation is proportional to the data size. When *d* and *n* increase proportionally, the net effect is still to increase the average redistribution size. For example, with a block distribution under 3D rotation, increasing the number of nodes from 8 to 64 decreases the average redistribution size to about 1/3 while the data size increases by a factor of eight producing a net factor of 8/3 increase at each node. In order to maintain a constant average redistribution size as *d* and *n* get larger, the rotation angle must decrease.

4.3. Object Partition

In object partition algorithms (Fig. 3) nodes compute images of their local data subset and redistribute the local images among themselves to combine them into a final image [Hsu93] [Ma93] [Cama93] [Chal91] [Yoo91]. The view point and aspect ratio of the data subsets affect the redistribution size. Slabs, shafts, and blocks vary from highly unbalanced aspect ratios to perfectly balanced ratios. As the view point changes, local images cover varying amounts of the screen, thereby varying the number of pixels moved in redistribution.

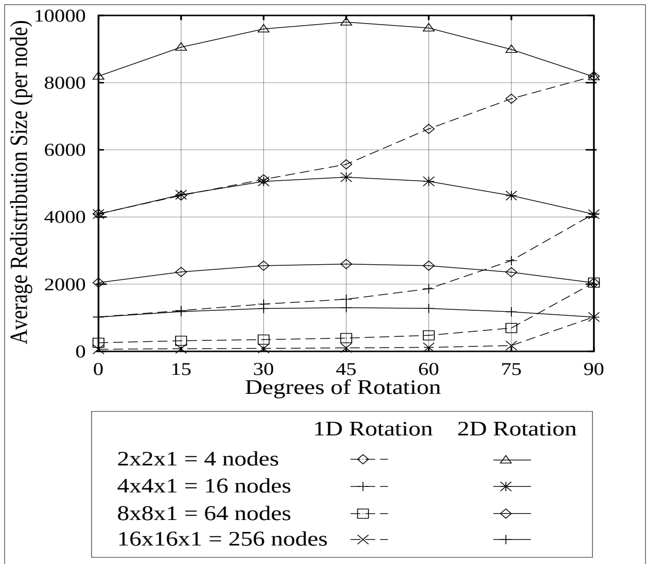

Fig. 11 - Redistribution with shaft data subsets

4.3.1. Data Distribution Shape

Figures 10 - 12 are graphs of the average, per-node, redistribution size for different data subsets over a range of rotation angles. These graphs are experimentally obtained using a 64^3 data size and a 128^2 screen size. Rays are traced through the data subsets and the number of subsets encountered is recorded. The aggregate number of data subsets the rays pass through is the minimum redistribution size. The view transformation is affine and formulated so that a rotation of zero degrees produces a full-screen image of the data. Based on the data subset orientations in figure 5, all 1D rotations (Figs. 10 - 12) specified by the abscissas are applied about the horizontal axis. The 2D shaft rotations (Fig. 11) create a worst-case by applying a constant 90° vertical axis rotation in addition to the variable horizontal axis rotation. The 3D block rotations (Fig. 12) create a worst-case by applying the abscissa angle equally about all three axes.

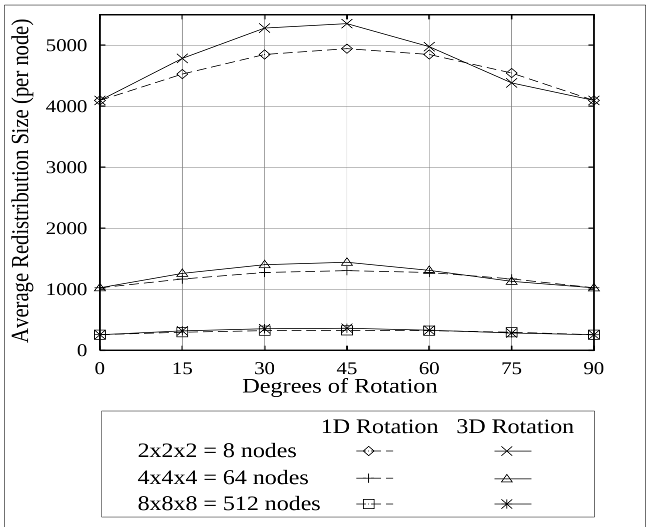

Fig. 12 - Redistribution with block data subsets

A block data distributions produces the lowest maximum redistribution size and achieve the most view-independence. The slab and shaft distributions have slightly lower best-case figures, but their strong view-dependence makes their worst-cases much higher. Therefore, blocks are considered the optimal data distribution.

4.3.2. Redistribution Costs

The local image size at each node in a block data distribution is approximately $p\ n^{-2/3}$ pixels, where p is the number of pixels in the final image. The local images must be combined properly to produce the final image. To achieve good load balance and network utilization, many small screen regions are assigned to each node in a random interleaved distribution. Approximately 1/n’th of each node’s local image pixels are composited into the same node’s assigned compositing regions so the total redistribution size is

$$m_{\text{redist}} \cong p\ n^{1/3}\ (1 - 1/n) \qquad (8)$$

Use of interleaved compositing regions also randomizes the redistribution network traffic, thereby matching the assumptions of the network performance model. The redistribution time for a 2D and 3D mesh under a normalized load of 0.3 is

$$\begin{aligned} t_{\text{2Dredist}} &\cong m_{\text{redist}} / (0.3\ n\ q_{\text{mesh}}) \\ &\cong p\ n^{1/3}\ (1 - 1/n) / (0.3\ n\ 4\ c / k) \\ &\cong p\ (1 - 1/n) / (1.2\ n^{1/6}\ c) \qquad (9) \end{aligned}$$

$$\begin{aligned} t_{\text{3Dredist}} &\cong p\ n^{1/3}\ (1 - 1/n) / (0.3\ n\ 4\ c / k) \\ &\cong p\ (1 - 1/n) / (1.2\ n^{1/3}\ c) \qquad (10) \end{aligned}$$

Toroidal topologies exhibit the same behavior except for a factor of four increase in network throughput. If the screen size p is held constant as the number of nodes increases, the redistribution size increases but the network throughput increases even faster so the time for redistribution actually decreases. Furthermore, since equations 9 and 10 are independent of d, both d and n may be increased without increasing the redistribution time. This behavior is better suited to large scalable systems than that of image partitions. Experimental verification is shown in section 5 with tests run on the Touchstone Delta.

The above analysis holds for a constant screen size. As data size increases it may be necessary to increase the screen size to prevent undersampling. Adopting the convention that $p^{1/2} = 2d^{1/3}$ causes the local image size at each node to become a function of the data size ($4\ d^{2/3}\ n^{-2/3}$) making the total redistribution size

$$m_{\text{redist}} \cong 4\ d^{2/3}\ n^{1/3} \qquad (11)$$

Substituting equation 11 into the expressions for redistribution time yields

$$t_{\text{2Dredist}} \cong 4\ d^{2/3}\ (1 - 1/n) / (1.2\ n^{1/6}\ c) \qquad (12)$$
$$t_{\text{3Dredist}} \cong 4\ d^{2/3}\ (1 - 1/n) / (1.2\ n^{1/3}\ c) \qquad (13)$$

When d and n are increased proportionately, these expressions exhibit the same asymptotic behavior as the image partition times given by equations 6 and 7, but for a given data set size, the redistribution time of an object partition is lower by a factor of $\sim(d/n)^{1/3}$ due to the local compositing that occurs before redistribution.

4.3.3. Observations

Redistribution costs for object partitions are much lower than for image partition algorithms, but there are disadvantages to object partitions in other respects. Load balance is difficult to maintain especially when the view point zooms in on a portion of the data set. Potentially, only one node’s data subset is visible making it responsible for rendering the entire image. There is a complementary case with image partition algorithms when the view point recedes so that much of the data falls into one node’s screen region. The application dictates the probability of either case occurring and therefore should influence the selection of a parallel algorithm.

Another drawback to object partitions is a loss of rendering efficiency. Nodes in an object partition have no knowledge of whether their data is obscured or not. Portions or all of a local image not seen in the final image represent wasted computation effort.

5. Network Performance on Touchstone Delta

The Touchstone Delta with its 2D mesh network is used to experimentally verify the predicted redistribution costs for object partitions. A test program is used to measure only the redistribution costs without including any rendering costs. The program computes the bounding rectangle of each node’s local image and pixels are redistributed according to an interleaved static assignment of screen regions. Pixels are received by the destination nodes, but compositing times are not included in the test times.

Region assignments are varied to test for sensitivity to any pattern of assignment. Twenty different assignments were tested and the variations in redistribution time are small (<20%) and not repeatable. These variations are thought to be due to network I/O traffic through the machine partition caused by other user’s programs. (The Delta supports multiple users in separate mesh partitions.)

An object partition with a block data distribution is mapped onto the smallest "near-square" 2D mesh with sufficient nodes. A square, or near-square mesh partition is used to maintain the largest bisection possible. The 3D to 2D map-

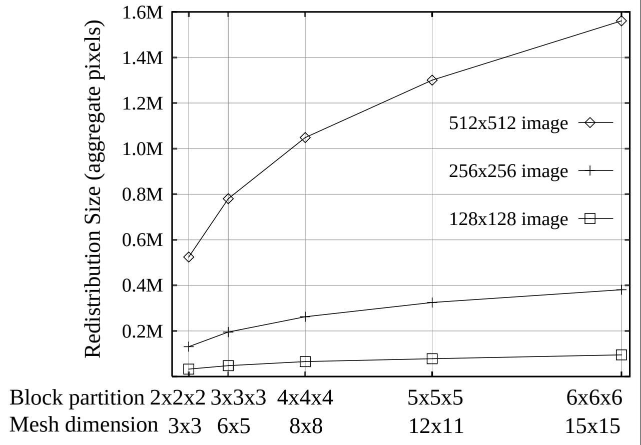

Fig. 13 - Redistribution sizes on the Touchstone Delta

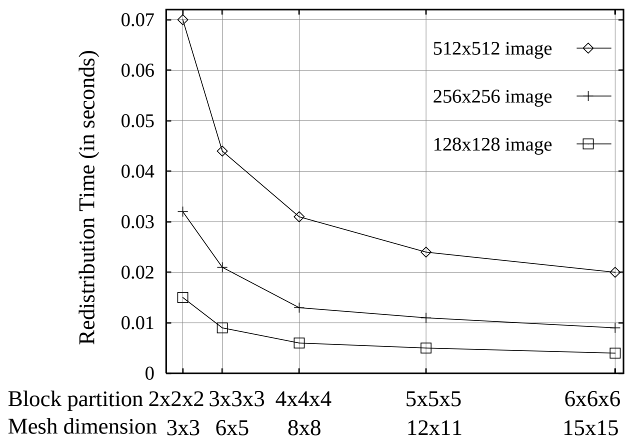

Fig. 14 - Redistribution times on the Touchstone Delta

ping is simply done by enumerating the partition blocks in x, y, z-order and assigning them to the corresponding partition node number. For example, a $2{\times}2{\times}2$ block partition fits into a $3{\times}3$ mesh with blocks $\langle 0,0,0\rangle$, $\langle 0,0,1\rangle$, … , $\langle 1,1,1\rangle$ assigned to nodes 0, 1, 2, …, 7, respectively. In this example case, the last node (node 8) is unused and doesn't contribute to the test.

Figure 13 shows the growth of redistribution size as the number of nodes increases for three fixed image sizes. Figure 14 shows the measured redistribution time shrinking over the same increase in the number of nodes. These two figures agree with the predicted behavior for object partitions and mesh networks — as *n* increases the redistribution size also increases, but the redistribution time decreases.

6. Volume Rendering on Touchstone Delta

An object partition volume rendering algorithm was implemented on the Touchstone Delta. The algorithm uses a block data distribution. Local images of test data are rendered by ray casting to provide perspective views (Fig. 15). The performance of this implementation is tabulated in figure 16 as the frame rates achieved for various data and system sizes. In all cases the screen size is 256^2 pixels. The data is analytically generated from Gaussian point and line sources sampled at different densities. The reader is referred to [Neum93] for further details about the isosurface shading and load balancing used in this implementation.

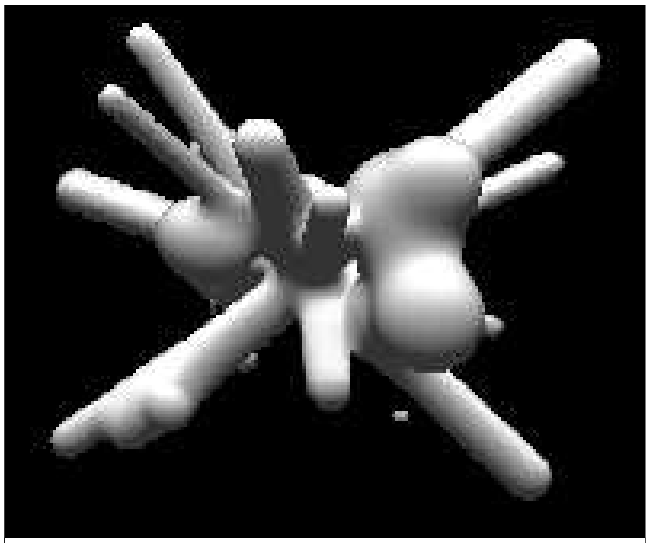

Fig. 15 - Isosurface rendering of test data

The Delta provides access to a frame buffer through an I/O node that feeds a HIPPI channel. Although the renderer assembles a complete 256^2 image in one node, it is not sent to the HIPPI frame buffer I/O node during these tests since updating the frame buffer limits the frame rate to about four Hertz for this image size.

6.1. Scaling

Performance scaling for this and other object partition implementations [Ma93] is lower than one might expect when nodes are added for a constant data size. This is partially due to the loss of rendering efficiency obtained from adaptive ray termination. Note the slower frame rate of the 4^3 system relative to the 3^3 system size and the low sensitivity to data size. At each node the implementation uses adaptive sampling, adaptive ray termination, and an octree encoding of the minimum and maximum data value in each octant. The effectiveness of the speedups vary for different

Data size	**System** 2^3	3^3	4^3	5^3	6^3
64^3	<u>1.8</u>	2.9	2.7	5.0	
128^3	1.6	2.6	<u>2.5</u>	4.2	5.1
192^3			2.3	4.1	<u>4.9</u>

Fig. 16 - Touchstone Delta rendering performance (frames per second)

data block sizes and numbers. Adaptive sampling is done by the isosceles-triangle recursive subdivision method [Shu91]. Nodes construct a separate octree for their data block. The octree "fit" of the features in the data varies with block dimension and placement. Adaptive ray termination only effects local image rendering, so as the depth complexity of the partition goes up and the data blocks get smaller its effectiveness diminishes. Although adaptive ray termination becomes less efficient as the number of blocks (and nodes) increases, its low overhead makes it worthwhile in all the cases tested.

Much better scaling is observed for the underlined cases (Fig. 16) where the data and system sizes are varied proportionally. In these cases each node is assigned a 32^3 data block and the system speed increases by more than a factor of two. This is at least partially due to the shrinking size of the local images as nodes are added.

The implementation illustrate a case where an object partition algorithm succeeds in reducing the redistribution costs to an insignificant level. A 6^3 system computes a 256^2 image in about 200 ms. The measured redistribution time in that case is under 10 ms. consuming under five percent of the total frame time.

7. Summary and Discussion

Parallel volume rendering algorithms inherently require communication of data. Parallel algorithms may be grouped into three classes each characterized by unique redistribution costs. Redistribution sizes for three classes of parallel algorithms are derived from analysis and simulations. A network model is used to estimate the time required for redistribution on mesh networks. The performance of object partition algorithms is verified by tests on the Touchstone Delta.

Further rendering speedups and hardware accelerators are clearly important areas of future research. A large portion of rendering time is consumed to reconstruct and resample the volume. Dedicated 3D texture hardware accelerates this process in new graphics systems. These systems provide high performance volume rendering through image partition parallelism, but the scalability of these systems for larger data and even higher performance is in question since current designs completely replicate the data at each processing node ($r = n$) to eliminate redistribution costs. This research raises the question of whether a more cost effective approach can be used to build general purpose or dedicated hardware capable of scaling to very large data sizes and high performance levels. In my opinion the answer is likely to lie to a highly parallel system with hundreds or perhaps thousands of simple processors. Based on their low redistribution costs, two algorithms seem likely contenders for such systems: image partitions with static data and large caches, or object partitions with block data distributions. Neither approach requires a high replication factor so memory is efficiently used, and a low bandwidth (low cost) mesh network is sufficient for either approach. The image partition supports rendering efficiencies but the caching adds complexity to the nodes. An object partition is less efficient at rendering, but its lower complexity and redistribution costs are better suited to systems with smaller, more numerous processors.

8. References

[Cama93] Emilio Camahort and Indranil Chakravarty. "Integrating Volume Rendering on Distributed Memory Architectures." *1993 Parallel Rendering Symposium*, 89-96, October 1993, ACM Proceedings.

[Chal91] Judy Challinger. "Parallel Volume Rendering on a Shared-Memory Multiprocessor." *Computer and Information Sciences, UC Santa Cruz*, Tech Report CRL-91-23, Revised March 1992.

[Corr92] Brian Corrie and Paul Mackerras. "Parallel Volume Rendering and Data Coherence on the Fujitsu AP1000." *Department of Computer Science, The Australian National University*, Tech Report TR-CS-92-11, August 1991.

[Hsu93] William M. Hsu. "Segmented Ray Casting for Data Parallel Volume Rendering." *1993 Parallel Rendering Symposium*, 7-14, October 1993, ACM Proceedings.

[Ma93] Kwan-Liu Ma, James S. Painter, Charles D. Hansen, Micheal F. Krogh. "A Data Distributed Algorithm for Ray-Traced Volume Rendering." *1993 Parallel Rendering Symposium*, 15-22, October 1993, ACM Proceedings.

[Mont92] C. Montani, R. Perego, and R. Scopigno. "Parallel Volume Visualization on a Hypercube Architecture." *1992 Workshop on Volume Visualization*, 9-16, October 1992. Workshop Proceedings.

[Neum93] Ulrich Neumann. "Volume Reconstruction and Parallel Rendering Algorithms: A Comparative Analysis." *Department of Computer Science, UNC at Chapel Hill*, Tech Report TR93-017, May 1993. Ph.D. Dissertation.

[Ngai89] John Y. Ngai. "A Framework for Adaptive Routing in Multicomputer Networks." *Department of Computer Science, California Institute of Technology*, Tech Report CS-TR-89-09, May 1989. Ph.D. Dissertation.

[Nieh92] Jason Nieh and Marc Levoy. "Volume Rendering on Scalable Shared-Memory MIMD Architectures." *1992 Workshop on Volume Visualization*, 17-24, October 1992. Workshop Proceedings.

[Shu91] Renben Shu and Alan Liu. "A Fast Ray Casting Algorithm Using Isotriangular Subdivision." *IEEE Visualization'*91, 232-237, October 1991. Conference Proceedings.

[Vézi92] Guy Vézina, Peter A. Fletcher, and Philip K. Robertson. "Volume Rendering on the MasPar MP-1." *1992 Workshop on Volume Visualizati*on, 3-8, October 1992. Workshop Proceedings.

[Yoo91] Terry S. Yoo, Ulrich Neumann, Henry Fuchs, Stephen M. Pizer, Tim Cullip, John Rhoades, Ross Whitaker. "Achieving Direct Volume Visualization with Interactive Semantic Region Selection." *IEEE Visualization'*91, 58-65, October 1991. Conference Proceedings.