Visibility Culling for Time-Varying Volume Rendering Using Temporal Occlusion Coherence

Jinzhu Gao* ¶
The Ohio State Univ.
Oak Ridge National Lab

Han-Wei Shen†
The Ohio State Univ.

Jian Huang‡
The Univ. of Tennessee

James Arthur Kohl§ ¶
Oak Ridge National Lab

ABSTRACT

Typically there is a high coherence in data values between neighboring time steps in an iterative scientific software simulation; this characteristic similarly contributes to a corresponding coherence in the visibility of volume blocks when these consecutive time steps are rendered. Yet traditional visibility culling algorithms were mainly designed for static data, without consideration of such potential temporal coherency. In this paper, we explore the use of *Temporal Occlusion Coherence (TOC)* to accelerate visibility culling for time-varying volume rendering. In our algorithm, the opacity of volume blocks is encoded by means of *Plenoptic Opacity Functions (POFs)*. A coherence-based block fusion technique is employed to coalesce time-coherent data blocks over a span of time steps into a single, representative block. Then POFs need only be computed for these representative blocks. To quickly determine the subvolumes that do not require updates in their visibility status for each subsequent time step, a hierarchical "TOC tree" data structure is constructed to store the spans of coherent time steps. To achieve maximal culling potential, while remaining conservative, we have extended our previous POF into an *Optimized POF (OPOF)* encoding scheme for this specific scenario. To test our general TOC and OPOF approach, we have designed a parallel time-varying volume rendering algorithm accelerated by visibility culling. Results from experimental runs on a 32-processor cluster confirm both the effectiveness and scalability of our approach.

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, time-varying data visualization, volume rendering, plenoptic opacity function, large data visualization

1 INTRODUCTION

It has become increasingly common for scientists to model complex physical phenomena using large-scale computer simulations. The dynamic nature of such physical simulations typically generates massive time-varying volumetric datasets. One example is the Richtmyer-Meshkov Turbulent Simulation [22] at Lawrence Livermore National Laboratory. This simulation is designed to study instabilities at the interface between two gases of different densities, and produces datasets containing hundreds of time steps, each being 7.5 gigabytes in size. Astrophysics simulations of supernovae [21] performed at Oak Ridge National Laboratory have generated several terabytes of data per run and will produce even larger datasets in the near future. Although past research has explored a variety of techniques for visualizing large-scale datasets, with both static and time-varying data [3, 26, 19, 9], major challenges remain to efficiently store, process and visualize such massive datasets at interactive rates, especially with the increasingly high resolutions required.

A central obstacle to efficient time-varying data visualization stems from the ever-widening disparity between the available I/O, memory and computing bandwidth, and the rapidly increasing amount of data to be visualized. Various aspects of data coherence could be exploited to reduce the amount of data that pass through the visualization pipeline. Coherence in both space and time can be applied as a means of acceleration. Often, large datasets are partitioned into spatial blocks. Here we refer to all blocks residing in the same spatial location over time as a set of "co-spatial" blocks. A number of different types of coherence can be leveraged among co-spatial blocks. Many existing algorithms focus on coherence in raw data values among co-spatial blocks. If a set of co-spatial blocks do not vary over time, then much I/O and computing overhead can be saved.

Previous research [15, 17, 11, 4] has suggested that a considerable portion of a large dataset is often invisible due to the spatial occlusion. We hypothesize that the visibility of a set of co-spatial blocks could also be correlated over time. In fact, we observe that an invisible block at time step t often remains invisible at time step $t + \triangle t$ ($\triangle t$ is a small integer value), especially when the view angle doesn't change significantly over the course of the entire rendering sequence. An obvious approach to capitalize on visibility culling in time-varying volume rendering might be to apply a standard static data visibility culling scheme repeatedly to each individual time step. However, this simplistic approach will likely incur unnecessary visibility estimation overhead for subsequent time steps, without utilizing the visibility coherence. A modified scheme is required to apply minimal re-computation of the visibility from time step to time step. Such extensions to culling algorithms should also maintain high efficiency in the context of accelerating existing parallel solutions. In this paper, we present our work to develop an efficient visibility culling framework for scalable parallel volume rendering of large-scale time-varying datasets.

The remainder of this paper is organized as follows. The related work of visibility culling and time-varying data visualization are briefly discussed in Section 2. The overview and details of our visibility culling scheme for time-varying volume rendering and the experimental results are presented in Sections 3 through 7. In Section 8, we summarize our contributions and discuss future work.

2 RELATED WORK

Much research has been explored in algorithms for visibility culling and time-varying data visualization, but primarily in separate contexts. Integrating visibility culling to accelerate time-varying data

*gao.52@osu.edu. Currently affiliated with Oak Ridge National Lab.
†hwshen@cis.ohio-state.edu
‡huangj@cs.utk.edu
§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.

visualization has not been widely studied. Below we briefly review some related research work.

2.1 Visibility Culling

Visibility culling, also known as occlusion culling, is an effective technique for reducing unnecessary rendering computation by eliminating invisible portions of data before visualization. Visibility acceleration has become widely used in many polygon rendering applications; a thorough survey can be found in [2].

As pioneered in work on early ray termination [15], visibility culling has been applied both in direct volume rendering and isosurface extraction, especially when dealing with large-scale datasets. Law and Yagel [14] presented a ray-front scheme that employs visibility culling. In the image-aligned sheet-based splatting algorithm [11], both individual and groups of voxels can be culled away when the corresponding screen footprint is covered by fully opaque pixels. Guthe and Strasser [8] applied visibility test to multiresolution volume rendering which also allows the change of transfer function. Livnat and Hansen [17] introduced a view-dependent algorithm for isosurface extraction. Parker *et al.* [24] developed a highly efficient ray-casting system to visualize view-dependent isosurfaces in volume datasets without explicit extraction of the surfaced triangles. Later, Liu *et al.* [16] described a progressive view-dependent isosurface extraction algorithm. This approach determines visible voxels by casting a small number of viewing rays and then propagating the visibility information up from these "seed" voxels to obtain the full visibility information for the volume. All of these visibility acceleration methods are very effective for datasets exhibiting sufficient opaqueness.

Beyond accelerating the sequential visualization algorithms using visibility heuristics, a few methods have been developed to accelerate parallel visualization. Huang *et al.* [12] developed a visibility-assisted parallel splatting algorithm for volume datasets with moderate to heavy occlusion. To accelerate parallel isosurface extraction, Gao and Shen [5, 6] proposed a progressive visibility culling method that efficiently eliminates invisible isosurface triangles, achieving satisfactory parallel speedups. Recently, Gao *et al.* [4] proposed a highly-scalable visibility culling method based on Plenoptic Opacity Functions (POFs), which will be discussed in more detail as part of our design overview in Section 3.

2.2 Time-Varying Volume Visualization

Efficient algorithms for time-varying data visualization have become increasingly important to the visualization community. A number of algorithms have been developed that enable high interactivity and improve data understanding.

A wealth of literature exists on the compression of time-varying data. Blocks at different time steps can be compressed using Wavelet Transforms [31]. In a hierarchical manner, Discrete Cosine Transforms, Vector Quantization, 3-D Wavelet Transforms and MPEG compression schemes [18, 7, 29] have all been explored. Hierarchical compression schemes have been applied to capitalize on inter-block dependencies, and to eliminate insignificant frequency coefficients that correspond to minor features in the data. Exploiting data coherence, Shen and Johnson [27] proposed a differential volume rendering strategy that was shown to reduce rendering time and storage space by upwards of 90% for two test data sets. Shen *et al.* [26] introduced a time-space partitioning (TSP) tree structure to capture both spatial and temporal coherence in time-varying data sets. Sutton and Hansen [30] presented a temporal Branch-on-Need tree structure for efficient time-varying isosurface extraction. Using quite a different compression approach, Neophytou and Mueller [23] converted a 4-D dataset from a regular grid into a Body-Centered Cartesian grid and achieved better efficiency.

Other research on time-varying data visualization includes the tracking of features in a time-varying dataset [28, 13], high-performance parallel algorithms [25, 19] and various alternative methods [10, 1, 32].

3 Algorithm Overview

Our design goal is to utilize *Temporal Occlusion Coherence* (TOC), that is, the temporal coherence in terms of a block's capability to occlude other blocks behind it along the viewing direction, to accelerate volume rendering of time-varying datasets. This work exploits a visibility culling scheme based on Plenoptic Opacity Functions (POF) [4]. This scheme was introduced to encode volume blocks' occluding capabilities, which we will refer to as *opacities*, from all external viewpoints. After the POF-encoding preprocessing step is completed, efficient visibility culling can be performed at run time. A framework using POF was shown to achieve highly scalable parallel volume rendering. Here we further develop the concept of POF to accelerate the visibility culling for time-varying data visualization.

Due to the potentially overwhelming size of the time-varying data, it is desirable to determine the minimal possible set of visible blocks. A mathematically sound mechanism is needed to efficiently encode the opacity of representative blocks, in an effective but conservative manner. The POF scheme is generally a viable technique for this purpose, however, it can be overly conservative for certain cases, which could hamper the achievable acceleration. Even a moderate increase in the amount of false positives, where an invisible block is classified as visible during visibility estimation, could result in a considerable increase in I/O and rendering time. Our initial study of this work shows that the original POF scheme needs to be optimized to minimize such over-conservativeness. We call our new scheme *Optimized Plenoptic Opacity Function (OPOF)*. We have carefully extended our original POF concept to construct an OPOF scheme; this new scheme is still conservative but encodes the opacity distribution of any block with a much tighter lower bound. Significant improvements in culling performance have been observed in experiments using OPOFs (see Section 7).

To determine the temporal coherence of the opacity among co-spatial blocks, we need a means for effective detection of opacity variations for each block over time. Traditionally, researchers have analyzed the differences in raw data values to search for temporal coherence. However, opacity also depends on the specific transfer functions utilized. In our algorithm, we attempt to find the opacity variations among co-spatial blocks. If the opacity variations are below a user-specified threshold for two co-spatial blocks at neighboring time steps, then we classify these blocks as temporally coherent. For each set of co-spatial blocks, a set of contiguous time spans defines the temporal coherence. For each time span, the co-spatial blocks are *fused* into a representative block, and an OPOF is computed for this representative block.

Using pre-computed OPOFs at each time step, the opacity of each representative block can easily be computed at run time. Clearly, there can be a high coherence in occluding capability among representative blocks residing in local spatial vicinities. To further accelerate the visibility estimation process, a hierarchical TOC tree is constructed during pre-processing. Using the TOC tree, those blocks whose opacity has changed versus the previous time step can be quickly identified. This allows us to perform visibility estimation only for the blocks whose visibility has been affected by those blocks.

To demonstrate the viability of our approach, we present a parallel time-varying volume rendering system that leverages both TOC and OPOF in the visibility acceleration. In Sections 4, 5 and 6, we discuss OPOF, block fusion and TOC tree construction, respectively, in detail for our time-varying volume rendering system.

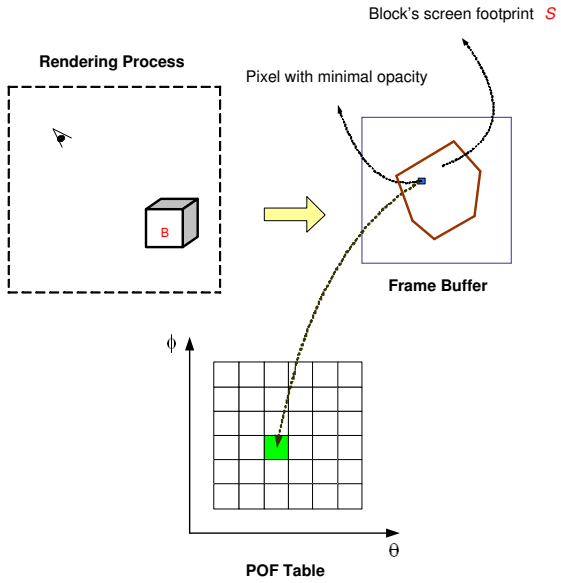


Figure 1: For each view (upper-left), the opacity channel of a volume block B is rendered into a frame buffer. Suppose the pixel shaded with blue has the minimal opacity among all non-empty pixels. This minimal opacity value is stored into the entry shaded with green in a 2D table indexed by θ and ϕ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

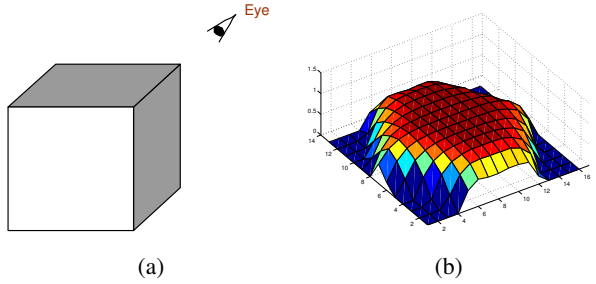


Figure 2: Illustration of conservative POF computations: (a) A uniform block is rendered from a sample view, and (b) Scalar plot of POF as a height field showing the values in the opacity buffer.

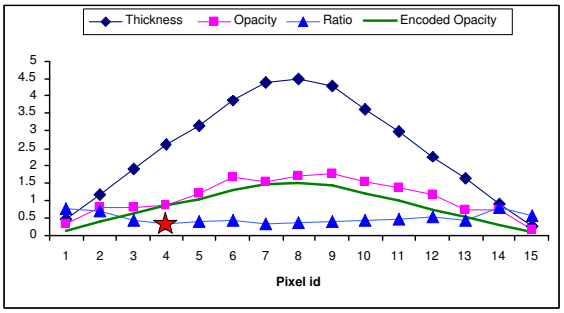


Figure 3: 2-D examples of the thickness mask D, the opacity mask O, the ratio mask τ ($\tau_p = O_p/D_p$ for each pixel p) and the encoded opacity mask O' ($O'_p = D_p MIN(\tau_p)$) for a block during the OPOF computation. A red star points out the minimal ratio $MIN(\tau_p)$. Note that the thickness values and opacity values are selected only for illustration purpose.

4 Optimized Plenoptic Opacity Function (OPOF)

In [4], we proposed a Plenoptic Opacity Function (POF) scheme, which encodes the minimal occluding capability, or the *opacity*, of a volume block under all possible external views. For a given view, considering all the rays intersecting a block, the minimal opacity value accumulated within the block is defined as its opacity under this view. The computational process for computing a POF is briefly illustrated in Figure 1. A 2-D POF table (Figure 1, bottom), parameterized by the angular part of spherical coordinates, θ and ϕ, is built as follows. For any sample view (θ_i,ϕ_i) (Figure 1, upper-left), an opacity image S of a block B is rendered. The lowest value among all pixels inside S shows the minimal occluding capability of B, which is stored at the location corresponding to (θ_i,ϕ_i) in the POF table. To be more space-efficient, a discrete POF table can be encoded by a polynomial or spline, as long as the process remains conservative. The POF can fully capture the view dependent variation of a block's opacity under all possible external views, so the opacity of the block for any specific view can always be obtained by simply evaluating the POF. By accumulating the opacities only at the block level, visible blocks can be quickly identified at run time.

Our original POF scheme has proven to be very effective when used in large-scale parallel volume rendering. However, using the minimum opacity of a block as the overall opacity can sometimes be too conservative, potentially producing false positives. For instance, with blocks of uniform voxel values, the minimum opacity will most likely be found at the corners of the block. Figure 2 shows an example $32\times32\times32$ block with uniform voxel values all mapped to an opacity of 0.1, rendered at a $(45,45)$ viewing angle. The minimal opacity of the block is 0.1, found at the corners, yet the opacities at the center are significantly higher. In a large-scale simulation, such homogeneous blocks commonly exist, especially when the size of each block is relatively small. In fact, for the Richtmyer-Meshkov Turbulent data set [22], which has a $1024\times1024\times960$ spatial resolution and a block size of $32\times32\times32$, about 70% of all blocks have very small internal variations in their voxel values. For those blocks, the minimum accumulated opacity will be too conservative. This resulting opacity estimation will produce a suboptimal culling rate for such time-varying datasets. An optimized opacity encoding scheme is necessary to alleviate this over-conservativeness.

We have developed an optimized algorithm based on the original POF scheme, called *Optimized Plenoptic Opacity Function* (OPOF) scheme (Refer to Section 7 and Figure 11 for the comparison of culling effect between POF and OPOF). This revised scheme is based on the following observation: in a homogeneous block, the opacity accumulated at each pixel is related to the length of the corresponding ray segment inside the block. Here, we refer to this length as the *thickness* of a pixel. By computing the thickness for all pixels in the block's screen projection S, we obtain a thickness mask D. We can also compute an opacity mask O, by accumulating the opacities along the ray segments inside each block. According to [20], the opacity along a viewing ray is computed as $\alpha = 1-e^{-\int_0^d \tau(t)\,dt}$, where $\tau(t)$ is the extinction coefficient defined along the viewing ray and α is the resulting opacity. Assuming uniform voxel values along the corresponding ray segment, for every pixel p inside the block's screen projection S, the resulting opacity becomes $\alpha_p = 1-e^{-D_p\tau}$ where α_p and D_p represent the opacity and the thickness at the pixel p. The term $e^{-D_p\tau}$ can be approximated by the first two terms of its Taylor expansion: $1-D_p\tau$, thus α_p can be approximated by $D_p\tau$. Therefore, the opacity of a ray segment can be approximated by a linear function of its thickness.

Of course, not all volume blocks are uniform. To account for this

case, we compute the ratio between the opacity mask and the thickness mask on per pixel basis. The minimal ratio is used to encode the opacity of the block to remain conservative. The minimal ratios from all sample views are stored in an OPOF for runtime visibility estimation. This process is illustrated in Figure 3.

To estimate the opacity of a block at run time, we first compute its thickness mask D, which is the same for all blocks if orthographic projection is used. Then, after looking up the ratio τ from its pre-computed OPOFs, a conservative opacity mask O can be easily computed from D as: $O_p = D_p\tau$ for every pixel p inside the block's screen projection S. This opacity mask serves as a conservative but tighter estimate of the block's opacity. In the worst case, where the block's raw data distribution is irregular and therefore causes the resulting opacity to be unrelated to the thickness, then the opacity computed will reduce to the minimal opacity as used in the original POF algorithm. This case still remains conservative.

5 Coherence Based Block Fusion

Having constructed a viable and efficient scheme to encode the opacity of volume blocks, we must now develop a mechanism for extracting the temporal coherence of blocks' opacity. In this section, we describe the details of our metric for identifying this temporal occlusion coherence between consecutive time steps for co-spatial blocks. This metric relies on a coherence-based block fusion method that combines coherent co-spatial blocks into a single representative block. We also discuss a hierarchical data structure, a *Temporal Occlusion Coherence* (TOC) tree, that enables efficient querying of these representative blocks as time progresses.

5.1 Opacity Enhanced Temporal Difference Metric

To determine whether data coherence exists between co-spatial blocks for two consecutive time steps, a difference metric is defined. For the purpose of visibility culling, the difference metric need only measure the variance in the blocks' opacity which depends on the raw data as well as the underlying opacity transfer function. To distinguish this metric from traditional temporal data coherence, we refer to such a categorization as *temporal occlusion coherence*. For a given opacity transfer function, we define the distance between two scalar raw data values as:

$$Dis(v_1, v_2) = |Opa(v_1) - Opa(v_2)| \tag{1}$$

where $Opa(v)$ is the opacity for the data value v. The distance is defined as the difference between corresponding opacity values.

Let $B_i(t)$ represent the i^{th} block at t^{th} time step. The difference between $B_i(t_1)$ and $B_i(t_2)$ can be defined as:

$$Diff(t_1, t_2) = \frac{\sum_{(x,y,z)\in B_i} (Dis(b_i(x,y,z,t_1), b_i(x,y,z,t_2)))^2}{N} \tag{2}$$

where $b_i(x,y,z,t)$ is the data value at the location (x,y,z) in $B_i(t)$ and N is the number of voxels that are non-empty in either $B_i(t_1)$ or $B_i(t_2)$.

5.2 Block Fusion

Based on the difference metric defined above, we can fuse two data blocks, $B_i(t)$ and $B_i(t')$ into a new representative data block $B_{i,[t,t']}$, if $Diff(t,t')$ is below a certain small difference threshold, such as 0.001. In particular, the fusion operation performs the following:

$$b_{i,[t,t']} = \begin{cases} b_{i,t} & if(Opa(b_{i,t}) \le Opa(b_{i,t'})) \\ b_{i,t'} & Otherwise \end{cases} \tag{3}$$

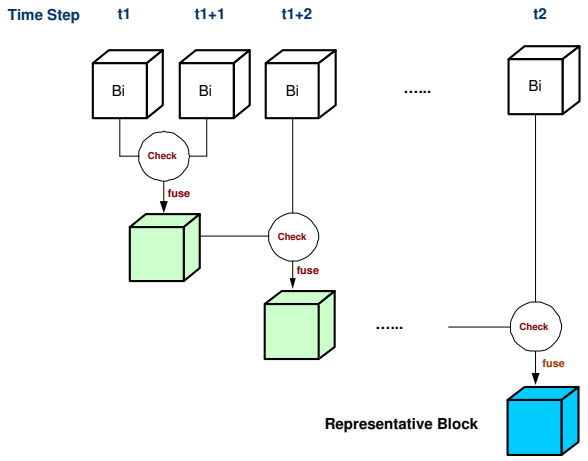


Figure 4: An example of block fusion. $B_i(t_1)$, $B_i(t_1+1)$ ... $B_i(t_2)$ are fused into a representative block based on $Diff(t,t+1)$.

where $b_{i,t}$ represents the data value at location (x,y,z) in the i^{th} block at t^{th} time step. The new fused block will keep the raw value associated with the lower opacity for each (x,y,z) coordinate.

Our algorithm fuses together as many blocks as possible to construct coherent time spans. This is achieved by performing the block fusion incrementally. Figure 4 illustrates the process. Starting from the block at the first time step t_1, we calculate the difference between co-spatial blocks at time step t_1 and t_1+1. If the difference $Diff(t_1, t_1+1)$ is smaller than our threshold, then we fuse these two blocks together. The resulting fused block will then be applied to continue the coherence test with subsequent blocks. After time step t_2, if the temporal difference rises above our difference threshold, then the fusion process is terminated for that particular fused block, and a new round of block fusion will begin from time step t_2+1. The complete fused block is the *representative block*, denoted as $B_{[t_1,t_2]}$, for the coherent time span t_1 through t_2. We assume that the opacity of this block throughout the coherent time span is invariant for any given view. Therefore, only a single OPOF need be computed for the representative block $B_{[t_1,t_2]}$, offering an additional savings in OPOF computation and storage.

As mentioned in Section 4, a conservative opacity mask for each block can be easily computed at run time, based on the thickness mask D, which is same for all blocks in the volume if an orthogonal projection is used. After constructing the opacity mask for each representative block, the visibility estimation can then be done by front-to-back compositing.

As discussed in [4], an initial opacity transfer function can be decomposed into several basis functions. These bases, together with their scaling factors, can be used to generate a *family* of transfer functions. Using this methodology, visibility estimation can be performed even when the transfer function changes at run time. To handle this case, the above computational steps must be applied to each basis function. The coherent time spans determined for each basis are likely to be different. The runtime computation of an opacity mask for any transfer function in the family will be similar to the opacity calculation described in [4]. That is, the opacity value of a pixel p in the overall opacity mask is:

$$\alpha'_p = 1 - \prod_{i=1}^{p} (1 - \alpha_{p,i})^{k_i} \tag{4}$$

where α'_p is the opacity value at pixel p in the final opacity mask, and $\alpha_{p,i}$ is the opacity value at pixel p in the opacity mask computed

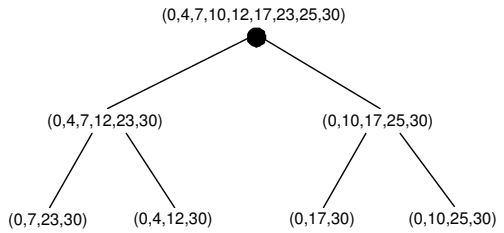


Figure 5: An example of a TOC tree (a binary tree is used for illustration purposes). The numbers represent the ids of time steps separating adjacent coherent time spans.

for the i^{th} basis function.

5.3 Temporal Occlusion Coherence Tree Construction

Block fusion allows for utilizing temporal coherence. At each time step, it is only necessary to perform a visibility check for a volume block if those other blocks that affect its visibility have experienced a change in their opacities. Even so, this search through the whole spatial volume, for representative blocks that have expired, can still be optimized for better efficiency. In the spatial domain, it is expected that neighboring blocks in a local vicinity will have similar coherent time spans. Hence, to enable an efficient search, a hierarchical Octree structure is computed spatially. At each tree node, the coherent time spans of the corresponding co-spatial (meta)-blocks are stored. This tree is referred to as *Temporal Occlusion Coherence* (TOC) tree.

The coherent spans stored in the tree nodes are computed in a bottom-up manner. The leaf nodes correspond to the true volume blocks in the partitioned volume. The coherent time spans on the leaf nodes are the result of the block fusion process. On each internal node, the coherent time span can be calculated from the time spans of all child nodes as shown in Figure 5.

At run time, the TOC tree is traversed for each new time step $t+1$ in a top-down manner. When visiting each tree node, our algorithm checks t and $t+1$ to verify that they belong to the same coherent time span. If not, then the search descends to the next lower level to check this condition for each of the child nodes. When a leaf node is reached, if the two time steps still do not belong to the same coherent time span, the corresponding block is marked as having had its opacity changed, thereby affecting the visibility of all other blocks behind it along the viewing direction. Upon traversing the TOC tree in this way, all volume blocks whose opacities have changed will have been identified. Then the algorithm commences the visibility check for all volume blocks whose visibility is affected by a marked block. The visibility of other blocks remains the same as in the previous time step t. This approach significantly reduces the visibility estimation cost if data at two consecutive time steps are highly coherent.

6 Parallel Time-Varying Volume Rendering with OPOF Assisted Visibility Culling

The data flow for the pre-processing and run-time phases of our parallel time-varying volume rendering algorithm with visibility acceleration is illustrated in Figure 6. Data distribution, block fusion and OPOF pre-computation are the three major tasks performed at the pre-processing stage. The data at each time step is first partitioned into volume blocks, which are then distributed to processors along a space-filling curve. As discussed in [4], such static data distribution allows the run-time algorithm to achieve a balanced workload, without the need for dynamic data redistribution. Controlled by a

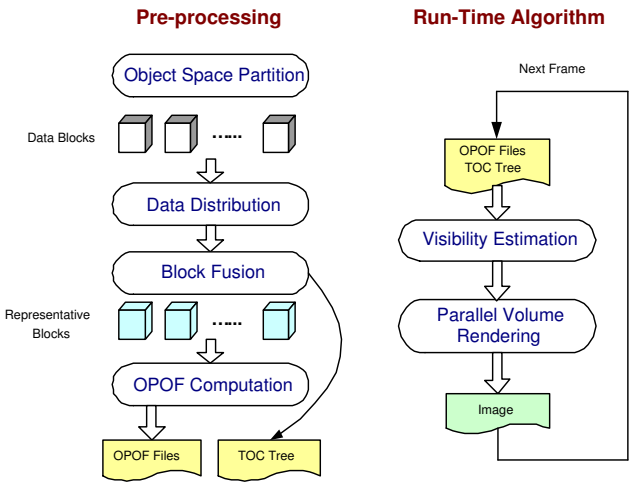


Figure 6: The data flow of our parallel time-varying volume rendering algorithm accelerated by visibility culling.

user-supplied difference threshold, our algorithm fuses co-spatial blocks into a set of representative blocks with coherent time spans. Based on the fusion results, a TOC tree is constructed to manage the coherent time intervals for every spatial partition. The tree is used to reduce the run-time visibility estimation cost. Finally, an OPOF is pre-computed for each representative block and stored for run-time usage. Due to the negligible storage required for OPOFs, we can afford to replicate a copy of the entire OPOF results for all blocks on every node, reducing run-time communication cost.

Our run-time time-varying volume rendering algorithm performs both the visibility estimation and parallel volume rendering. The goal of the visibility estimation is to identify all visible blocks before starting the parallel rendering. By doing this, our algorithm avoids the global synchronization that is needed in other multi-pass algorithms [5]. Workload balancing is also easier as the exact rendering load is known in advance. Similarly to the method used in [4], our visibility culling scheme is done in parallel as follows. First, the bounding box of the whole volume's screen projection is partitioned into image tiles of equal size, where the number of tiles equals the number of processors. Each processor is assigned one tile and is responsible for identifying which visible block's screen footprints overlap with the tile, as well as compositing the final image for the assigned tile. Then, for a given view (θ,ϕ), all volume blocks' bounding boxes are checked in a front-to-back order. Only those blocks whose screen footprints overlap with the assigned tile need be tested for visibility. An opacity buffer, of a size equal to the tile size, is used to store an accumulated opacity value for each pixel in the tile. To test the visibility of each volume block, the screen footprint of the block is computed based on the opacity buffer queried. If all values inside a 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 its opacity, computed from the thickness buffer and the block's OPOF, will be composited into the opacity values in the block's screen footprint. The visibility estimation tests are done in parallel to further reduce the cost. At the end of visibility estimation, a global communication is done so that all processors know the indices of all visible blocks. Each processor then starts to load and render the visible blocks locally, as pre-assigned during the data distribution. The partial image rendered for each block is sent to all processors whose tiles overlap with the partial image. When the rendering process finishes, each processor composites all the partial images it has received to generate the final image for its tile.

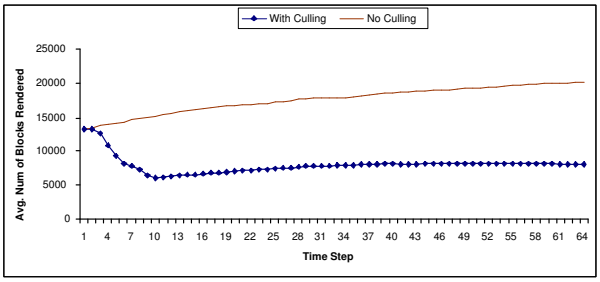
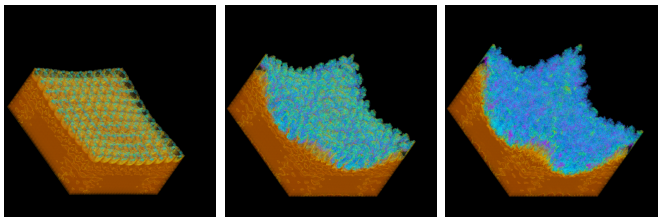


Figure 7: Average number of non-empty blocks rendered for 18 sample views, with or without visibility culling, at each time step.



(a) (b) (c)

Figure 8: Rendering results of: (a) 8^{th} time step (8671 visible blocks, 6377 invisible blocks), (b) 36^{th} time step (12138 visible blocks, 6200 invisible blocks) and (c) 63^{th} time step (12878 visible blocks, 7388 invisible blocks), under the view (225, 45).

Finally, a host processor collects these image tiles to form the final image.

Utilizing temporal occlusion coherence, we can effectively reduce the visibility estimation time over the course of a time-varying volume visualization. For the first time step, visibility must be estimated for every block, in the front-to-back order as described above. After that, however, for each subsequent time step the visibility estimation is only needed for those blocks whose visibility status may have changed, as determined via the temporal occlusion coherence tree described in Section 5.3.

7 Results

In this section, we present and analyze experimental results obtained using a time-varying volume rendering system with TOC-accelerated visibility culling. A software ray casting algorithm was implemented to perform volume rendering and the image size was set to be 512×512. Our testing platform is a PC cluster with 32 2.4GHz Pentium IV Xeon processors, with an interconnection by Dolphin Networks. Our primary testing dataset came from a Richtmyer-Mevhkov Instability (RMI) simulation at Lawrence Livermore National Laboratory. At each of the 274 time steps, the simulation produced 7.5GB of data with a spatial resolution of $2048 \times 2048 \times 1920$. Due to the storage limitation, we chose 64 time steps from the data set, starting with the 4th time step and picking one out of every four time steps, with each spatial dimension downsampled to half of its original resolution. The total size of this selection is 60GB. Each of the experiments assumes a fixed viewing direction until all time steps have been visualized. To further demonstrate the benefit of OPOF, as compared to our previous POF approach, we also ran experiments using a static volume dataset, the $512 \times 512 \times 1728$ Visible Woman. During pre-processing, each dataset was partitioned into $32 \times 32 \times 32$ data blocks, which were then evenly distributed to 32 processors along a space filling curve.

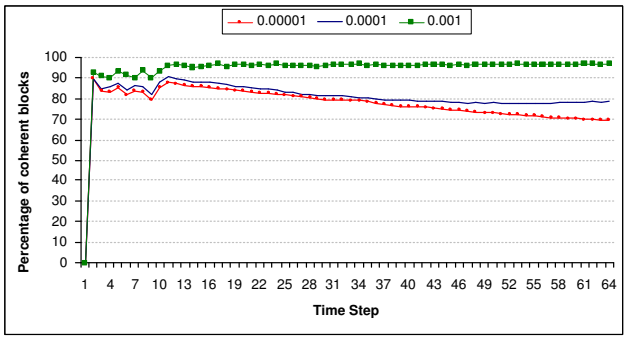


Figure 9: Percentage of the blocks that utilize temporal occlusion coherence at each time step for three difference thresholds.

We tested visibility acceleration for the RMI dataset from a set of sample views. Under a sample view from which the temporal evolvement of the data is clearly shown, without leveraging visibility culling, 64 time steps of the RMI dataset required approximately 7.6 minutes to complete on 32 processors, including the time for both parallel rendering and parallel I/O. Utilizing our TOC-accelerated visibility culling framework, we reduced the total time to less than 1.5 minutes (about 1.4 seconds per time step on average), or an 80% reduction in the rendering and I/O costs for this test view. Figure 7 shows the average culling effects, as measured by the average number of blocks rendered under 18 sample views, spaced 20 degrees apart, for each time step. On average, 50% of nonempty blocks were culled away for most time steps. A few sample images of the RMI dataset are shown in Figure 8.

Temporal occlusion coherence typically exists to some degree in all time-varying datasets, which is captured by a threshold in our algorithm. To demonstrate the existence of such coherence, Figure 9 shows the percentage of blocks that exhibit temporal occlusion coherence at each time step, for three difference thresholds. It can be seen that the tighter the difference threshold, the less coherence is observed.

The occlusion coherence determined during block fusion must be efficiently accessible at runtime, to achieve the full potential of visibility acceleration. Using a TOC tree to heuristically analyze and fuse spatial coherence among neighboring blocks, visibility is checked only for those pixels whose occlusion accumulation is affected by the advent of a new time step. The runtime cost of visibility estimation is proportional to the number of pixels covered by the screen footprints of incoherent blocks. The overhead of TOC tree usage was negligible, at about 0.25 seconds for construction and 1.8MB total storage, for the RMI dataset with a difference threshold of 0.001. In Figure 10, we compare the number of affected pixels at each time step, with or without using the temporal occlusion coherence. Without using the temporal occlusion coherence, all pixels need be checked to update the blocks' visibility. When the temporal occlusion coherence is used, the total time for visibility determination throughout the entire time sequence was reduced from from 2.4 seconds to 1.1 seconds, using 32 processors.

As an additional benefit of temporal occlusion coherence, we need only compute one OPOF for each representative block. Although the use of OPOFs is highly efficient at runtime, OPOF precomputation still incurs some overhead, especially for very large time-varying datasets. The actual pre-computation overhead from using OPOF depends on the user-chosen difference thresholds. Table 1 compares OPOF computation time (in seconds), storage cost (in Mbytes) and the resulting number of visible blocks under a given view, for five difference thresholds. With a difference threshold of 0, essentially no temporal coherence is considered and the best

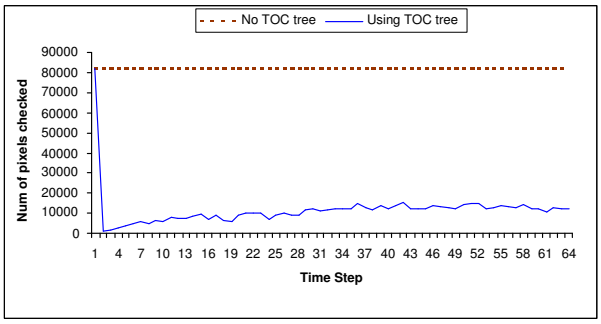


Figure 10: Comparison of the number of pixels checked for visibility estimation in each of 64 time steps, with or without the TOC tree.

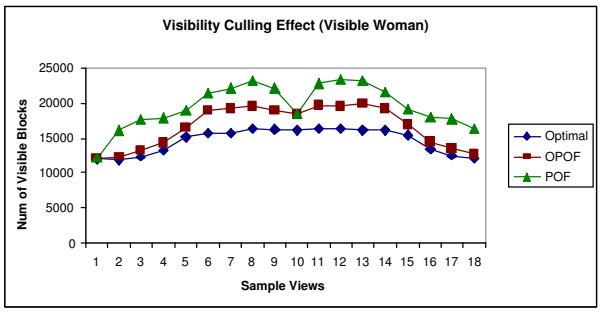
Diff Threshold	Time (min)	Storage (MB)	# Visible Blocks
0	204.7	91.9	191,082
0.00001	109.1	41.2	191,282
0.0001	77.5	35.1	191,736
0.001	25.0	10.7	194,832
0.01	8.8	2.8	1,148,677

Table 1: OPOF computation time (in seconds), storage cost (in MBytes) and number of visible blocks (under one sample view) for five difference thresholds.

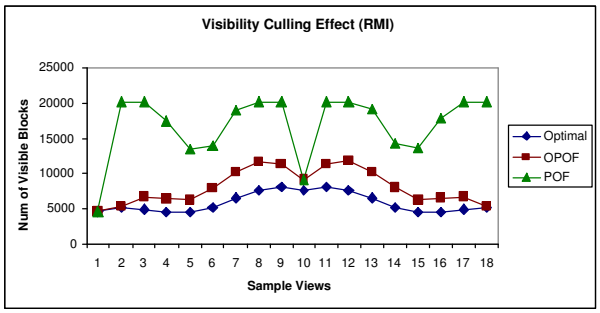
culling effects can be achieved, but at high pre-computation cost. However, even with a very small non-zero difference threshold, both computation time and storage space are used more efficiently. Note that although an increased difference threshold can save time and storage overheads with OPOF, it can ultimately have an adverse effect on culling performance if the threshold is chosen to be too large. This is because each fusion operation ensures conservativeness. The occluding capability of the new block after a fusion operation is always lower or equal to that of any original block. The higher the threshold, the more the fusion operations that are executed. The aggregate effect of a large number of fusion operations is a culling that is too conservative, which means more blocks will be considered as visible. For instance, in our experiment, using thresholds as high as 0.001 offers great culling results, but using 0.01 causes a dramatic increase in the number of blocks determined as visible, most of which are clearly false positives. Thus, we believe 0.001 is a good choice for this particular dataset.

Using OPOF, our run-time visibility culling algorithm can achieve better culling effects than our previous POF-assisted visibility culling algorithm. In Figure 11, we compare the culling effect of an early-ray termination type of culling algorithm with methods using POF and OPOF, from 18 sample views spaced 20 degrees apart around the test dataset. (Note that early-ray termination represents the theoretical limit of occlusion culling performance, although it cannot be used efficiently in practice with large-scale parallelism, due to overwhelming runtime communication for very large datasets.) All three algorithms are coarse-grained and cull away volume blocks directly, not individual voxels. For this comparison, both the Visible Woman data set and a single time step of the RMI dataset were used. By adjusting the opacity transfer function, the regions of interest, such as the skin in the Visible Woman data, were made highly opaque, but in the RMI dataset very low opacity values were used. It can be seen that, for both datasets, using OPOF can achieve culling results very close to the benchmark algorithm, while POF sometimes produces somewhat more false positives, especially for highly transparent datasets like RMI.

With our space-filling curve mechanism of data distribution, and



(a) Visible Woman



(b) RMI

Figure 11: The culling effects of three algorithms - Optimal, POF and OPOF - from 18 sample views during rotation around the Y axis, using: (a) Visible Woman data set (42,193 nonempty blocks), and (b) One time step of RMI data set (20,266 nonempty blocks).

an efficient TOC framework, our parallel time-varying volume rendering algorithm achieved balanced workload and good scalability. Figure 12 shows the number of total blocks rendered by each of 32 processors throughout 64 total time steps. During the rendering of each time step, our parallel renderer consistently achieved above 80% CPU utilization on a 32-processor PC cluster.

8 Conclusion and Future Work

In this paper, we have introduced an algorithm based on temporal occlusion coherence that is able to perform coarse-grained visibility culling for large-scale time-varying volume rendering with effectiveness, efficiency and parallel scalability. There are three essential components to this algorithm: OPOF encoding of opacities, block fusion to determine temporal occlusion coherence, and a TOC tree data structure to facilitate highly-efficient runtime visibility deter-

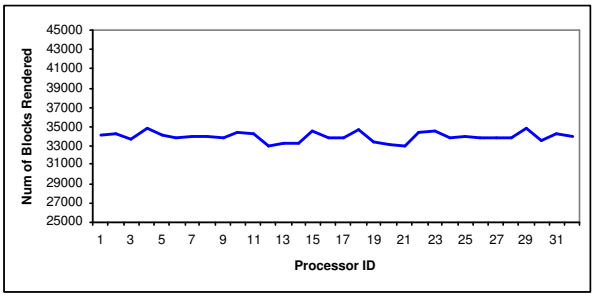


Figure 12: The number of total blocks rendered by each processor throughout 64 time steps.

mination. A parallel time-varying volume rendering algorithm accelerated by visibility culling was implemented to demonstrate the effectiveness of these components.

In the future, we plan to investigate the fully interactive exploration of time-varying datasets, allowing both runtime variation of spatial viewpoints and time tick, which we expect to be very useful to real world production scientific users. In addition, many existing techniques such as TSP-tree, hardware acceleration, multiresolution rendering and compression-accelerated rendering should all be incorporated into our system to improve its performance.

ACKNOWLEDGEMENTS

This work was supported in part by NSF grant ACI-0329323, NSF ITR grant ACI-0325934, DOE Early Career Principal Investigator award DE-FG02-03ER25572, and NSF Career Award CCF-0346883 and 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. Special thanks to Professor Jack Dongarra and Clay England at University of Tennessee, Don Stredney, Dennis Sessanna and Jason Bryan from Ohio Supercomputer Center. The RMI data set was provided by Mark Duchaineau at Lawrence Livermore National Laboratory. The Visible Woman dataset was provided by the National Library of Medicine.

REFERENCES

[1] C. Bajaj, C. Pascucci, G. Rabbiolo, and D. Schikore. Hypervolume visualization: A challenge in simplicity. In *IEEE/ACM Symposium on Volume Visualization'98*, pages 95–102, 1998.

[2] Frédo Durand. *3D Visibility: analytical study and applications*. PhD thesis, Université Joseph Fourier, Grenoble I, July 1999.

[3] L. A. Freitag and R. M. Loy. Adaptive, multiresolution visualization of large data sets using a distributed memory octree. In *Proceedings of SC99: High Performance Networking and Computing*, Portland, OR, 1999. ACM Press and IEEE Computer Society Press.

[4] J. Gao, J. Huang, H.-W. Shen, and J. A. Kohl. Visibility Culling Using Plenoptic Opacity Functions for Large Data Visualization. In *Proceedings of IEEE Visualization '03*, pages 341–348, 2003.

[5] J. Gao and H.-W. Shen. Parallel view-dependent isosurface extraction using multi-pass occlusion culling. In *2001 IEEE Symposium in Parallel and Large Data Visualization and Graphics*, 2001.

[6] J. Gao and H.-W. Shen. Hardware-assisted view-dependent isosurface extraction using spherical partition. In *Joint EUROGRAPHICS-IEEE TCVG Symposium on Visualization*, pages 267–276, 2003.

[7] S. Guthe and W. StraBer. Real-time Decompression and Visualization of Animated Volume Data. In *IEEE Visualization '01*, pages 349–356, 2001.

[8] S. Guthe and W. Strasser. Advanced Techniques for High-Quality Multi-Resolution Volume Rendering. *Computers & Graphics*, 28(1):51–58, 2004.

[9] S. Guthe, M. Wand, J. Gonser, and W. StraBer. Interactive Rendering of Large Volume Data Sets. In *Proceedings of IEEE Visualization '02*, pages 53–60, 2002.

[10] A. Hansen and R. Cross. Interactive Visualization Methods for Four Dimensions. In *IEEE Visualization '93*, pages 196–203, 1993.

[11] J. Huang, K. Mueller, N. Shareef, and R. Crawfis. Fastsplats: Optimized splatting on rectilinear grids. In *IEEE Visualization '00*, pages 219–227, 2000.

[12] J. Huang, N. Shareef, R. Crawfis, P. Sadayappan, and K. Mueller. A parallel splatting algorithm with occlusion culling. In *3rd Eurographics Workshop on Parallel Graphics and Visualization*, pages 125–132. Girona, Spain, 2000.

[13] G. Ji, H.-W. Shen, and R. Wenger. Volume Tracking Using Higher Dimensional Isocontouring. In *IEEE Visualization '03*, 2003.

[14] A. Law and R. Yagel. Multi-frame thrashless ray casting with advancing ray-front. In *Graphics Interface '96*, pages 70–77, 1996.

[15] M. Levoy. Efficient ray tracing of volume data. *ACM Transactions on Graphics*, 9(3):245–261, 1990.

[16] Z. Liu, A. Finkelstein, and K. Li. Progressive view-dependent isosurface propagation. In *Vissym'01*, 2001.

[17] Y. Livnat and C. Hansen. View dependent isosurface extraction. In *IEEE Visualization '98*, pages 175–180, 1998.

[18] E. Lum, K. Ma, and J. Clyne. Texture Hardware Assisted Rendering of Time-varying Volume Data. In *IEEE Visualization '01*, pages 263–270, 2001.

[19] K.-L. Ma and D. M. Camp. High performance visualization of time-varying volume data over a wide-area network status. In *Proceedings of Supercomputing Conference '00*, pages 59–59, 2000.

[20] N. Max. Optical models for direct volume rendering. *IEEE Transactions on Visualization and Computer Graphics*, 1(2), 1995.

[21] O.E.B. Messer, M. Liebendoerfer, W. R. Hix, A. Mezzacappa, and S. W. Bruenn. The Impact of Improved Weak Interaction Physics in Core-Collapse Supernovae Simulations. In *Proceedings of the ESO/MPA/MPE Workshop*, 2002.

[22] A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White. Very High Resolution Simulation of Compressible Turbulence on the IBM-SP System. In *Proceedings of Supercomputing Conference '99*, 1999.

[23] N. Neophytou and K. Mueller. Space-time points: 4d splatting on efficient grids. In *IEEE/ACM Symposium on Volume Visualization'02*, pages 97–106, 2002.

[24] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98*, pages 233–238, 1998.

[25] S. G. Parker and C. R. Johnson. Scirun: A scientific programming environment for computational steering. In *ACM/IEEE Supercomputing'95*, 1995.

[26] H.-W. Shen, L.-J. Chiang, and K.-L. Ma. A fast volume rendering algorithm for time-varying fields using a time-space partitioning (tsp) tree. In *IEEE Visualization'99*, pages 371–377, 1999.

[27] H.-W. Shen and C. Johnson. Differential volume rendering: A fast volume visualization technique for flow animation. In *IEEE Visualization '94*, pages 180–187, 1994.

[28] D. Silver and X. Wang. Tracking and visualizing turbulent 3d features. *IEEE Transactions on Visualization and Computer Graphics*, 3(2), 1997.

[29] C. Sohn, C. Bajaj, and V. Siddavanahalli. Feature based volumetric video compression for interactive playback. In *IEEE/ACM Symposium on Volume Visualization'02*, pages 89–96, 2002.

[30] P. M. Sutton and C. D. Hansen. Isosurface extraction in time-varying fields using a temporal branch-on-need tree(t-bon). In *IEEE Visualization '99*, pages 147–154, 1999.

[31] R. Westermann. Compression time rendering of time-resolved volume data. In *IEEE Visualization '95*, pages 168–174, 1995.

[32] J. Woodring, C. Wang, and H.-W. Shen. High Dimensional Direct Rendering of Time-Varying Volumetric Data. In *IEEE Visualization '03*, pages 417–424, 2003.