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

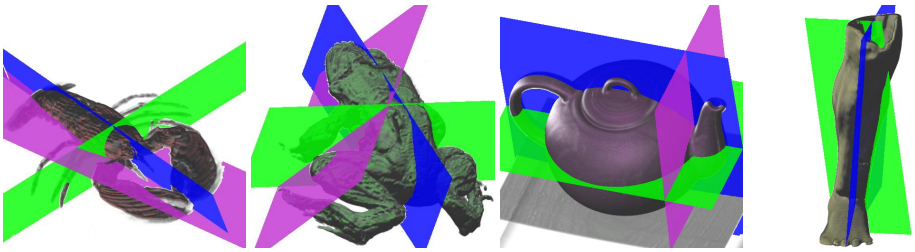


Figure 1: *3D volume data with symmetric features. The primary reflective planes are drawn in blue and some other orthogonal planes are used to illustrate the visibility of symmetry continuity.*

Abstract

Many volume data possess symmetric features that can be clearly observed, for example, those existing 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 use 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 optimal view selection and slice position explorion.

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 for 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

† Computer Science Department

another half reflectively. This is especially useful in the multiresolution rendering where the reflective portion can be reduced to 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 a volume data set is to design an efficient algorithm, which is a non-trivial task. Today's scientific applications are capable of generating high resolution, high dimensional data sets with large sizes. Although computer hardware has advanced, the data sizes have grown faster making it almost impossible to compute symmetry using a single PC or workstation. To address this challenge, a viable solution is to utilize parallelism to speedup the computational time.

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 have proposed various methods for symmetry encoding and computation for large scale volumes, few studies have focused on designing parallel algorithms for such a purpose. The contributions of our work are three-fold. First, we utilize a basic parallelization process for symmetry computation. Second, we propose several optimization methods to minimize both space and run-time computation overheads. A *Monte Carlo* method is applied to quickly estimate the trends of symmetries and eliminate unnecessary computation for many non-symmetric planes. We use this method to optimize our performance rather than calculating the actual symmetries. Third, we investigate several possible applications in the context of volume rendering. Figure 1 shows several volume datasets with primary reflective planes drawn in colors. In many cases the best slicing-cross section coincides with data's main reflective plane.

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 proof of existence of parallelization in symmetry computation, construction of the octree tree with hierarchical representation, and data distribution with space-filling curve traversal. Section 6 presents our adaptive planar sampling scheme using a Monte Carlo method. Results on parallel symmetry computation and load balancing among different processors are given in Section 7. Concise descriptions of symmetry applications in volume visualization are introduced in Section 8. The paper is concluded in Section 9, with an outline of future work for our research.

2. Related Work

2.1. Symmetry Applications

Many graphics applications can use symmetry information. The alignment problem is one of the symmetry applications which was mentioned in [PSG*06]. Alignment of objects into a canonical coordinate frame is a task, especially useful in volume visualization, view selection and image composition. To perform volume alignment, an origin is selected and a set of axes are chosen to establish the dimensional orientations. Better alignment will increase users' understanding of the visualized volume data. In medical visualization doctors prefer to align volume objects in a natural way. Volume data, however, usually is visualized at the origin of the data center and aligned by data principal axes. In some cases the centers of datasets are misaligned with the centers of volume objects which makes it difficult to adjust the view-direction. The Lobster is such a case. Symmetry information can be applied to search for optimal alignment. [PSG*06] introduced *principal symmetry axes* and *center of symmetry*: three perpendicular planes with maximal symmetries form the three principal axes and the origin.

Cross sections are used to explore the internal structures of volume objects. Generally the main intersection plane should be chosen across an object's center of mass and should lie along the principle symmetric plane. It is also reasonable to place further cross-sections along perpendicular symmetric planes on the volume data. This can be solved by application of symmetry alignments. Dual-rendering technique can be used to display contrasting visual information along the symmetric cross-sections. Section 8.1 will discuss this issue in detail.

As geometric properties of symmetric features are essentially related to objects' shape, they can be used to construct shape descriptors. In a volume database system objects in the dataset are classified by their global symmetries. This is based on the observation symmetry properties are often consistent within a class of objects [PSG*06]. Volume segmentation is fulfilled by comparing objects' local symmetries: m local maxima planes in symmetry are firstly selected; Then for each voxel the contributions for each of m local maximal planes are calculated and built into a feature vector; Finally clustering these feature vectors according to their proximities in the m dimensional feature space.

2.2. Symmetry Computation

Most existing symmetry detection methods deal with discrete symmetries; perfect symmetry or imperfect symmetry under rotation, reflection, or translation. 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 work only with perfect symme-

try 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. Only in a few cases is there only one unique perfect symmetry plane through an object. Zabrodsky *et al.* [ZPA93, ZPA95] defined the *continuous symmetry distance* to quantify the degree of symmetry in objects, which is the L_2 distance between a given shape and the smallest shape that is perfectly symmetric with respect to the same plane. Kazhdan *et al.* [KCD*03] extended this concept to define a *shape descriptor* that calculates the symmetry of an object with respect to all the planes going through the center. Kazhdan *et al.* applied the Fast Fourier Transform to compute the convolution for all angles that define corresponding reflective planes. They also used Fourier Decomposition to create a multiresolution approximation based on the fact that global features will be symmetric in low resolutions. Their method is efficient in computing the descriptor from a volumetric representation. However, there isn't necessary for parallelization of the computation to obtain scaling performance.

Extending [KCD*03], 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 convolution. To improve the efficiency, Podolak *et al.* designed Monte Carlo sampling algorithm by exploiting sparsity in the data volume. Our work differs from this research in that we exploit parallelism in symmetry computation instead of relying upon stochastic methods.

The scientific and visualization community is witnessing a rapid growth of data. It is normal to see a scientific simulation producing petabytes of data.To visualize large volumes, computer scientists have utilized parallel computing to alleviate the burden incurred by the large data sets. In some cases, parallelism is the only valid choice since no single machine can process so much data in a reasonable amount of time.

Various parallel computing algorithms have been described for volume visualization. For example, researchers have 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. Optimizations, including visibility culling, were also introduced into parallel environment to further reduce the rendering time of large-scale data sets [GS01, GHS*03].

3. Background

In this section, we provide background information for symmetry distance and briefly explain how to exploit parallelism to speed up computation. Zabrodsky *et al.* [ZPA95] and Kazhdan *et al.* [KCD*03] described the concept of symmetry distance, $SD(f,\gamma)$, using the 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 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. Their method is essentially the same as [KCD*03].

If function f is defined as volume data $N \times N \times N$ points, f can be decomposed into a collection of concentric spheres, all centered around the object center. The problem to measure reflective symmetry with respect to the reflection plane γ translates 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 exponentially, which is intolerable for most applications.

Kazhdan *et al.* [KCD*03] mentioned that the function f should be transformed to a volumetric representation that describes not only the space locations of points but also the

distance to the object defined by the function f. It is unnecessary in our work since we start with the volume data directly. In order to smooth the noise and capture the imperfect symmetries, we apply a 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 a different notation:

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

$D(x_1, x_2, M)$ is the difference between two values x_1 and x_2 in volume data M, and σ is a user-defined coefficient to limit 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 value and gradually decreases when x_1 and x_2 differ from each other. When computing reflective symmetry, the difference between values of point x and its reflection $\gamma(x)$ is smoothed by Equation 3 and is summed together to compute the final global symmetry.

The key problem is to compute $SD(f, \gamma)$ efficiently. In Section 4, we discuss in detail the method for a parallel implementation.

4. Algorithm Overview

Our parallel symmetry computation algorithm consists of three stages: preprocessing, compressed data compositing and run-time computation. In the preprocessing stage, we distribute the data blocks among different processors using a hierarchical space-filling curve to maintain load balance. Section 5.1 introduces the statical data allocation technique along a space-filling curve that improves 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 composition 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 is similar to that used in image composition in [MJC*94], except we replace the *over* operator with the *append* operator. The main goal of this stage is to grow the local wavelet tree residing in each processor to reduce data request at run-time. Pairing-up processors will exchange half of their local compressed wavelet blocks with each other. Exchanging blocks are selected by a user-defined multiresolution error tolerance. Each processor will add the received blocks from its partner to its own compressed wavelet block links.

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 planes and the distances from the origin to those planes. When working in 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 propose our parallel algorithm for calculating symmetry distance of a $N \times N \times N$ volume dataset defined by function f. The following describes the main work-flow of our parallel algorithm. Some details, such as data exchanges and compression, are described in Section 5.

The brute-force algorithm is a trivial solution to calculate symmetry. For every possible plane reflection γ for every point separately, we execute the three stages described previously. 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 approximately 100 seconds on average to compute symmetry on volume data with $128 \times 128 \times 72$ grid.

In Algorithm 1, we use a multiresolution technique to speed up the symmetry calculation. Points are sampled according to their gradient magnitudes. Points with gradient magnitudes smaller than a user-defined threshold are thrown away. Our assumption is that points with smaller gradients most likely bear trivial importance, contributing little objects' surfaces. Furthermore, we utilize the wavelet tree by partially reconstructing the block data f based upon the user-defined error tolerance. In detail, we do not reconstruct data stored in the wavelet tree completely from the root down to the leaves during the retrieval. We stop reconstruction at a higher level of wavelet tree if the error-tolerance is satisfied. 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 are interpolated to

obtain higher accuracy by taking advantage of the continuity property shown 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
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
```

5. Optimization Methods

This section introduces several optimization techniques used in Algorithm 1.

5.1. Data Distribution in Space-Filling Curve

For efficiency in 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 a model away from the center naturally have lower reflective symmetries since their reflective counterparts are outside the bounding volume and can be skipped (section 5.2). This imbalanced workload distribution should 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, such that the traversal path along the curve always visits the adjacent blocks before it leaves the local neighborhood. Data are distributed to processors following the space-filling curve in a consecutive round-robin manner. Volume blocks will then tend to be distributed evenly among the processors, breaking the spatial clumping on few processors. Each processor compute symmetries of the blocks statically assigned to it. 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 hierarchical data. Figure 2 shows how 16 2D blocks are traversed in a space-filling curve.

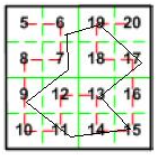


Figure 2: *A 2D example of data distribution along the Hilbert space-filling curve. Block numbers are centered in dash boxes. Data are distributed to processors in round-robin along the red line. The black line stands for the possible 2D shape.*

5.2. Bounding Data Representation

For some symmetry planes, parts of the object will be reflected outside of the object's bounding volume. It is unnecessary to compute the symmetry distances for those regions. Figure 3 illustrates the situation when some of the data are reflected out of the bounding box.

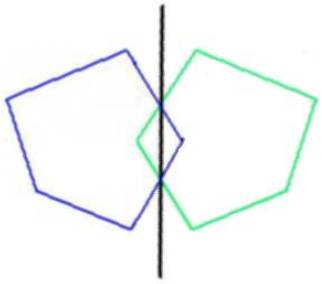


Figure 3: *A 2D example of object (in blue line) that is partially reflected (green line) out of its bounding shape.*

An optimization method used to accelerate the computation is the octree organization of data. The octree is built in a bottom-up manner, with bounding information stored in each node. To speed up the computation, when computing the reflective symmetry, given a reflective plane γ, we hierarchically test blocks with γ and the node's bounding volume. If the testing results show that reflected blocks are outside the object's bounding volume we skip those blocks.

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

storage space and reconstruction time at run-time. We have selected the Haar wavelet because it has a simple basis which is relatively easy to compute [KS99].

Our 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 an average-filtered subblock and several wavelet coefficient subblocks. The average-filtered subblocks from adjacent leaf nodes in the wavelet tree are grouped into a lower resolution data block in the wavelet hierarchy. We recursively apply the 3D wavelet transform and subblock grouping process until the root of the tree is reached. The wavelet coefficients are compared against a user-provided threshold and set to zero if they are smaller than the threshold for lossy compression. The wavelet coefficients are compressed using run-length encoding combined with a Huffman encoder [KS99].

5.4. Data Dependency Reduction

Theoretically a point in volume data can have reflected counterparts anywhere in the bounding volume. However, it is not practical to transfer the entire wavelet tree to every processor, even at a high compression rate, assuming large data sizes. It is also unreasonable to transmit large amounts of data blocks on the fly, for the same reason. We apply a hybrid method to address the data dependency problem.

Given a point x its reflective counterpart $\gamma(x)$ is known hereafter as:

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

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

In the preprocess stage, we can calculate the histogram of locations of $\gamma(x)$ statically. Our strategy is to distribute the wavelet blocks most used in symmetry computation to every processor. A heuristic is used 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 during parallel computation.

When a processor needs data $\gamma(x)$ in a block that it does not hold, the computation continues, but the unavailable $\gamma(x)$ is inserted into a request queue, *Req_Queue*, which lists unavailable $\gamma(x)$. At synchronization, *Req_Queue* is broadcasted, and the processor continues its remaining computation. Processors holding the needed data will respond and send back the blocks corresponding to $\gamma(x)$, which will be accepted into *Recv_Queue* by the requesting processor.

The data request and the computation is overlapped in time by using a multi-threading to deal with *REQUEST*/*REPLY* issues, two extra threads, *send_thread* and *recv_thread*, are invoked: *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* informs the main thread to fetch available $\gamma(x)$.

6. Adaptive Planar Sampling Method

Two kinds of uniform samplings are used in Algorithm 1. One is spatial voxel sampling and the other is plane sampling. In the following section we will discuss uniform planar sampling and its performance downsides.

6.1. Uniform Planar Sampling

To perform uniform planar sampling, we first parameterize the planes over (r, θ, ϕ) (see 4.1 for details), and sample the planar space uniformly along the three axes. The pseudocode for planar sampling is:

Algorithm 2 Uniform Planar Sampling

```
1: for r_min To r_max do
2:   for θ=0 To π/2 do
3:     for φ=0 To 2π do
4:       Sample voxels with respect to plane (r, θ, φ)
5:       Perform symmetry computation on plane
         (r, θ, φ)
6:     end for
7:   end for
8: end for
```

Once a plane is fixed, the corresponding spacial voxel sampling is performed. The optimization methods described previously are mainly focused on voxel sampling, including octree representations, empty space avoidance, and data compressions. There has not been any optimization thus far in the planar sampling space. However, many planes can usually be easily rejected to avoid unnecessary computation.

6.2. Inefficiencies in Uniform Planar Sampling

We first point out two potential inefficiencies if we simply apply uniform planar sampling. The first one is the redundancy of numerous of low-symmetry planes. Fig. 4 shows a histogram of computed symmetries for the lobster dataset in $301 \times 324 \times 56$. There are a total of 1238591 planes sampled. We notice that most of the planes (around 85%) are in low symmetry while only a very small potion of planes bear relatively large symmetries (around 2%). Much redundant computation time was spent on those low symmetry planes. The red line in Fig. 4 indicates the approximate computation time for each bin.

Another problem is that uniform planar sampling can miss primary symmetry planes as illustrated in Fig. 5(a). The green object is uniformly planar sampled by blue planes. The red plane represents the primary symmetry plane that is missed between two blue sampled planes. Many blue planes

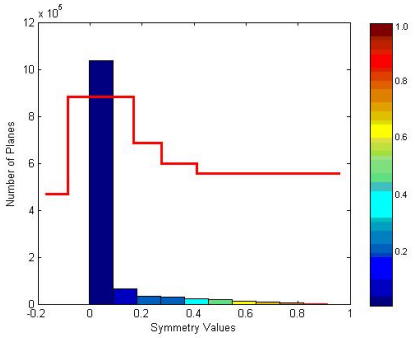
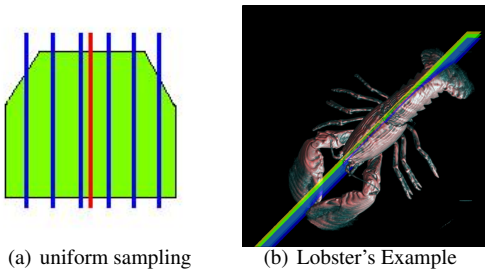


Figure 4: *A histogram of symmetries for Lobster (*$301 \times 324 \times 56$*).The colorbar on the right side represents the different symmetry spans from low to high. The red line depicts the computation time trend for the bins.*

are only related to unsymmetrical potions, leading to unnecessary computation. Fig. 5(b) illustrates the primary plane for the lobster dataset. The green primary plane lies between two parallel planes (yellow and blue). Although plane interpolation can compensate to some extent, it is not guaranteed that interpolations will work for all data sets. Increasing the sampling rates can alleviate this problem but will incur much more computation overhead as a result. For example, in Fig. 5(b) the step value for r is set to 3, while the exact value for the primary plane is $r = 201.5$. So uniform plane sampling will miss the primary plane unless the r step value is adjusted to 0.5 which would increase the computation cost six fold.

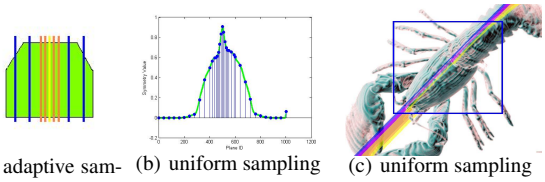


(a) uniform sampling (b) Lobster's Example

Figure 5: *(a) shows uniform sampling (blue) can miss sampling the primary plane (red) while wasting unnecessary sampling on low symmetric potions (green). (b) shows the Lobster dataset is sampled with step* $= 3$ *for plane parameter r. The true green primary plane occurs at* (r, θ, ϕ)*=(201.000000 1.570796 0.785398) while two sampled planes are adjacent, missing the true primary plane.*

6.3. Adaptive Planar Sampling

We adopt adaptive methods to save unnecessary computation and to increase symmetry precision.The concept of adaptive planar sampling is similar to importance sampling. We give more consideration to planes that are most likely to have higher symmetries than other planes. Fig. 6(a) illustrates this idea. The center of the 1 green objects has more symmetry than its sides. So, more samples (red ones) are taken in the vicinity to further improve the symmetry computation, thereby the yellow primary plane is also sampled. Fig. 6(b) shows the symmetry values for different sampling planes in Fig. 6(c), where more samples are computed near the areas with the peak symmetries.



(a) adaptive sampling (b) uniform sampling (c) uniform sampling

Figure 6: *Adaptive Planar Sampling: Lobster dataset*

The key point for adaptive planar sampling is to know in advance where in the data possibly shows higher symmetries. Since the planar sampling will depend on the predicted symmetries, a quick and relatively precise method is required to examine the data before the complete computation as in Algorithm 1 runs. To achieve this goal, we use a Monte Carlo method to quickly obtain these symmetries, which in turn will be used as gauges to adjust planar sampling rates accordingly. Our use of Monte Carlo method is different from [PSG*06] in that we only apply it to guess the symmetries with respect to every plane on the other hand Podolak *et al.* [PSG*06] solely used Monte Carlo integration for computing the discrete symmetries.

6.4. Monte Carlo Method for Approximate Symmetries

Previously stated, the volume data are allocated to processors in a round-robin manner along a space-filling curve. Each node only holds a potion of the complete data. In [PSG*06] authors had proved that reflective symmetry can be computed by integrating the product of $f(x)$ and its reflective counterpart $\gamma(f(x))$. In this paper we apply Monte Carlo integration to approximate the local symmetries in each processor and sum up the results across processors to obtain the final results. The pseudocode for the parallel Monte Carlo symmetry approximation is shown below:

The value of *Max_Iter* is user-defined, which is usually two or three millions for a standard $256 \times 256 \times 256$ volume data set. Generally, the higher the value of *Max_Iter* is chosen the more precise the symmetries are approximated. However,higher value of *Max_Iter* will lead to higher computational costs. What we want is to approximate the symmetries but not to precisely calculate them, which is the main

Algorithm 3 Parallel Monte Carlo Symmetry Approximation

1: Each node execute the following:
2: **for** *Count*=1 To *Max_Iter* **do**
3: Randomly select 2 voxels x and x' from the allocated f
4: Use x and x' to determine a plane γ'
5: Bin the plane γ' to a proper plane γ
6: Calculate symmetry $D(f,\gamma) += \frac{f(x)f(x')}{p(x)p(x')}$
7: **end for**
8: Each node send the $D(f,\gamma)$ to root node
9: The root node sums $D(f,\gamma)$ and produces the approximation by averaging the results with *Max_Iter*.

task of Algorithm 1. The approximations can be used to adjust planar sampling lately.

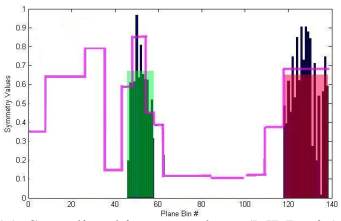
$p(x)$ and $p(x')$ are the probabilities for selecting voxel x and x'. Importance sampling is applied to accelerate sampling efficiency. We only randomly choose voxels whose values are above a user-defined threshold. $p(x)$ and $p(x')$ then can be defined easily in the preprocessing stage by counting all voxels whose values are higher than the user-defined threshold and dividing by the total number of voxels in the volume data. $p(x)$ actually is the occurrence frequency in the volume data.

2 spacial points x and x' can define 1 reflective plane γ. To compute planes' symmetries we bin the planes which have close norms and distances to the origin. It is possible that several improper planes are in the same bin, which can lead to imprecision in the final results, because of the arbitrarily selecting of points x and x'. To address this [PSG*06] added an extra Jacobian determinant weight to balance the planar bin size. We cannot simply apply the Jacobian determinant as a weight since there isn't a mesh for the volume data. Our solution is to remove the improper planes from the bins by comparing the plane's parameters. For example, a plane γ' is voted in plane bin γ in Step 5 in Algorithm 3. We calculate the L_2 difference between the normals of the voted plane γ' and the bin plane γ. If the difference is smaller than a user-defined threshold it means the plane γ' is proper, and the dot product of $f(x)$ and $f(x')$ is added to compute $D(f,\gamma)$ as in Step 6 in Algorithm 3. Otherwise points x and x' will be throw away to filter those unsuitable planes.

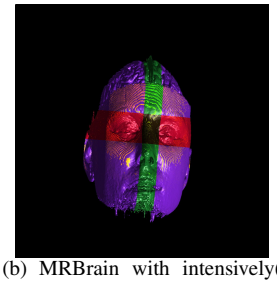
After finishing Algorithm 3, the next step is to utilize the obtained approximations to adjust the plane sampling rates by setting plane's parameters (r,θ,ϕ) accordingly. A linear mapping strategy is used to set the planar sampling steps: the highest sampling rate to the planes space area with the highest approximate symmetries.

6.5. Effects and Performance

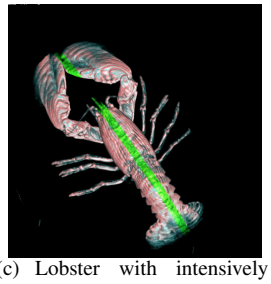
Fig. 7(a) shows the symmetries of MR_Brain dataset at a resolution of $256\times256\times109$. To reduce the space, 1055507 planes are computed in the bar chart. The heights of bars represent the corresponding symmetry values, and the number of bars represents the sampling rate at that planar space.For clarity, only a few sampling bars are drawn. The purple line shows the symmetry approximation calculated by the Monte Carlo method. From Fig. 7(a), we can see that the planar space between the 70th bin and 90th bin shows low symmetries and are given low sampling rates, correspondingly. The red and green areas are intensely sampled, since they have relatively high symmetries. Their corresponding spatial regions are drawn in Fig. 7(b). They both pass through the center of the brain, an repeatedly observed phenomena for medical data. Similarly, Fig. 7(c) shows green areas that have high symmetries for the Lobster dataset.



(a) Sampling histogram bars (MRBrain)



(b) MRBrain with intensively sampled parts



(c) Lobster with intensively sampled parts

Figure 7: *Adaptive Sampling Examples for the Brain and Lobster Dataset*

Table 1 compares the computation time between uniform planar sampling and adaptive planar sampling. Each of the three tests shows a roughly 20% time reduction with the adaptive sampling scheme.

Dataset	processors	Static	Adaptive
MRBrain	54	99	74
Lobster	28	53	40
Frog	22	35	29

Table 1: *Computation times in seconds. MRBrain is* $256\times256\times109$*, Lobster is* $301\times324\times56$ *and Frog is* $256\times256\times44$

7. 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 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 Leg in $341 \times 341 \times 93$. 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 8, the small variation of the computation times used by each of the 36 processors shows that, with a Hilbert space-filling curve implemented, our algorithm can achieve better load balancing. This implies good scalability for our parallel symmetry algorithm.

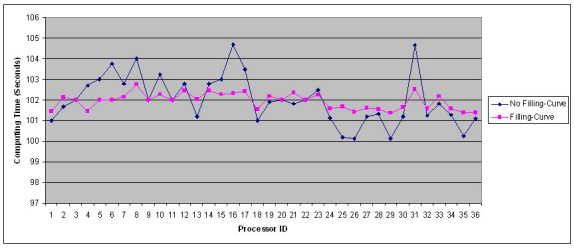


Figure 8: *The computation time in seconds, for Brain dataset, on each of* 36 *processors. The blue line indicates the time without space filling-curve data distribution, and the red line indicates time with space filling-curve data distribution.*

Figure 9 gives the speedup factors obtained using different number of processors. The multiresolution algorithm and brute-force algorithm are computed against their sequential counterparts. It is shown that our multi-resolution algorithm achieves better speedup performance than the brute-force algorithm. Approximately 87% and 81% parallel utilization were observed for 16 and 32 processors, respectively.

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 10 shows the symmetries calculated by Algorithm 1 with different error tolerances. With larger error tolerance, which is data at higher resolution, the resulting symmetry is closer to the symmetry calculated by the brute-force algorithm.

Table 2 shows different average computation times under

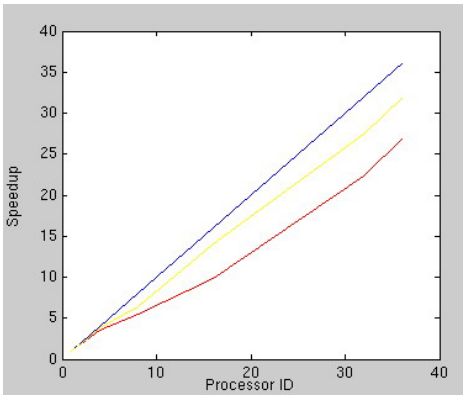


Figure 9: *Speedup factors, for Brain dataset, of our algorithms when using* 1, 2, 4, 8, 16, 32 *and* 36 *processors. The blue line is the ideal speedup, the red line is the speedup using the brute force method, and the yellow line is the speedup using the multiresolution method with error tolerance of* 1500.

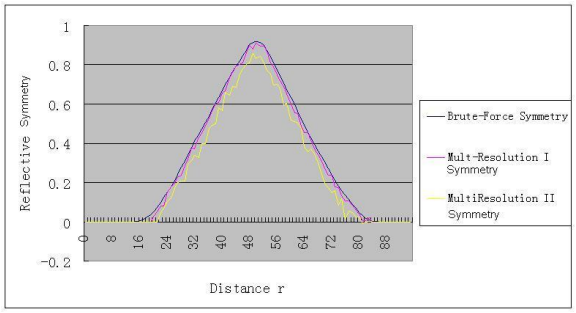


Figure 10: *The calculated reflective symmetries, for Brain dataset, for one direction [*$\theta = 0.1745\phi = 0.1745$*]. The blue line is using brute-force algorithm and the red line and yellow line are using multiresolution algorithm. Red line has a higher resolution than yellow.*

three different error tolerances by utilizing multiresolution optimization. Figure 11 and Figure 12 show the calculated

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

Table 2: *Computation times for the Vis woman in seconds, with multiresolution optimization under three different error tolerances.* 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.*

primary symmetric planes for the tests described in Table 2. The primary symmetric planes are very similar to each other. Our multiresolution symmetry computation algorithm can attain a good result in a small amount of time, if the user-defined error tolerance is properly selected.

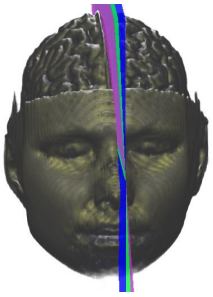


Figure 11: *The calculated primary symmetries for 128×128×72 brain data. The blue symmetric plane represents for the highest resolution Test*1 *in Table 2; the green pla for lower resolution Test*2 *in Table 2 and the purple plane for the lowest resolution Test*3 *in Table 2.*

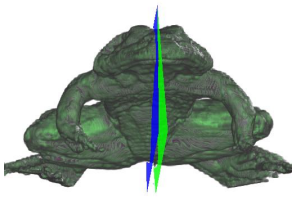


Figure 12: *The calculated primary symmetries for 256×256×44 frog. Two symmetric planes (blue and green) are derived from represent two different resolutions.*

8. Applications

Previous works have introduced various applications using symmetry. In this section, we present additional novel applications that are useful in volume visualization.

8.1. Cross-Section and Dual Rendering

A cross section is a common method for displaying the internal structure of a 3-dimensional volume object in two dimensions, which is especially useful in the area of medical imaging. By slicing a volume data along the plane with the highest calculated symmetries, we reveal more inner information, since most primary symmetric planes go through the object's center. Furthermore, if the object is rendered along the reflective plane with two different rendering techniques on each side such as tissue vs bone structure, more information can be expressed.

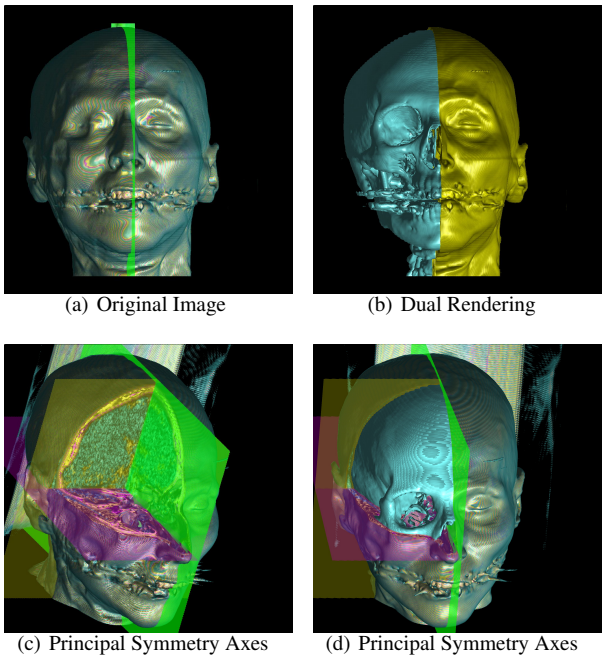
Fig. 13 is an example of exploiting symmetries for a CT head dataset. Fig. 13(a) is the original rendered image with the green primary plane is displayed Fig. 13(b) is rendered with the dual rendered effect of Fig. 13(a) with respect to the primary reflective plane. Half of the head is rendered with iso-surfacing, and the other half with a raycasting technique to draw two kinds of boundaries in skin and skull. We use three orthogonal primary reflective planes to find the center of objects and the corresponding main axis. In Fig. 13(c) and Fig. 13(d), we show the three green, yellow, and purple primary axes as planes perpendicular to each other. In Fig. 13(c), one quarter of data are removed to explore the inner components, while in Fig. 13(d), the skull is drawn instead. Fig. 13(e) is the cross section of the CT head by halving the data along the primary reflective plane.

8.2. View Selection

For some medical volume data, the reflective symmetries can be applied to find reasonable best view directions. In this paper, we apply a similar idea presented in [PSG*06]. Symmetry represents redundancy and should be avoided in the view directions. Therefore, it is possible to recommend a good viewpoint for 3D volume data by eliminating symmetrical redundancy. The view selecting procedure extracts local maxima planes from the calculated symmetries. Then, for each view direction vector, we dot-product the vector with the local maxima planes' normals. The score, for a view direction v, is $S(v) = \sum_{u \in W} |v \cdot u| Symmetry(u)$, where W is the set of local maxima planes. So, the best viewpoint which occurs at the highest score, means minimal symmetric information presented. Fig. 14 shows several viewpoint selecting examples by applying this idea. The difference between our method and the method in [PSG*06] is that we use the first and the second derivatives to obtain the local maxima planes instead of the iterative method described there since, we work with volume data. Several datasets are tested and shown in Fig. 14, These selected view directions, which intend to avoid the redundant symmetric information, are solely determined by calculated symmetries. Symmetry-based view selection can be a feasible, but it is not the only way to choose optimal view direction. It is possible to combine symmetry with other view selection methods to attain better effects.

9. Conclusion and Future Work

We have presented a parallel reflective symmetry computation algorithm utilizing several different optimizations. We show that the algorithm is efficient and stable for several different volume datasets. Our experiments also show that, the primary reflective symmetric planes pass through the centers of the objects, a reasonable phenomenon that coincides

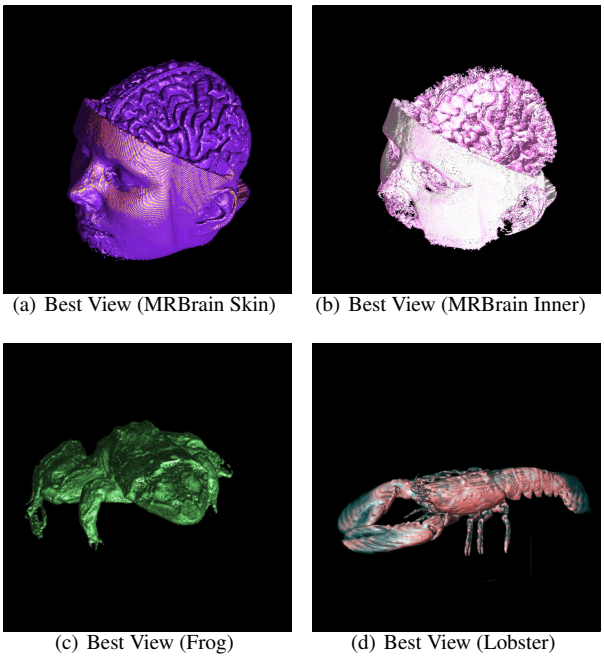


(a) Original Image (b) Dual Rendering

(c) Principal Symmetry Axes (d) Principal Symmetry Axes

(e) Section Crossing

Figure 13: *Cross sections and dual rendering effects for the CT Head Dataset (*$256 \times 256 \times 113$*)*



(a) Best View (MRBrain Skin) (b) Best View (MRBrain Inner)

(c) Best View (Frog) (d) Best View (Lobster)

Figure 14: *View Selection Examples*

with common sense. In the future work, we hope to extend our algorithm to include other types of symmetry. For example, the rotational symmetry is one possible area needing 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.

We understand that not all volume data have reflective symmetric features. Those data from scientific computation such as energy distribution and simulation are too complex to be studied solely by symmetry property. Our work is a complement to the current volume data visualization methods and hopefully to broaden its perspective in the future.

10. Acknowledgements

This work 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 borrowed from Information and Computing Sciences Division, Lawrence Berkeley Laboratory. The Teapot was from Terarecon Inc. The Leg dataset was provided by the German Federal Institute for Material Research and Testing (BAM), Berlin, Germany. The Lobster dataset was acquired from the VolVis distribution from 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.

[GS01] GAO J., SHEN H. W.: Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *Proc IEEE symposium in Parallel and Large Data Visualization and Graphics '01* (2001), pp. 67–74.

[GHS*03] GAO J., HUANG J., SHEN H. W. KOHL J. A.: Visibility culling using plenoptic opacity function for large data visualization. In *Proc IEEE Visualization '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.