Visibility Culling Using Plenoptic Opacity Functions for Large Volume Visualization

Jinzhu Gao*¶ The Ohio State Univ.
Jian Huang† The Univ. of Tennessee
Han-Wei Shen‡ The Ohio State Univ.
James Arthur Kohl§¶ Oak Ridge National Lab

Abstract

Visibility culling has the potential to accelerate large data visualization in significant ways. Unfortunately, existing algorithms do not scale well when parallelized, and require full re-computation whenever the opacity transfer function is modified. To address these issues, we have designed a Plenoptic Opacity Function (POF) scheme to encode the view-dependent opacity of a volume block. POFs are computed off-line during a pre-processing stage, only once for each block. We show that using POFs is (i) an efficient, conservative and effective way to encode the opacity variations of a volume block for a range of views, (ii) flexible for re-use by a family of opacity transfer functions without the need for additional off-line processing, and (iii) highly scalable for use in massively parallel implementations. Our results confirm the efficacy of POFs for visibility culling in large-scale parallel volume rendering; we can interactively render the Visible Woman dataset using software ray-casting on 32 processors, with interactive modification of the opacity transfer function on-the-fly.

CR Categories: I.3.1 [Computer Graphics]: Parallel processing—; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—visible line/surface algorithms

Keywords: visibility culling, volume rendering, plenoptic opacity function, large data visualization

1 Introduction

A growing number of scientific and medical applications are now producing very large datasets, ranging from gigabytes to terabytes, on a daily basis. Such immense data require sophisticated techniques for analysis and presentation; scientific visualization is an indispensable tool for analyzing and understanding those datasets. However, as dataset sizes increase, the usability of traditional visualization approaches is severely challenged, with high requirements on the necessary storage, computation speed and network bandwidth. Faster rendering hardware and optimized algorithms are unlikely to prove sufficient to address the full breadth of challenges faced by visualization researchers in the near future. More intelligent and novel methods must be designed and developed to minimize the data movement and processing overheads, thereby increasing the level of interactivity in viewing very large datasets. In this paper, we specifically focus on the subject of volume rendering in the context of large volume visualization.

Among several possible alternatives, visibility culling is a popular technique that eliminates unnecessary visualization computation. Yet while visibility culling can accelerate the rendering speed, larger datasets also require more tractable data management, as well as massive-scale parallelism. It is crucial to have scalable parallel solutions for determining the visibility of coarse-grained groups of voxels, or *volume blocks*. Previous visibility assisted parallel visualization algorithms, such as [Huang et al. 2000b; Gao and Shen 2001], are difficult to scale efficiently in massively parallel environment because of the significant inter-processor dependencies needed to maintain a globally coherent opacity buffer. The main challenge is due to the difficulty of knowing the exact set of visible blocks before the rendering is finished. In addition, parallel volume rendering algorithms with visibility culling are also subject to the specific underlying opacity transfer function. Each time the opacity transfer function changes, the whole process of visibility determination must be repeated. This greatly hinders interactive exploration for large-scale datasets.

In this paper we advocate a coarse-grained approach to visibility culling, based on a novel way to encode pre-computed view-dependent opacity of a volume block. Our new representation of view-dependent opacity is based on *Plenoptic Opacity Functions* (POFs). Our key research motivation is that the opacity variation of a volume block under all external viewpoints can be treated as a function defined on spherical coordinates. This opacity function possesses a high coherence and, hence, can be represented in compact mathematical terms using conservative approximations. Therefore, a POF representation of a volume block is space efficient and can be reused for all possible views outside the block, i.e. it is plenoptic. Further, a block's opacity information must not always be re-computed for each new opacity transfer function. We have discovered, through rigorous mathematics deduction, that an existing POF representation of a volume block can be accurately reused for all opacity transfer functions sharing the same set of basis functions, without the need to re-compute. This set of functions is referred to as *a family of opacity transfer functions*. POFs are conservative and effective in visibility culling, are fast to compute off-line, and can be evaluated during run-time for opacity determination under an arbitrary view angle. Using POFs, scalable visibility culling can be done on a coarse-grained level, for both online parallel processing and out-of-core computation. We have experimentally tested the efficacy of POFs for visibility accelerated parallel volume rendering, and have interactively rendered the Visible Woman dataset in software using 32 processors. The POF solution allows modification of the opacity transfer function on-the-fly without loss of interactivity.

While early ray termination is the widely accepted foundational idea behind all visibility culling methods in volume rendering, the

*e-mail: gao@cis.ohio-state.edu
†e-mail: huangj@cs.utk.edu
‡e-mail: hwshen@cis.ohio-state.edu
§e-mail: kohlja@ornl.gov

¶Research supported in part by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

IEEE Visualization 2003,
October 19-24, 2003, Seattle, Washington, USA
0-7803-8120-3/03/$17.00 ©2003 IEEE

compelling diversity of visibility acceleration algorithms proposed over the years [Livnat and Hansen 1998; Parker et al. 1998; Huang et al. 2000b; Gao and Shen 2001] reflects the very nature that a universal way to implement this concept does not exist. A specific scenario may require a solution uniquely optimized to its very setting. Indeed, it is fair to regard our framework as a scalable instantiation of the basic concept of early ray termination to be used in parallel settings to render large volumes with a family of transfer functions. Using POFs, we avoid the otherwise significant system overhead from excessive interdependence among large numbers of processors for early ray termination and drastically different access bandwidth of volume data due to unpredictable nature of occlusion. Further, our approach uses a low overhead one-time precomputing step to support a family of transfer functions of an infinite number.

In the remainder of this paper, we briefly discuss the background of large volume visualization and visibility culling (Section 2). The details of POF, our parallel volume rendering algorithm and experimental results are presented in Sections 3 through 5, respectively. In Section 6, we summarize our contribution and conclude with a discussion of possible future work.

2 Related Work

The definition of "large" with respect to dataset size continues to evolve by orders of magnitude, from Gigabytes and Terabytes now to Petabytes in the near future. Scientific visualization is an indispensable tool to glean scientific insights from such massive datasets, as can be seen by the wealth of publications with pioneering research results. Approaches for more efficient rendering have been proposed, such as large-scale parallelism [Ma et al. 1994; Ma and Crockett 1997], hardware acceleration [Lum et al. 2002], visibility culling [Livnat and Hansen 1998; Parker et al. 1998; Huang et al. 2000a; Gao and Shen 2001; Klosowski and Silva 2001] and compression mechanism [Bajaj et al. 2000; Guthe et al. 2002]. For large volume visualization, it is imperative to utilize as many processing units as possible, but in an efficient and scalable manner. Inter-process communication and synchronization overheads must be reduced to a minimum. As even special purpose hardware has a limited number of resources, intelligent algorithms must be designed to manage memory efficiently, especially when paging a large dataset in and out [Zhang et al. 2001].

2.1 Visibility Culling

In order to reduce the cost of unnecessary computation, a number of visibility culling methods have been proposed, especially for polygon rendering applications [Greene 1996; Zhang et al. 1997; Wonka et al. 2001]. A multidisciplinary survey on 3D visibility was given by Durand in [Durand 1999]. In volume visualization, visibility culling aims to eliminate occluded portions of the data volume, as early as possible in the visualization pipeline, to reduce the rendering cost. Discovering and encoding visibility information has often been rather expensive, and is strictly dependent on the given opacity transfer function. Also, most culling methods do not easily scale in parallel settings [Huang et al. 2000b; Gao and Shen 2001].

The early ray termination method used for ray casting [Levoy 1988; Levoy 1990] was one of the first approaches in volume rendering to utilize an occlusion heuristic for acceleration. A ray-front scheme was introduced by Law and Yagel [1996] to take advantage of both the image-order and the object-order traversal and avoid thrashing. Image-aligned sheet-based splatting [Huang et al. 2000a] uses a summed area table of the opacity buffer to cull away a single voxel or a group of voxels projected to a screen region whose average opacity is already 1.0. However, due to the object-space based nature of splatting, this incurs more overhead for visibility culling than for ray-casting approaches. To accelerate isosurface rendering, Livnat and Hansen [1998] proposed a view-dependent isosurface extraction algorithm that utilizes a visibility culling technique similar to the one proposed by Greene [1996]. This algorithm reduces both extraction and rendering time by culling away the occluded isosurface patches. However, visibility determination may still incur extra overhead, which can be a potential performance bottleneck. Parker *et al.*[1998] introduces an interactive ray tracing algorithm to generate the visible portion of an isosurface. The algorithm computes the intersection point of a ray and the isosurface directly by solving a cubic equation, without explicitly extracting triangles. In general, this algorithm is well-suited for parallel implementation. Liu *et al.*[2001] defines an algorithm to extract the visible isosurface progressively using both ray casting and propagation.

Occlusion acceleration is very effective for datasets which contain a large number of occluded regions. However, because visibility culling is inherently sequential by nature, a scalable parallel implementation is difficult to construct. It is also unlikely to achieve a balanced workload after dynamically removing occluded data, therefore few parallel rendering algorithms utilize visibility culling. An effective parallel rendering algorithm with visibility culling was developed by Huang *et al.*[2000b] for volume datasets with heavy to moderate occlusion. The algorithm utilizes view-dependent object-space data partitioning with an image-space task partition. An occlusion map is also kept for culling occluded data. Unfortunately, the algorithm incurs a global barrier for synchronization of the opacity buffer for all processors on slab boundaries, therefore it does not scale well for more than 16 processors. Parallel implementations of view-dependent isosurface extraction are challenging because the visibility of most blocks is unknown before extracting the isosurface. To utilize visibility culling in parallel isosurface extraction, Gao and Shen [2001] proposed a progressive visibility culling method that can efficiently eliminate invisible isosurface triangles, achieving satisfactory parallel speedups.

To summarize, visibility culling for large volume visualization must be very efficient with scalable parallel implementations, and must be conservative while still remaining effective. Interactive opacity computation while rendering is not feasible for large volume visualization. These facts have motivated our search for an opacity representation of coarse-grained voxel blocks that enables efficient visibility culling.

3 Plenoptic Opacity Function

Although occlusion acceleration for volume rendering seems to be only effective for datasets showing a dominating amount of opaqueness, large datasets with highly transparent opacity transfer functions can still benefit from visibility culling. Consider the following example.

In [Max 1995], the opacity value along a viewing ray is computed as follows:

$$\alpha = 1 - e^{-\int_0^d \tau(t)\,\mathrm{d}t} = 1 - e^{-\sum_{i=1}^{l} \int_{s_i}^{s_{i+1}} \tau(t)\,\mathrm{d}t} = 1 - \prod_{i=1}^{l} (1 - \alpha(s_i)) \quad (1)$$

where d is the total length of the viewing ray, which is broken into l segments to compute the Riemann sum, and $\alpha(s_i)$ represents the accumulated opacity value of each individual segment along the viewing ray. In practice, using a step size of 1.0, we usually approximate $\alpha(s_i)$ with the opacities of sample points on the viewing ray. For practical visibility culling, it is common to use a threshold slightly lower than but close to 1.0, such as 0.95. In this case, as soon as a pixel has reached 0.95 in opacity, the pixel is considered fully opaque. With a highly transparent opacity transfer function, where the opacity of every sample point is as low as 0.05, we can

determine the shortest length l for this viewing ray to reach full opacity:

$$0.95 = 1 - \prod_{i=1}^{l}(1 - 0.05)$$

Solving for l we find that $l = 58.4$. That is, as long as a viewing ray is more than 59 sample points in length, the corresponding pixel will have reached full opacity. For a large dataset whose size is on the order of $1000 \times 1000 \times 1000$, being able to cull all voxels after the first 59 non-empty sample points of only 0.05 opacity allows significant acceleration. Therefore, visibility culling is useful for a much wider spectrum of large scale dataset scenarios than expected. The only exceptions are those relatively sparse datasets with a small range of depth, which do not exhibit as significant of a performance problem in the first place.

In the following, we describe a method to pre-compute and encode the opacity information for each volume block using a novel Plenoptic Opacity Functions (POF) scheme. By "plenoptic," we mean that a single POF can capture the view dependent opacity of a volume block for all possible external views. At run time, using the POFs for all blocks, scalable visibility culling is supported without the need for global barriers [Huang et al. 2000b] or multiple passes [Gao and Shen 2001].

3.1 The Design of POF

Because opacity computation at run time is expensive and can affect the scalability of parallel applications, we aim to pre-compute a large portion of such information during pre-processing, such that run-time visibility determination is efficient and scalable. Our rationale is that the opacity variation of a volume block from all external viewpoints can be considered as a function defined on spherical coordinates. With any block, the rendered opacities obtained from adjacent views should show a high degree of coherence, although the exact intrinsics of such coherence may be complicated. But, in coarse-grained visibility culling, occlusion of a block can depend solely on the pixel with the lowest opacity within its screen projection area. Leveraging this observation, we opt to focus on the minimal opacity value of a volume block among all pixels within its screen footprint from every view and such minimal opacity value will be considered conservatively as *the opacity of the block* in this paper. In this way, the opacity of a block from every view can be greatly simplified and encoded as a scalar function of two variables, θ and ϕ, in spherical coordinates. This is in fact the form of POF we use in our work. However, it is impractical to pre-compute and store the opacities from all possible views. [Zhang and Turk 2002] points out that the visibility information from a limited number of sample views can be used to interpolate and estimate the visibility information from any practical sample view. Therefore, by taking advantage of view coherence, we need to only pre-compute the opacity information for a manageable set of evenly spaced sample viewpoints.

As shown in Figure 1, when constructing a POF, we store the minimal opacity values of a volume block, for all sample views, into a 2D table. Here, a volume block, B, is rendered from view $(0,0)$ into an alpha (opacity) image, S. In the opacity image, we search for the pixel with the lowest opacity, represented by the shaded pixel. We refer to this pixel as q_{min} in this paper. Obviously the location of q_{min} varies from view to view. The opacity value of q_{min} is then stored at location $(0,0)$ in a two dimensional table indexed by θ and ϕ. This process is repeated for all locations in the discrete table representing the POF, corresponding to all sample views. Each view is parameterized as $view_{i,j} = (\theta_i, \phi_j)$ and the 2D table indeed represents a function $minopacity(\theta, \phi)$ indexed by θ and ϕ. Occasionally, due to insufficient sample rates along the dimensions of θ and ϕ, sharp spikes may occur. In this case, one can always use a

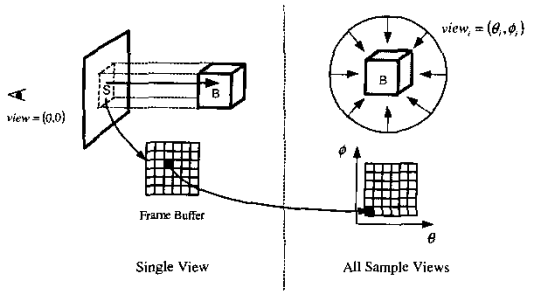


Figure 1: For each view (left), the opacity channel of a volume block B is rendered into a frame buffer. Suppose the pixel shaded with blue has the minimal opacity among all non-empty pixels. This minimal opacity value is stored into the entry shaded with green in a 2D table indexed by θ and ϕ (right). The same process is done for all sample views around block B.

classical noise reduction filter, such as a median filter, to conservatively smooth out such singular data points. In addition, to ensure the correctness of the minimal opacity value, the resolution of the alpha image should be large enough so that a voxel's projection area is larger than a pixel in the alpha image.

Although there may be some undulations of minor amplitudes in a POF function, the overall shape of variation should demonstrate a smooth envelope signifying the underlying view coherence. Due to such coherence, using more sophisticated mathematical representations could achieve higher storage efficiency, although keeping a POF in discrete forms is also affordable. Potential candidates for such representations include polynomials, splines, Fourier transforms and Wavelets. However, it is critical for the compact mathematical representation chosen to have a high run-time efficiency for evaluation from any given viewpoint. As a note, no matter which compression method is used, the opacity value from each view angle in the compressed POF must not be larger than the value in the corresponding entry in the original discrete POF table. In our algorithm, we calculate a polynomial first and then shift it down until it satisfies the criteria. We have experimented with both discrete table and 3rd order polynomials as two options to store a POF, due to their fast run-time evaluation. To represent a POF as a third order polynomial, classical function fitting techniques [Press et al. 1992] are used. After this pre-processing stage, each volume block has a POF computed and stored for run-time visibility determination.

3.2 Run-Time Opacity Determination Using POFs

The POF we compute for each volume block is dependent on the underlying opacity transfer function. When the opacity transfer function is changed, the opacity of volume blocks can become totally different, and therefore the POF must be re-computed, which is time-consuming and limits the flexibility and interactivity of the algorithm. In this section, we describe how the POFs can instead be directly reused at run time when the opacity transfer function changes. The basic idea is to construct the initial opacity transfer function from a set of basis functions. We compute a set of POFs for each of the basis opacity transfer functions at the pre-processing stage. Then, at run time, a new opacity transfer function can be constructed simply by weighted summation of the basis set of opacity transfer functions. We show below that the POFs can be directly reused to determine the opacity of a volume block with high efficiency.

3.2.1 Family of Opacity Transfer Functions

Our focus in this paper is on acceleration of volume rendering using opacity culling. We discuss in this section how to extend our algorithm to handle a large number of opacity transfer function with merely a one-time precomputation. One should note that the subject of obtaining a meaningful transfer function, given an arbitrary dataset is, however, beyond our scope. The base transfer functions used in our work are obtained by partitioning transfer functions, designed with approaches such as those in [Kindlmann and Durkin 1998; Pekar et al. 2001; Levoy 1988], into piecewise continuous segments.

In [Kindlmann and Durkin 1998], higher opacities are assigned to areas in the dataset that are most likely to have boundaries present; this calculation is based on raw data values, as well as the first and second derivatives of the raw data. [Pekar et al. 2001] also provides an efficient method to detect the intensity transitions in volume data automatically. Transfer functions can be extended to multi-dimensional spaces as well [Levoy 1988], where opacity values can be determined using both raw data values and variations in the strength of gradients. Subsequently, this has been shown to be effective in data exploration when the opacity values of data can be interactively scaled and translated under user control [Kniss et al. 2001].

After applying a transfer function design approach, such as [Kindlmann and Durkin 1998], to the raw data, certain value intervals representing boundaries are assigned non-zero opacity values. There will be one basis opacity transfer function for each of such value intervals. The basis functions can overlap each other in the space of raw data values, however. Then scaling the opacity of separate basis functions individually by different factors can provide a family of opacity transfer functions, each emphasizing the various boundary areas to a different degree.

In $\alpha = 1 - e^{-\int_0^d \tau(t)\,dt}$, $\tau(t)$ is the extinction coefficient defined along the viewing ray. The value of τ is commonly referred to as the opacity of a sample point in the volume. Its value is view independent and directly related to the opacity transfer function used. The resulting α, however, represents the opacity accumulated along the viewing ray, composed of a number of volume samples. The value of α is also view dependent. Because of the direct relation between the opacity of a volume sample and the opacity transfer function, in this paper, we use the symbol of τ to represent opacity transfer functions as well. Assuming an initial opacity transfer function is $\tau(v)$, consisting of p basis functions, where the value of v is between the minimum and maximum value of the raw data, we can represent the opacity transfer function as: $\tau(v) = \sum_{i=1}^{p} \tau_i(v)$. The family of opacity transfer functions using the basis functions is defined as

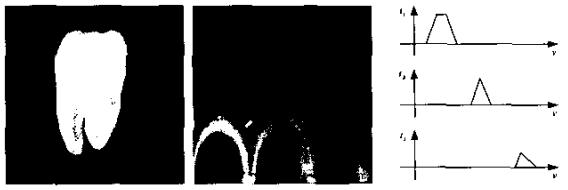
$$\tau'(v) = \sum_{i=1}^{p} k_i \tau_i(v) \tag{2}$$

Each opacity transfer function in the family is controlled by a scale, k_i, ($k_i \geq 0$, $i \in [1:p]$), and $\tau_i(v)(i \in [1:p])$ are the basis functions of the family.

For illustration purposes, in Figure 2, from the relationship between f' and f, a possible set of basis functions can be generated. Each basis function emphasizes one area and can be scaled using a different factor, k_i ($i \in [1:4]$), to get different opacity transfer functions belonging to a same family.

3.2.2 Opacity Determination for a Single Ray

In this subsection, we describe how the construction of the opacity transfer function, as well as the scaling of the individual basis functions, can be accounted for in determining the opacity information



(a) (b) (c)

Figure 2: An example illustrating the generation of basis opacity transfer functions: (a) tooth dataset; (b) f' versus f; (c) a possible selection of a set of basis functions.

at run time. We show how the pre-computed accumulated opacity for a single ray can be adjusted for each of the bases, $\tau_i(v)$, as scaled by k_i, directly from the same set of POFs. In the next section, we will generalize this principle to update the opacity of a block.

According to [Max 1995], opacity along a viewing ray is computed as $\alpha = 1 - e^{-\int_0^d \tau(t)\,dt}$. After scaling all the opacity values along the viewing ray by a factor k, we can get a new opacity, $\alpha' = 1 - e^{-k\int_0^d \tau(t)\,dt}$. It is easy to derive $\alpha' = 1 - (1-\alpha)^k$, which describes how the final opacity would change when a single basis is scaled by the factor of k. When an opacity transfer function $\tau(v)$ consists of multiple basis functions, written as $\tau_i(v)$, only considering one basis function at a time, we obtain a separate opacity value, α_i, along the same viewing ray for each basis function:

$$\alpha_i = 1 - e^{-\int_0^d \tau_i(v(t))\,dt}$$

Fortunately, from these separate opacities α_i, the total opacity can be computed as:

$$\alpha = 1 - e^{-\int_0^d \sum_{i=1}^{p} \tau_i(v(t))\,dt} = 1 - \prod_{i=1}^{p} e^{-\int_0^d \tau_i(v(t))\,dt} = 1 - \prod_{i=1}^{p} (1-\alpha_i)$$

Assuming there are p basis functions in total, when each basis function is scaled by a different factor, k_i, we can obtain the accumulated opacity of the ray as:

$$\alpha' = 1 - \prod_{i=1}^{p} (1-\alpha_i') = 1 - \prod_{i=1}^{p} (1-\alpha_i)^{k_i} \tag{3}$$

There are no approximations involved in the above derivation. Therefore, for any opacity transfer function in the same family, the final formula for computing the opacity for a single ray is exact.

3.2.3 Opacity Determination for a Volume Block

At run time, to determine the opacity of a block is very straightforward if a single basis, $\tau_0(v)$, is used in the family of opacity transfer functions. The process includes looking up the minimal opacity of the current view from the stored POF computed for this basis, and then inserting it into Equation (3) with a scaling factor, k, controlled by the user. The resulting opacity is the opacity of the block in this view using the opacity transfer function $k\tau_0(v)$. This process is exact, and does not involve any approximation.

When multiple bases are used, it is more complicated to determine the opacity of a block because the pixel showing minimal opacity under one single basis may not be the pixel showing the minimal opacity when all bases are considered. A trivial solution is to compute a POF for each pixel in the block's screen footprint for each basis. At run time, we compute the true opacity of each pixel according to Equation (3) and search for the minimal value. But this process is too expensive in storage and inefficient in computation.

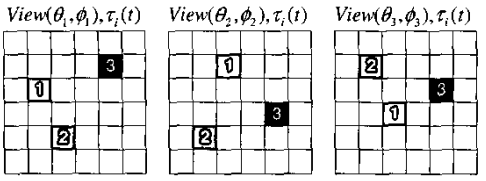


Figure 3: An example shows that, for each basis opacity transfer function $\tau_i(v)$, the location of q_{min} can be different under different views. Here we assume there are 3 basis opacity transfer functions. A cell with a number i inside represents the pixel with minimal opacity value for the basis opacity functions $\tau_i(v)$.

Instead, using POFs, we can compute the opacity of a block from a given view by looking up the opacity from each POF and then inserting them into Equation (3), with the chosen k_i's. The resulting opacity can then be considered as the opacity of the block. However, this method may be overly conservative for some datasets as well as some opacity transfer functions even though it achieved satisfactory culling performance in our experiments.

To improve the above approach for handing multiple bases, we notice that the collection of q_{min}'s of all bases actually provides a small but highly probable set of locations which the pixel with minimal opacity may fall in from a given view. Obviously, from a given view, the location of q_{min} for each basis function may be different. We refer to this fact by saying each basis has a different q_{min}. In addition, the location of a q_{min} is view dependent, illustrated by Figure 3. For p bases, we have only p q_{min}'s from a given view. Thus, to support opacity determination of a block when multiple bases are used, we developed the following more aggressive approach: for the basis, $\tau_i(v)$, we compute a POF from all sample views as in previous sections; in addition, $p-1$ POFs are computed for each of other $p-1$ bases at the location of $\tau_i(v)$'s q_{min} from each sample view. The purpose of keeping these extra $p-1$ POFs is to compute correct opacity at the location of $\tau_i(v)$'s q_{min}, using Equation (3), under any view. Collectively, for each basis, the p POFs are referred to as a POF set. Obviously, there are p POF sets for p basis functions. When the user changes the opacity transfer function by manipulating individual weights, k_i, we can compute the opacity on each of the p q_{min}'s by using the modified values of k_i in Equation (3). The minimal value among these p resulting opacities is very likely to be the new opacity of a block. In this way, we can have a better estimation as to which pixel may show the minimal opacity after all bases are considered, without having to check every pixel in the footprint.

As a summary, when a family of opacity transfer functions with a single basis is used, determining the opacity of the block under a given view is straightforward, efficient and conservative. When multiple bases are necessary, one can directly use the multiple POFs to get the opacity of a block conservatively, or use POF sets to achieve a more aggressive opacity estimation. According to our experiments, since visibility culling takes a negligible amount of time, the two methods we have on multiple bases performed rather similarly.

3.3 Visibility Determination using POF

Leveraging the mathematics equations developed in the previous section, the POFs of each block are used for conservative run-time visibility determination. For a given view (θ, ϕ), all volume blocks are visited in a front-to-back order and an opacity buffer will be used to store an accumulated opacity value for each pixel. To test the visibility of each volume block, the screen footprint of the block is computed, based on which the opacity buffer is queried. If all values inside the screen footprint are beyond a pre-defined threshold of opaqueness, say 0.95, then the volume block is identified as invisible. Otherwise, the volume block is visible and the minimal opacity value retrieved from the block's POF (θ, ϕ) will be composited into the opacity values inside the block's screen footprint. As long as the front-to-back order is maintained, such visibility culling is straightforward and carries a low overhead. Parallel execution of the visibility tests further reduces the absolute time needed in visibility determination.

4 Parallel Volume Rendering with POFs

To allow for interactive visualization of very large scale datasets, computation can be done in parallel to accelerate the rendering process. Visibility culling offers great potential to further speed up parallel visualization algorithms. A parallel volume rendering algorithm can be an ideal test case to demonstrate the efficacy of POFs, using which, occlusion can be determined with negligible overhead before rendering begins. It is possible to predict occlusion in parallel and still allow for proper load balancing. Combined with a suitable data distribution method, it is possible to achieve highly scalable parallel rendering performance. Our algorithm is described in more detail in the following subsections.

4.1 Data Distribution along Space-Filling Curves

To achieve a balanced workload distribution during run-time parallel volume rendering, an entire volume is first partitioned into volume blocks. The distribution process is performed by traversing the volume blocks along a space filling curve [Pascucci et al. 2003], such as a Hilbert curve, during which blocks are assigned to processors in a round robin fashion.

The basic motivation for this static data distribution is that in most datasets the data values are continuous. The visible parts of these datasets are then likely to be contiguous with a high spatial locality. Space filling curves dictate that a curve traverses its local neighborhood completely before stepping outside of the local neighborhood. Therefore, when data are distributed following the space-filling curve, in a consecutive round-robin manner, visible volume blocks will tend to be distributed evenly among the processors. This will be regardless of the underlying opacity transfer function, provided that the block partition is sufficiently fine in scale. Each processor only renders the blocks statically assigned to it; such static data distribution is especially desirable when rendering large-scale datasets, as redistributing even a small portion of a dataset could incur very high communication overhead.

4.2 Parallel Volume Rendering with POF-Assisted Visibility Culling

Given pre-computed POF and a static data distribution, visibility culling can be applied to parallel volume rendering without the need of global barriers. Our rendering algorithm consists of the two major portions, as described in the following subsections: parallel visibility culling and parallel volume rendering.

4.2.1 Parallel Visibility Culling

Because the storage overhead for POFs is quite small when encoding them as polynomials, each processor keeps a copy of the precomputed POF information for every block in the volume, including the blocks that are assigned to other processors. This allows visibility culling to be done in parallel without additional communication overheads. The relevant details are itemized below:

- **Image Space Partition:** The screen space bounding box of the whole volume's screen projection area is partitioned into smaller tiles with equal size where the number of the tiles equals the number of processors. Each processor is assigned one tile and is responsible for identifying the visible blocks whose screen footprints overlap with the tile. This processor is also responsible for compositing the final image of the assigned tile.

- **Visibility Culling:** For a given view (θ, ϕ), at each processor, all volume blocks will be visited in a front-to-back order which can be enforced by traversing the blocks along a 3D z curve, and only the blocks whose screen footprints overlap with the assigned tile are tested for visibility. An opacity buffer of size equal to the tile size will be used to store an accumulated opacity value for each pixel inside the tile. Final visibility determination is performed for each block as discussed in Section 3.3.

4.2.2 Parallel Volume Rendering

After visibility determination, all volume blocks that are not occluded are rendered. Through collective communication, each processor obtains the indices of the non-occluded blocks it should render. Specifically, each processor only render the non-occluded blocks pre-assigned to it during data distribution, to capitalize on local data accesses. Because the volume blocks are distributed along the space filling curve, the rendering workload should be quite well-balanced among the various processors. During image space partitioning, each processor is assigned a tile. Throughout the course of the parallel volume rendering, this assignment remains static and each processor is responsible for compositing the final image for its tile. When a processor finishes rendering one volume block, the resulting partial image is sent to any processors whose assigned tiles overlap with the partial image of this block. After all the blocks have been rendered, each processor composites all the partial images received to produce the final image for the tile. Finally all image tiles are collected from the processors by the host node to form the final image.

5 Results

In this section, we present experimental results for POF construction, as well as the effectiveness and the efficiency of our parallel volume rendering algorithm utilizing POF-assisted visibility culling. All tests were run using 32 1.53 GHz AMD Athlon 1800MP processors, on a parallel cluster connected by Myrinet. The $512 \times 512 \times 1728$ Visible Woman dataset from the National Library of Medicine was used as our test dataset. In our experiments, the whole volume is partitioned into 110,592 ($16 \times 16 \times 16 = 4096$ voxels) volume blocks, distributed to processors along a space filling curve in a round robin fashion. The image resolution is 512×512 by default.

The POFs can be computed in embarrassingly parallel fashion. It took approximately 4.78 minutes for us to compute the POF information for all blocks in the Visible Woman dataset, using 32 processors, from 1296 sample views. The POF for each block is stored either discretely as in a raster table or in a closed form as a 3rd order polynomial. The size of the raster POF table is proportional to the number of sample views used. For the Visible Woman dataset, with 1296 sample views (in a 36×36 grid defined in spherical coordinates), the total storage requires about 55 Mbytes, using a single basis opacity transfer function; and about 500 Mbytes, using three basis opacity transfer functions. As a form of compression, a polynomial function is used to represent each POF. In the 2D space

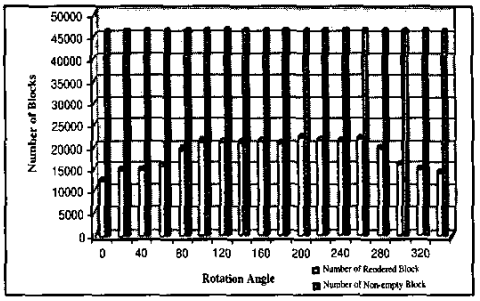


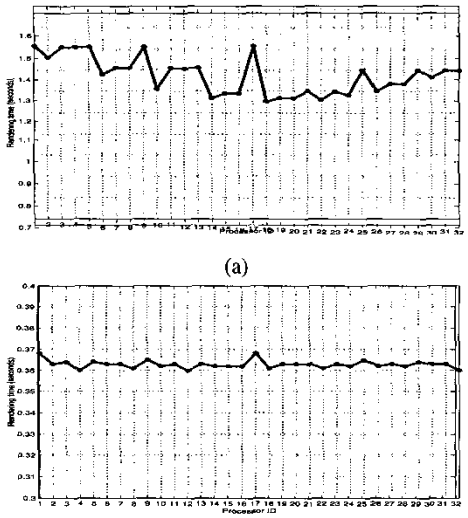
Figure 4: The visibility culling effect of our parallel volume rendering algorithm at 18 test views, while rotating the viewpoint around Y axis.

of spherical coordinates, θ and ϕ, utilizing a third order polynomial, we can encode a POF table using ten coefficients, amounting to only 40 bytes. Using the polynomial encoding, only about 5 Mbytes is needed to store the POF information for the Visible Woman dataset, using a single basis function, and about 50 Mbytes when three basis functions are used. However, compression with polynomials may cause the POF encoding to be more conservative. Using the opacity transfer function shown in Figure 7, 10,862 out of 46,037 non-empty volume blocks are determined to be visible when using the discrete POF table, while 12,356 are found to be visible with the polynomial POF representation. Since both forms of POF encoding cull about 73% of non-empty volume blocks, we believe that the third order polynomial representation is a practical option to improve storage efficiency.

With pre-computed POF information, our algorithm can cull away the invisible portion of the dataset effectively. Using POF, our algorithm doesn't need to re-compute opacity information each time the view changes. Figure 4 shows the variation of the number of rendered blocks as we rotate the viewpoint around the Y axis, to demonstrate that that POF-assisted visibility culling is quite stable for all views. Figure 5 compares the total time used by the algorithm without coarse-grained visibility culling with our algorithm for a test view. In both algorithms, early ray termination is still used when a volume block is rendered. This shows that a significant performance improvement was gained when coarse-grained visibility culling was utilized.

To verify that POF calculation is not overly conservative, we implemented a sequential visibility culling algorithm, without using POF, for a comparison. The focus of this experiment was to compare the effectiveness of the visibility culling irrespective of performance. In the sequential algorithm, the volume blocks are sequentially visited in a front-to-back order. Each time a block is visited, it is tested for visibility by comparing the screen projection area of the block and the composited rendering result of previously rendered blocks. If it is not occluded by previously rendered volume blocks, then it will be rendered. Otherwise, it will be skipped. This sequential algorithm found 10,033 volume blocks that are not occluded and culled about 78% non-empty volume blocks, which is very close to the result of our algorithm using POF table(76%).

Static data distribution along a space filling curve gives our parallel volume rendering algorithm a well-balanced workload. In our tests, both a Hilbert curve and a Z curve have been tested, producing very similar results. In Figure 5, the small variation of the rendering time used by each of the 32 processors shows that, even with visibility culling, our algorithm can still achieve good load balance. This implies good scalability for our parallel volume rendering algorithm. Figure 6 gives the speedup factors obtained using different number of processors. Our algorithm achieves approxi-



(a)

(b)

Figure 5: Two graphs showing the rendering time, including the time spent on visibility determination, ray casting and image composition, on each of 32 processors: (a) using the algorithm without visibility culling, and (b) using the algorithm with visibility culling. They are shown separately due to the large difference in the rendering time.

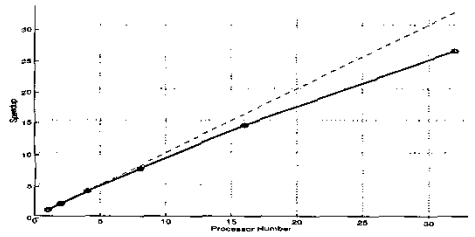


Figure 6: The speedup of our algorithm when using 1, 2, 4, 8, 16 and 32 processors.

mately 2 frames per second when using 32 processors. About 89% and 81% parallel utilization are observed for 16 and 32 processors, respectively. These results show a large improvement over previous occlusion accelerated parallel algorithms [Huang et al. 2000b; Gao and Shen 2001].

POFs are very useful for families of opacity transfer functions defined on multiple basis functions. Pre-computing POFs for each new opacity transfer function is avoided. After the POFs are computed for a set of basis opacity transfer functions, they are used to perform visibility culling for any opacity transfer function generated from the same set of basis functions. Figure 7 shows two basis functions in an initial opacity transfer function and the images produced using each basis functions. The images generated using different scaling factors are shown in Figure 8. From these results, it can be seen that visibility culling works well when the opacity transfer function is adjusted on the fly.

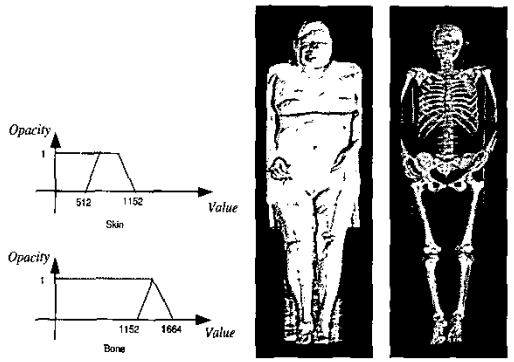


Figure 7: Basis functions representing skin area and bone area.

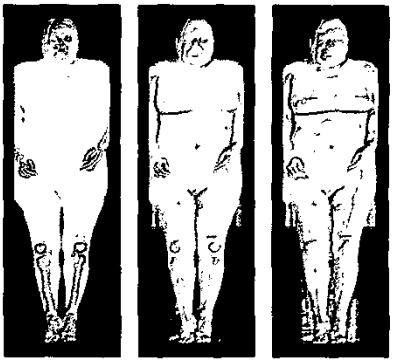


Figure 8: Rendering results after changing the scaling factors of basis functions. Left: k0=0.05, k1=1.0, 45866 visible blocks, 173 culled blocks; Middle: k0=0.25, k1=0.9, 24515 visible blocks, 21524 culled blocks; Right: k0=0.80, k1=0.5, 13458 visible blocks, 32581 culled blocks; Here k0 and k1 are factors to control the opacity transfer functions for skin area and bone area, respectively. For the best interpretations of these results, please see the color images.

6 Conclusion and Future Work

This paper introduces the concept of Plenoptic Opacity Functions (POFs) to encode the opacity information for each volume block in a volume rendering algorithm. Such opacity information can be pre-computed for each block from a set of sample views. A parallel scalable volume rendering algorithm has been designed and implemented to utilize such pre-computed POFs, using static data distribution along space filling curves, to cull away occluded volume blocks. When the view changes, although the visibility determination still needs to be done, a full re-computation of the opacity information is avoided. By analyzing the integral form of the opacity calculation for a single ray, POF pre-computation has been extended to support a family of opacity transfer functions. In this way, our algorithm can perform visibility culling for any opacity transfer function in a family without full POF re-computation.

It is true that discrete representation of a POF may not be as accurate as its continuous space counterpart. However, to be conservative and effective in occlusion culling, we only need to use a tight lower bound of opacity. Our framework offers this capability. Further, like all sampling methods, as long as the underlying continuous space function is relatively smooth compared to the sampling rate, the discrete representation is accurate with proper interpolation. We observe that POFs normally do have a very smooth en-

velope. Therefore, we believe the approach of POF is conservative in general. Finally, we do not know of an automatic mechanism to determine a minimal necessary sampling resolution of POF.

Our parallel rendering algorithm used for testing the efficacy of POF performed well for orthogonal projection. As a part of our future work, we would like to further investigate the effectiveness of POFs for perspective views. Our algorithm works correctly for a family of opacity transfer functions. However, more analysis might be necessary to improve it for even better culling performances without estimation, especially when multiple bases are used. In addition, we plan to provide coarse-grained visibility culling for hardware implementations of volume rendering, and further extend coarse-grained visibility culling using POFs to view-dependent isosurface extraction, as well as time-varying volume visualization. The concept of POF can also be used in hierarchical volume rendering and we will explore that in the future.

We understand that several other acceleration methods exist in large volume visualization, including compression techniques [Guthe et al. 2002] and hardware support [Lum et al. 2002]. POF can be complementary to these methods.

Acknowledgments

The work was supported in part by the Mathematics, Information and Computational Sciences Office, Office of Advanced Scientific Computing Research, U. S. Department of Energy, under contract No. DE-AC05-00OR22725 with UT-Battelle, LLC and in part by NSF ACR-0118915, NASA grant NCC-1261, Ameritech Faculty Fellowship, and Ohio State Seed Grant. Special thanks to Professor Jack Dongarra and Clay England at University of Tennessee, Don Stredney, Dennis Sessanna and Jason Bryan from Ohio Supercomputer Center, and Professor Dhabaleswar Panda at The Ohio State University for providing the test environment. The Visible Woman dataset is provided by the National Library of Medicine. We also thank the anonymous reviewers for their useful comments and suggestions.

References

BAJAJ, C., IHM, I., PARK, S., AND SONG, D. 2000. Compression-based ray casting of very large volume data in distributed environments. In *HPC-Asia 2000*, 720–725.

DURAND, F. 1999. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, Grenoble I. http://www-imagis.imag.fr.

GAO, J., AND SHEN, H.-W. 2001. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *2001 IEEE Symposium in Parallel and Large Data Visualization and Graphics*.

GREENE, N. 1996. Hierarchical polygon tiling with coverage masks. In *ACM SIGGRAPH 96*, ACM SIGGRAPH, 65–74.

GUTHE, S., WAND, M., GONSER, J., AND W., S. 2002. Interactive rendering of large volume data sets. In *IEEE Visualization'02*.

HUANG, J., MUELLER, K., SHAREEF, N., AND CRAWFIS, R. 2000. Fastsplats: Optimized splatting on rectilinear grids. In *IEEE Visualization '00*, 219–227.

HUANG, J., SHAREEF, N., CRAWFIS, R., SADAYAPPAN, P., AND MUELLER, K. 2000. A parallel splatting algorithm with occlusion culling. In *3rd Eurographics Workshop on Parallel Graphics and Visualization*, Girona, Spain, 125–132.

KINDLMANN, G., AND DURKIN, J. W. 1998. Semi-automatic generation of transfer functions for direct volume rendering. In *IEEE/ACM Symposium on Volume Visualization'98*, Research Triangle Park, NC, 79–86.

KLOSOWSKI, J., AND SILVA, C. 2001. Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Transactions on Visualization and Computer Graphics 7*, 4, 365–379.

KNISS, J., KINDLMANN, G., AND HANSEN, C. 2001. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. In *IEEE Visualization'01*.

LAW, A., AND YAGEL, R. 1996. Multi-frame thrashless ray casting with advancing ray-front. In *Graphics Interface '96*, 70–77.

LEVOY, M. 1988. Display of surfaces from volume data. *IEEE Computer Graphics and Applications 8*, 5, 29–37.

LEVOY, M. 1990. Efficient ray tracing of volume data. *ACM Transactions on Graphics 9*, 3, 245–261.

LIU, Z., FINKELSTEIN, A., AND LI, K. 2001. Progressive view-dependent isosurface propagation. In *IEEE TCVG Symposium on Visualization(VisSym'01)*.

LIVNAT, Y., AND HANSEN, C. 1998. View dependent isosurface extraction. In *IEEE Visualization '98*, 175–180.

LUM, E., MA, K., AND CLYNE, J. 2002. A hardware-assisted scalable solution for interactive volume rendering of time-varying data. *IEEE Transactions on Visualization and Computer Graphics 8*, 3, 286–301.

MA, K.-L., AND CROCKETT, T. 1997. A scalable, cell-projection volume rendering algorithm for 3d unstructured data. In *Proc. of 1997 Symposium on Parallel Rendering*, IEEE CS Press, 95–104.

MA, K.-L., PAINTER, J. S., HANSEN, C. D., AND KROGH, M. F. 1994. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications 14*, 4, 59–68.

MAX, N. 1995. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics 1*, 2.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P.-P. 1998. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98*, 233–238.

PASCUCCI, V., LANEY, D. E., FRANK, R. J., SCORZELLI, G., LINSEN, L., HAMANN, B., AND GYGI, F. 2003. Real-time monitoring of large scientific simulations. In *ACM Symposium on Applied Computing'03*, ACM Press.

PEKAR, V., WIEMKER, R., AND HEMPEL, D. 2001. Fast detection of meaningful isosurfaces for volume data visualization. In *IEEE Visualization'01*, 223–230.

PRESS, W. H., TEUKOLSKY, S. A., VETTERLING, W. T., AND FLANNERY, B. P. 1992. Numerical recipes in c: The art of scientific computing.

WONKA, P., WIMMER, M., AND SILLION, F. 2001. Instant visibility. In *EuroGraphics*, A. Chalmers and T.-M. Rhyne, Eurographics.

ZHANG, E., AND TURK, G. 2002. Visibility-guided simplification. In *IEEE Visualization'02*, 267–274.

ZHANG, H., MANOCHA, D., HUDSON, T., AND HOFF III, K. E. 1997. Visibility culling using hierarchical occlusion maps. In *ACM SIGGRAPH 97*, 77–88.

ZHANG, X., BAJAJ, C., AND RAMACHANDRAN, V. 2001. Parallel and out-of-core view dependent isocontour visualization using random data distribution. In *the Tenth SIAM Conference on Parallel Processing for Scientific Computing 2001*, SIAM Activity Group on Supercomputing.