Eurographics Symposium on Parallel Graphics and Visualization (2007)
Jean M. Favre, Luis Paulo dos Santos, and Dirk Reiners (Editors)

Parallel Reflective Symmetry Transformation for Volume Data

Y. Hong[†1] and H. W. Shen[1]

[1]the Ohio State University & Columbus OH, USA

Abstract

Many volume data possess symmetric features that can be clearly observed, for example, those existed in diffusion tensor image data sets. The exploitations of symmetries for volume data sets, however, are relatively limited due to the prohibitive computational cost of detecting the symmetries. In this paper we present an efficient parallel algorithm for symmetry computation in volume data represented by regular grids. Optimization is achieved by converting the raw data into a hierarchical tree-like structure. We design a novel algorithm to partition the tree and distribute the data among processors to minimize the data dependency at run time. The computed symmetries are useful for several volume data applications, including POF minimal opacity selection, transfer function generation and slice position selection.

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Symmetry; I.3.3 [Computer Graphics]: Volume Rendering; I.3.3 [Computer Graphics]: Parallel Computing

1. Introduction

Symmetry detection has been well studied since the 1980's, mainly focused upon calculating global symmetry with respect to planes passing through the center of an object. Recently, [PSG*06] introduced a symmetry transform that provides a continuous measure of the reflective symmetry of an object with respect to all planes. The resulting symmetry information can be applied to computer vision, computer graphics, medical image processing and other various areas. In [PSG*06] several examples were presented, including alignment of objects into a canonical coordinate system, geometric shape matching and optimal viewpoint selection. The redundant symmetry information can also be used to recover the missing data and object reconstruction [ZPA93].

In the field of volume visualization, symmetry information has not been widely utilized due to the large data sizes and hence the prohibitively expensive computation complexity incurred. The volume symmetry, however, can be very useful because many volume data intrinsically bear symmetric objects. One of the most obvious applications of volume symmetries is to speed up rendering. Since symmetry represents data redundancy to some extent, having the geometrically symmetric information in hand, we can only render half of a nearly symmetric volume object and display the another half reflectively. This is especially useful in the multiresolution rendering where the reflective portion can be reduced at a lower resolution. Knowing the volume symmetry can also be helpful to accelerate the computation of Plenoptic Opacity Function (POF) in the procedure of visibility culling [GHS*03].

The key problem for symmetry computation in volume data sets is to design an efficient algorithm, which is a non-trivial task. Today's scientific applications have generated high resolution, high dimensional data sets with large sizes that are beyond the resource capacity of a single PC or workstation. Although computer hardware has advanced very quickly recently, the large sizes of those data sets make it almost impossible to compute symmetry using only a single PC or workstation. To address this challenge, a viable solution is to utilize clusters of PCs to shift the computational

[†] Computer Science Department

burden from one machine to a group of loosely coupled computers .

In this paper, we present a parallel algorithm for symmetry computation with general volume data. We measure the reflective symmetry of an object with respect to each plane through its volume boundary [PSG*06]. Although researchers previously had proposed various methods for symmetry encoding and computing of large scale volumes on a single graphics workstation, fewer studies were focused on designing parallel algorithms for such a purpose using PC clusters. The contributions of our work are three-fold. First, we show the parallelization within the symmetry computation process. Second,we implement one parallel algorithm to make some performance analysis and propose several optimization methods to minimize both space and run-time computation overheads. Third, we investigate several possible applications in volume rendering. Figure 5 shows some volume datasets with primary reflective planes drawn in colors. In many cases the best slicing-cross section coincides with data's main reflective plane, as we can observed in Figure 1.

The rest of the paper is organized as follows. First, we review related work in Section 2. From Section 3 to Section 5, we describe our parallel symmetry computing algorithms, including the proof of existence of parallelization in symmetry computation, construction of the octree tree with hierarchical representation, data distribution with space-filling curve traversal. Results on parallel symmetry computation and load balancing among different processors are given in Section 6, and some concise descriptions of symmetry applications in volume visualization are introduced in Section 7. The paper is concluded in Section 8 with an outline of future work for our research. Section 9 is our acknowledgments.

2. Related Work

Most existing symmetry detection methods deal with discrete symmetries —perfect symmetry or imperfect symmetry under rotation, reflection, or translation. Many efficient algorithms have been designed to compute perfect symmetry. Atallah presented a substring matching algorithm [ATA85] to find perfect symmetry based upon the fact that a circular string is perfectly symmetric if it consists of two identical substrings. Ishikawa *et al.* [ISM*92] and Minovic *et al.* [MIK93] used an octree representation to find the symmetry of a 3D object. In [SS97], extended Gaussian images were used to detect symmetry of an object based on the idea that if an object is symmetrical, so is its extended Gaussian image. Generally speaking, these models merely work with perfect symmetry and only care about reflection about a given plane. Their computational complexities are relatively small, compared with symmetry measurement with respect to all the planes though the object.

Perfect symmetry, however, is rare in reality, especially with 3D objects. In few cases there is only one unique perfect symmetry plane through the object. Zabrodsky *et al.* [ZPA93, ZPA95] defined the *continuous symmetry distance* to quantify the degree of symmetry in an objects, which is the L_2 distance between the given shape and the smallest shape that is perfectly symmetric with respect to the same plane. Kazhdan *et al.* [KCD*03] extended this concept and defined a *shape descriptor* that calculates the symmetry of an object with respect to all the planes going through the center. Podolak *et al.* [PSG*06] considered the continuous symmetry with respect to all planes through the object's bounding volume. This extension greatly increases the computation complexity, which is up to $O(n^5 logn)$ even using the convolution. To improve the efficiency, Podolak *et al.* designed an efficient Monte Carlo sampling algorithm by exploiting sparsity in the data volume. Our work differs from previous research on symmetry computation in that we exploit parallelism in symmetry computation instead of relying upon stochastic method.

Nowadays the world is witnessing a rapid growth of data. It is normal to see a scientific program produces petabytes of data that was impossible several years ago. Therefore, to visualize large volumes people have utilized parallel computing to alleviate the burden incurred by the large data sets. In some cases, distributed clusters or parallelism are the only choice since no single machine can hold so much data.

Various parallel computing algorithms have been described for volume visualization. For example, previously researchers used a SIMD machine to speed up isosurface extraction [HH92]; a dynamic block distribution scheme for unstructured isosurface extraction [Ell95], and a parallel algorithm to render large scale particle systems [CA97]. Shen *et al.* [SHL*96] devised a parallel isosurface extraction algorithm based on span space subdivisions. Ma *et al.* [MPH*94] proposed a parallel algorithm that distributes data evenly to the available computing nodes and produces the final image using binary-swap composition. A parallelized shear-warp volume rendering algorithm was provided in [SL03]. Some other research [LMC02] achieved scalable volume rendering by utilizing lossy compression techniques to render time-varying scalar data sets.

3. Background

In this section, we first provide some background information for symmetry distance and very briefly explain how to exploit parallelism to speed up computation. Zabrodsky *et al.* [ZPA95] and Kazhdan *et al.* [KCD*03] have already given out the concept of symmetry distance, $SD(f,\gamma)$, with L_2 norm:

$$SD(f,\gamma) = \min_{g|\gamma(g)=g} \|f-g\|.$$

Here f is defined as a scalar-valued function and γ is the reflection plane. Symmetry distance, $SD(f,\gamma)$, describes a L_2

distance between f and the closest perfect symmetric function g. Kazhdan *et al.* [KCD*03] further simplified $SD(f,\gamma)$ by replacing the closest symmetric function g with the average of f and $\gamma(f)$:

$$SD(f,\gamma) = \|f - \frac{f+\gamma(f)}{2}\| = \frac{\|f-\gamma(f)\|}{2}. \quad (1)$$

Here $\gamma(f)$ is just the reflection of f with respect to γ. Podolak *et al.* proved in [PSG*06] that the calculation of the normalized symmetry distance, $\frac{SD^2(f,\gamma)}{\|f\|^2}$, can reduce to a series of dot products between f and $\gamma(f)$, if f is normalized too. Their method is essentially same with [KCD*03].

If function f is defined as a volume data $N \times N \times N$, f can be decomposed into a collection of concentric spheres all centered at the object center. The problem to measure reflective symmetry with respect to the reflection plane γ transfers to approximately calculating a series of symmetries of concentric spheres. That is:

$$SD(f,\gamma) \approx \sqrt{\sum_{r=0}^{N} SD^2(f_r,\gamma)}. \quad (2)$$

f_r is the function defined on the ball with radius r. Equation 2 naively shows the parallelism in symmetry computation—by assigning individual f_r to different processors a preliminary parallel algorithm is obtained. It can be seen that such an algorithm will not work efficiently because of the imbalanced work-loads at run-time. With the increasing radius r the data described in f_r becomes larger and larger, in an exponentially growing speed which is intolerable for most applications.

In order to smooth the noise and capture the imperfect symmetries we apply Gaussian Distance Transform (GDT) to the volume data when we compute the reflective volume symmetry. The GDT has the similar form as previously described in [KCD*03] and [PSG*06], but with different denotations:

$$GDT(x_1,x_2,M,\sigma) = e^{-D^2(x_1,x_2,M)/\sigma^2}, \quad (3)$$

where $D(x_1,x_2,M)$ is the difference between two values x_1 and x_2 in volume data M, σ is a user-defined coefficient to delimitate the Gaussian curve. Equation 3 is a Gaussian curve-like function that reaches the maximal value when x_1 and x_2 has the same values and gradually decreases when x_1 and x_2 differ from each other. In details, when computing reflective symmetry, the difference between values of point x and its reflection $\gamma(x)$ is smoothed by Equation 3 and be summed up to compute the final global symmetry.

The key problem is to compute $SD(f,\gamma)$ efficiently which is a challenging task in the traditional non-parallel environment. In Section 4 we discuss in details about our method to build a parallel implementation.

4. Algorithm Overview

Our parallel symmetry computation algorithm consists of three stages: preprocessing, compressed data compositing and run-time computing. In the preprocessing stage, we first distribute the data blocks among different processors along a hierarchical space-filling curve to maintain load balance. Section 5.1 introduces the statical data allocation technique along a Space-Filling curve that can improve load balancing. Then, for each processor, we build a hierarchical wavelet tree and compress the corresponding wavelet coefficients using a combination of run-length and Huffman encoding [KS99]. Section 5.3 shows the wavelet compression technique and explains how multiresolution data can be used for symmetry computation. A histogram of possible reflective points is calculated in this stage. Clusters in the histogram are used to determine which blocks will be frequently used in the subsequent stages and those blocks are allocated to each processor.

In the compressed data composite stage, the locally-built wavelet trees are sent back to the host node by using the binary swap algorithm introduced in [MJC*94]. The basic idea of binary swap is similar to that used in image composition in [MJC*94] except replacing the *over* operator with the *append* operator: pairing-up processors will exchange half of their local compressed wavelet blocks with each other that are selected by the user-defined multiresolution error tolerance. Each processor will add the received blocks from its partner to its own compressed wavelet block links. The main goal of this stage is to grow the local wavelet tree residing in each processor to reduce data request at run-time.

In the final stage, the processors compute the symmetry distances from the distributed data according to Equation 1. If the necessary data are not available $REQUEST/REPLY$ mechanism is called (see Section 5.4). The final global symmetry is generated by compositing the partially calculated symmetries at different processors. Several optimizations can be applied to our algorithm which will be discussed in the section 5.

4.1. Parameterizations

In our implementation, we use spherical coordinates to represent the reflection planes which can be parameterized by normals of the plane and the distances from the origin to those planes. When working with 3D, the normal $\hat{n}$ can be expressed as:

$$\hat{n} = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix}$$

where $\theta \in [0\ \frac{\pi}{2}]$ and $\phi \in [0\ 2\pi]$. The distance from origin to the plane is $r \in [0\ r_{max}]$.

4.2. Algorithms

In the following, we will propose our parallel algorithm for calculating symmetry distance of a $N \times N \times N$ volume dataset defined by function f. The following only describes the main work-flow of our parallel algorithm. Some details such as data exchanges and compression are described Section 5.

A brute-force algorithm is obviously a trivial solution: to calculate symmetry for every possible plane reflection γ for every point separately, following the three stages described above. The complexity of the brute-force algorithm for a p-processor parallel system, is $O(K^6 logK) + O(p)$, where $K = \frac{N}{p}$ and the second term of complexity, $O(p)$, stands for the communication and composition cost. If using convolution the complexity becomes $O(K^5 log^2 K) + O(p)$. It takes about 100 seconds in average to compute a volume data with $128 \times 128 \times 72$ grids.

In Algorithm 1 we use a multiresolution technique to speed up the symmetry calculation and sample the points according to the gradient magnitudes at those points, throwing away those points the gradient magnitudes are smaller than a user-defined threshold. The reason for importance sampling lies in the fact that, in a volume data, those points with smaller gradients most likely bear trivial importance, contributing little to the object's surfaces. Moreover, Since the wavelet tree built in the previous stage has a highly hierarchical nature we can utilize this by only partially reconstructing the block data f based upon the user-defined error tolerance. In details, we do not traverse the wavelet tree thoroughly from the root deep to the leaves during the retrieving procedure. Given a user-defined error tolerance we stop at a higher level of wavelet tree instead if the tolerance is satisfied by the previously calculated information saved in the wavelet tree nodes. The complexity of Algorithm 1, for a p-processor parallel system, is $O(K^5 C) + O(logp)$, where C is related to the error tolerance. The calculated symmetries then are interpolated to obtain higher accuracy by taking advantage of the continuity property showed in [PSG*06].

Algorithm 1 Algorithm: Multiresolution Method

```
1: Compress the distributed volume data into a local wavelet tree T_loc
2: Composite the global T_global by binary swap technique
3: for each plane γ do
4:    for each sampled point x do
5:       Retrieve f(x) from T_global, controlled by the user-defined error tolerance
6:       x' ← γ(x)
7:       Retrieve f(x') from T_global, controlled by the user-defined error tolerance
8:       D(f,γ)+ ← GDT(f(x), f(x'), γ)
9:    end for
10: end for
11: Composite the global symmetry by binary swap technique
```

5. Optimization Methods

This section introduces several optimization techniques used in Alg. 1.

5.1. Data Distribution in Space-Filling Curve

When designing a parallel algorithm, it should ensure that all the processors have an equal amount of workload at run time. However, when symmetry computation is performed, the processors with data blocks near the center will have heavier workloads than other processors, if data blocks are distributed in a spatially uniform way. This phenomenon was observed in [PSG*06]– portions of the model away from the center naturally have lower reflective symmetries since their reflective counterparts are outside the bounding volume and hence can be skipped (section 5.2). This imbalanced workload distribution needs to be avoided. The basic idea for our optimization method is to utilize the spatial coherence in volume data. In general, a volumetric data set usually exhibits strong spatial coherence. Kazhdan *et al.* [KCD*03] demonstrated that Equation 2 is stable even with the presence of high-frequency noise. It allows the objects to be slightly deformed so that imperfect symmetries can still be captured.

In our algorithm, a space-filling curve is utilized to assign the data blocks to different processors. The space-filling curve is used for its ability to preserve spatial locality, i.e., the traversal path along a space-filling curve always visits the adjacent blocks before it leaves the local neighborhood. Therefore, when data are distributed following the space-filling curve, in a consecutive round-robin manner, volume blocks will tend to be distributed evenly among the processors, breaking the spatial locality. Each processor only compute symmetries of the blocks statically assigned to it; such static data distribution is especially desirable when dealing with large-scale datasets. Moreover, the hierarchical property of a space-filling curve also makes it suitable to be applied to a hierarchical algorithm. Figure 1 shows how 16 2D blocks are traversed in a space-filling curve.

5.2. Bounding Data Representation

Another optimization method used to accelerate the computing procedure is the octree representation of data. Since we need to calculate reflective symmetry with respect to all planes through the object' bounding volume, for some planes, most parts of the object will be reflected to the outside of the object's bounding volume. It is unnecessary to compute the symmetry distances for those regions.

The octree is built in a bottom-up manner with bounding information stored in each node. To speed up the computation process, when computing the reflective symmetry, given

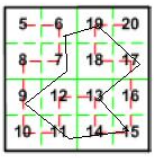

Figure 1: *A 2D example of data distribution along the space-filling curve. A Hilbert curve is used in this example. Data are visited along the red line. The black line stands for the possible 2D shape. Block numbers are centered in dash boxes. Since it is only one level of a hierarchical representation block numbers start from 5.*

a specific reflective plane γ, we hierarchically test γ with the node's bounding volume and skip those blocks if the testing results show that the reflected blocks are outside the object's bounding volume.

5.3. Wavelet Compression and Multiresolution Symmetry Computation

In this paper, we apply an efficient wavelet-based compression method [KS99] for volume data compression to save the storage space and reconstruction time at run-time. Here we have selected the Haar wavelet as a basic function that has a simple basis but is relatively easy to compute [KS99].

The procedure of a bottom-up blockwise wavelet-tree construction is standard. Starting with subdividing the volume data into a sequence of blocks, each 3D wavelet transform will produce a smoothing-filtered subblock and several wavelet coefficient subblocks. The smoothing-filtered subblocks from adjacent leaf nodes in the wavelet tree are then collected and grouped into a lower resolution data block in the wavelet hierarchy. We recursively apply this 3D wavelet transform and subblock grouping process until the root of the tree is reached. The wavelet coefficients associated with a tree node resulting from the 3D wavelet transform will be compared against a user-provided threshold and set to zero if they are smaller than the threshold. These wavelet coefficients are then compressed using run-length encoding combined with an encoder [KS99].

5.4. Reduce Data Dependency

Theoretically a geometric point in a volume data can have reflected counterparts anywhere in the bounding volume. However it is not practical to transfer the whole global wavelet tree to every processor, even at a high compression rate, due to the prohibitive size of volume data. It is also unreasonable to transport large amounts of data blocks merely on the fly for the same reason. In this paper, we apply a hybrid method to address this problem.

Given a point x its reflective counterpart $\gamma(x)$ can be known thereafter by:

$$\gamma(x) = x + 2r\hat{n} - 2(\hat{n}.x)\hat{n}$$

where the reflective plane γ is decided by the distance r and its normal $\hat{n}$.

In the preprocess stage we can calculate the histogram of locations of $\gamma(x)$ statically. Our strategy is to distribute those wavelet blocks whose containing $\gamma(x)$ are mostly used in symmetry computation to every processor. A heuristic method is to distribute the center parts of the volume data. This static data distribution scheme reduces the amount of requests and exchanges for data that are frequently needed at the parallel computation stage.

During the run-time computing stage, when a processor needs data at $\gamma(x)$ that it dose not hold, the computing flow continues but the unavailable $\gamma(x)$ is inserted into a request queue *Req_Queue* which contains all those unavailable $\gamma(x)$. At a later synchronization point the Req_Queue is broadcasted and the requested processor continues its remaining computation jobs. Those processors holding the needy $\gamma(x)$ will response and send back the corresponding $\gamma(x)$ which will be accepted into a *Recv_Queue* by the requesting processor.

Since the data request time and the computation time is overlapped the only overheads are the real transportation time. We use a multi-thread technique to deal with $REQUEST/REPLY$ issues which is supported by MVAPICH. Specifically, two extra threads, *send_thread* and *recv_thread*, are invoked after the parallel initialization stage: *send_thread* monitors *Req_Queue*. If *Req_Queue* is full *send_thread* broadcasts it. *recv_thread* monitors *Recv_Queue*. If *Recv_Queue* is full *Recv_thread* tells the main thread to fetch available $\gamma(x)$.

6. Results

In this section, we present the experimental results of our parallel symmetry detection algorithm running on a PC cluster consisting of 64 compute nodes, 6 storage nodes and one front end. Each compute node is a dual processor Opteron 250 (single core) with 8GB of RAM and 2×250GB SATA disk. MVAPICH based on MPICH $(MPI-1)$ is used for the module mpi and the Infiniband network for MPI communication. The datasets in Figure 1 are a Lobster in $301 \times 324 \times 56$, a Frog in $256 \times 256 \times 44$, a Teapot in $256 \times 256 \times 178$ and a Man's Leg in $341 \times 341 \times 93$ volumes. Our main test datasets include a $256 \times 256 \times 145$ UNC brain dataset and a $512 \times 512 \times 1728$ Visible Woman dataset. All tests were run using 22, 36 or 64 nodes of the cluster.

Static data distribution along a space filling curve gives our parallel symmetry computation algorithm a balanced workload. In Figure 2, the small variation of the computing times used by each of the 36 processors shows that, with a Hilbert space-filling curve implemented, our algorithm can

achieve better load balancing than the algorithm without it. This implies good scalability for our parallel symmetry computation algorithm.

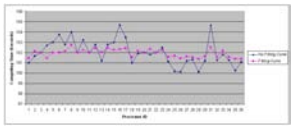

Figure 2: *The computation time via seconds (for Brain dataset) on each of* 36 *processors: the blue line indicates the algorithm without space filling-curve data distribution and the red line indicates algorithm with space filling-curve data distribution.*

Figure 3 gives the speedup factors obtained using different number of processors. Multiresolution Algorithm and brute-force one are computed against their sequential counterparts respectively. It is clearly shown that our multi-resolution algorithm achieves better speedup performance than the brute-force algorithm. About 87% and 81% parallel utilization were observed for 16 and 32 processors, respectively.

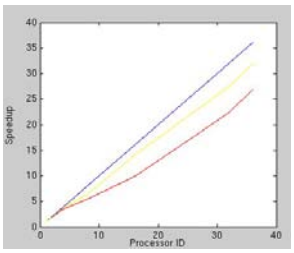

Figure 3: *Speedup factors (for Brain dataset) of our algorithms when using* 1, 2, 4, 8, 16, 32 *and* 36 *processors. The blue line is the ideal speedup , the red line is the speedup using no multiresolution method and the yellow line the speedup using multiresolution method with error tolerance* 1500.

Our multiresolution algorithm is based on the observation that symmetry held in data at high resolution data will persist in the lower resolution data too. Figure 4 shows the symmetries calculated by Algorithm 1 with different error tolerances. With higher errors, i.e., data at higher resolution, the resulting symmetry is closer to the symmetry calculated by Brute-force algorithm.

Table 1 shows different average computation times under three different error tolerances by utilizing multiresolution optimization. Figure 6 and Figure 7 show the calculated primary symmetric planes for the tests described in table 1. The primary symmetric planes are very close to each other. Considering the large amount of time saved, our multiresolution symmetry computation algorithm can attain a good effect if the user-defined error tolerance is properly selected.

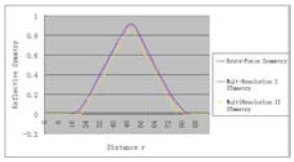

Figure 4: *The calculated reflective symmetries (for Brain dataset) for one direction [*$\theta = 0.1745\phi = 0.1745$*]. The blue line using brute-force algorithm and the red line and yellow line using multiresolution algorithm. Red line has a higher resolution.*

7. Applications

Many previous researches have introduced various symmetry applications. Here we only present a few novel ones that are useful in volume visualization. POF calculation [GHS*03] requires minimal integral opacities for each volume block along the viewpoint ray direction which can be solved by checking the minimal local symmetry of that block with respect to the reflection perpendicular to the view direction; For some medical volume data reflective symmetries can be applied to find reasonable transfer functions based on the idea that the more an isovalue contributes to the final symmetry the higher the possibility of its being on the surface of an object, hence the larger opacity it might bear; It might be an optimal choice to slice a volume data along the plane with highest calculated symmetries: revealing more inner information since most primary symmetric planes go through the object's center.

8. Conclusion and FutureWork

We present a parallel reflective symmetry computation algorithm utilizing wavelet-tree and space-filling curve optimizations. We show that the algorithm is efficient and stable for large volume datasets. Our experiments also show that, for most of the volume data, the primary reflective symmetric planes are going through the centers of the objects, a reasonable phenomenon that coincides with people's common sense. In the future work we hope to extend our algorithm to

Dataset	processors	Test1	Test2	Test3
Viswoman	64	1256	745	205
Brain	36	96	34	11
Frog	22	35	18	

Table 1: *Computation times (for Viswoman) in seconds with multiresolution optimization under three different error tolerances. In all the three cases,* 64 *processors are used for Viswoman,* 36 *for Brain dataset and* 22 *for Frog dataset. Tests are arranged for error tolerances of 5,000, 1,000, and 500 respectively (a higher value represents a higher resolution).*

include other types of symmetry. For example, the rotational symmetry is one of possible areas need to be considered. We also notice that further optimization of symmetry computation is possible if we can transform a 3*D* problem into several of 2*D* subproblems which can be solved quickly in parallel environment.

9. Acknowledgements

This wok was supported in part by NSF ITR Grant ACI-0325934, NSF RI Grant CNS-0403342, NSF Career award CCF-0346883, and DOE SciDAC DE- FC02-06ER25779. The Visible Woman dataset is provided by the National Library of Medicine. The Brain dataset is a partial copy of datasets in the "University of North Carolina Volume Rendering Test Data Set" archive. The Frog dataset is copied from Information and Computing Sciences Division, Lawrence Berkeley Laboratory, the Teapot from Terarecon Inc, the Leg dataset from German Federal Institute for Material Research and Testing (BAM), Berlin, Germany and the Lobster dataset from VolVis distribution of SUNY Stony Brook, NY, USA.

References

[ATA85] ATALLAH M.: On symmetry detection. *IEEE Trans. on Computers 34*, (1985), pp. 663–666.

[CA97] CROSSNO P., ANGEL E.: Isosurface extraction using particle systems. In *Proc IEEE Visualization '97* (1997), pp. 495–498.

[CDF*03] CAMPBELL P. C., DEVINE K. D., FLAHERTY J. E., GERVASIO L. G., TERESCO J. D.: Dynamic Octree Load Balancing Using Space-Filling Curves. Tech. Rep. CS-03-01, Williams College Department of Computer Science, 2003.

[Ell95] ELLSIEPEN P.: Parallel isosurfacing in large unstructured datasets. *Visualization in Scientific Computing '95* (1995), pp. 9–23.

[GHS*03] GAO J., HUANG J., SHEN H. W., KOHL J.: Visibility Culling Using Plenoptic Opacity Function for Large Scale Data Visualization. In *IEEE Visualization 2003 '03* (2003), pp. 341–348.

[HH92] HANSEN C., HINKER P.: Massively parallel isosurface extraction. In *Proc IEEE Visualization '92* (1992), pp. 189–195.

[ISM*92] ISHIKAWA S., SATO K., MINOVIC P., KATO K.: An interactive 3D symmetry analysis system. in *IAPR Workshop on Machine Vision Applications*, (Dec 1992), pp. 375–378.

[KCD*03] KAZHDAN M. AND CHAZELLE T. AND DOBKIN D. AND FUNKHOUSER T. AND RUSINKIEWICZ S. : A reflective symmetry descriptor for 3D models. *Algorithmica*, 38,1 (Oct. 2003).

[KS99] KIM T. Y., SHIN Y. G: An Efficient Wavelet-Based Compression Method for Volume Rendering. In *Proc. of Paciffic Graphics '99*, (1999), pp. 147–157.

[LM94] LACROUTE P., MARC L.: Fast volume rendering using a shear-warp factorization of the viewing transformation. In *Proc ACM SIGGRAPH '94* (1994), pp. 451–458.

[LMC02] LUM E., MA K., CLYNE J.: A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Trans. on Visualization and Computer Graphics 8*, 3 (2002), pp. 286–361.

[MJC*94] MA K. L., JAMES S. P., CHARLES D. H., MICHAEL F. K.: Parallel volume rendering using binaryswap compositing. *IEEE Computer Graphics and Applications*, 14(4), (1994), pp. 59–68.

[MIK93] MINOVIC P., ISHIKAWA S., KATO K.: Symmetry identification of a 3D object represented by octree. *IEEE Trans. on Pattern Analysis and Machine Intelligence 15*, 5 (May 1993), pp. 507–514.

[MPH*94] MA K. L., PAINTER J. S., HANSEN C. D., KROGH M. F.: Parallel Volume Rendering Using Binary-Swap Compositing. *IEEE Computer Graphics and Applications 14*, 4 (1994), pp. 59–68.

[PSG*06] PODOLAK J. AND SHILANE P. AND GOLOVINSKIY A. AND RUSINKIEWICZ S. AND FUNKHOUSER T.: A planar-reflective symmetry transform for 3D shapes. In *Proc. SIGGRAPH '06* (Jul. 2006), vol. 5.

[SHL*96] SHEN H. W., HANSEN C. D., LIVNAT Y., JOHNSON C. R.: Isosurfacing in Span Space with Utmost Efficiency (ISSUE). in *Proc. IEEE Visualization '96*, 2 (1996), pp. 287–294.

[SL03] SCHULZE P., LANG U.: The Parallelized Perspective Shear-Warp Algorithm for Volume Rendering. *Parallel Computing 29*, 3 (2003), pp. 339–354.

[SS97] SUN C., SHERRAH J.: 3D symmetry detection using the extended Gaussian image. *IEEE Trans. on Pattern Analysis and Machine Intelligence 2*, 2 (Feb 1997), pp. 164–168.

[ZPA93] ZABRODSKY H., PELEG S., AVNIR D. A.: Completion of occluded shapes using symmetry. In *Proc CVPR*, (1993), pp. 678–679.

[ZPA95] ZABRODSKY H., PELEG S., AVNIR D. A.: Symmetry as a continuous feature. *Trans. PAMI 17*, 12 (1995), pp. 1154–1166.