9. Summary, Conclusions, and Future Work

9.1. Summary and Conclusions

This work focuses on two major issues effecting the design and implementation of parallel volume-rendering algorithms for multicomputers - the efficiency of reconstruction and redistribution. Reconstruction is a dominant cost of volume rendering in uniprocessor and parallel systems; using an efficient method is therefore important. Splatting is shown to be the most efficient reconstruction method when compared to ray casting and volume shearing. Redistribution is a cost incurred in all parallel volume-rendering algorithms. Object partitions are shown to minimize the redistribution cost and scale well.

The Introduction formulated analytical and algorithmic models of volume rendering. The algorithmic model is based on the embedded image and object lattices. This model is useful for deriving the reconstruction costs and classifying parallel algorithms. The error and efficiency of three reconstruction filters is analyzed. Reconstruction error, as a function of feature size, is analytically-derived and experimentally-measured from rendered images. Both results show that a separable cubic reconstruction filter provides the lowest resampling error-bound when compared to a pyramid or Gaussian filter. Separable cubic reconstruction filters are used in the volume-shearing rendering method. A simplified two-pass version of this method is shown to provide low error with significant time and memory savings over the normal three-pass approach. Implementations of splatting, volume shearing, and ray casting are tested on several workstations. Analysis of the rendering loops, and the test timing-results show that splatting is the most efficient rendering method in terms of computation cost per reconstructed point. Splatting also gets increasingly efficient as the image size increases relative to the data size. Ray casting with trilinear interpolation is neither the most accurate, nor the most-efficient method, but is the only rendering method that supports perspective view-transformations without considerable added cost. The many speedup techniques that can be used with ray casting complicate an implementation considerably, but they often lower the rendering cost to about, or below, that of splatting.

The latter part of this work considers the possible ways to parallelize volume-rendering. A new taxonomy is presented that enumerates the possibilities in terms of the distribution of image-lattice and object-lattice points and the rendering method. Many of the algorithm options have inherent disadvantages and are culled. The remaining options are discussed and analyzed with respect to their communication costs on different network-topologies and their suitability for multicomputers using message-passing or shared-memory communication models. Three classes of algorithms emerge and provide a useful grouping of algorithms in terms of their communication requirements. These classes are a) image partitions with static data distributions, b) image partitions with dynamic data distributions, and c) object partitions with contiguous block data distributions.

Image partitions with static data distributions have been implemented by several researchers. Image partitions with dynamic data distributions have not been reported, but their bounded redistribution-costs and scalability make them attractive. In these algorithms, data migrates between nodes as the view transformation changes incrementally. If the incremental view-changes are bounded and the dimensions of the partition and the communication network are the same, then the network utilization is within a constant factor of that achieved for nearest-neighbor communication.

Analysis and simulations show that object-partition algorithms with contiguous block data distributions maintain the lowest redistribution size of any algorithm, and for a fixed screen-size, they scale well on mesh topologies as the data and system sizes increase. Two implementations of an object-partition algorithm are presented and used to demonstrate a solution to the previously-unsolved problem of load balancing such an algorithm. The implementations on the Touchstone Delta and Pixel-Planes 5 perform as well or better than any reported results for message-passing systems, but they are limited by the reconstruction and resampling required to render images whose size is $\geq 256^2$. This strongly indicates an imbalance between the computation capabilities and network performance of current generation multicomputers when applied to volume rendering. Further software rendering-speedups and hardware accelerators are clearly an important area of future research.

9.2. Future Work

For truly-interactive frame rates with large data and screen sizes, faster rendering methods must be found. The SGI implementation described in section 2.2.5 clearly demonstrates the possibility of applying hardware acceleration to the resampling task. The PixelFlow system under development at the University of North Carolina at Chapel Hill also has hardware suitable for 3D texture mapping [Molnar$^+$92]. PixelFlow will overcome the data-size limitation of the SGI system since subsets of the data can be loaded into separate texture engines which operate concurrently. In the taxonomy, the PixelFlow approach is an object partition with a contiguous static slab data distribution. The image lattice is not distributed since the composition network is the compositor of all the local images. Either, nodes stream their local images onto the composition network in the view-order sequence required by compositing, or nodes may use a single compositing-sequence with the front-to-back or back-to-front compositing-computation applied selectively at each node as a function of the view direction.

Another approach to accelerating resampling is to use vector processing. Volume shearing is particularly suitable for this due to its regular memory-access strides. All the transformation computations are simply DDA increments. Computing the cubic filter coefficients is feasible but costly. Linear filter coefficients could be computed and applied more simply with some reduction in image fidelity.

Splatting is the fastest method, per reconstructed sample-point, of the three rendering methods tested. Some new speedups, inspired by this work, are offered here for making it even faster. Reducing the kernel extent clearly increases the reconstruction speed. Recall that the Gaussian extent was truncated at $\zeta = 1.3$ for the speed tests conducted in chapter five. If a pyramid filter is used instead of a Gaussian, the extent is reduced to 1.0 with a commensurate speed increase. Isosurface rendering with ray casting is accelerated by the use of min-max octrees to quickly locate possible surface locations. Octrees may do the same for splatting if a 3D splat buffer and a pyramid filter is used. Instead of a single image-lattice plane that has points splatted onto it from a single object-lattice plane, memory is allocated for as many image-lattice planes as there are object-lattice planes. Any point may now be splatted, in any sequence, onto its respective image-lattice plane. The octree is traversed in any order, skipping over any node whose min-max values do not bracket the desired threshold. Any node with min-max values bracketing the threshold contains the surface and causes a descent to the node’s children. If a leaf-node brackets the threshold, all data-points bounding in the octant are transformed and splatted into their respective image-lattice planes. Some tags must be kept to prevent splatting the points on the octant-boundaries more than once. After octree traversal is completed, pixel values in adjacent image-planes are tested, from front-to-back, for values bracketing the threshold. When bracketing values are found, the location of the surface may be interpolated, and classification and shading performed for that pixel. Further examination of image-lattice values behind the first surface encountered need not be done.

The Delta implementation could be improved. Since it is currently CPU-bound, further efforts at increasing the efficiency of the ray-casting code, perhaps by manual assembly coding, could be fruitful. Splatting could also be tried as a substitute for the ray-casting renderer.

The KSR-1 [Kendall] multicomputer nodes have large local-memories that are essentially all cache. Since multiple read-only copies of data may exist in these caches and the system supports a low-latency shared-memory communication model, the KSR-1 may be an ideal candidate for the first implementation of an image-partition algorithm with a dynamic data distribution.

The algorithms described in this work could be applied to some of the visualization packages like AVS, Explorer, and apE currently being parallelized for multicomputers.