A Parallel Preintegration Volume Rendering Algorithm Based on Adaptive Sampling

Huawei Wang · Zhiwei Ai · Yi Cao · Li Xiao

2015/9/30
DISCLAIMER

This Report can be cited as follows,

This Report is a preprint of a paper intended for publication in a journal or proceedings, changes may be made before publication.
A Parallel Preintegration Volume Rendering Algorithm Based on Adaptive Sampling

Huawei Wang, Zhiwei Ai, Yi Cao, Li Xiao
Institute of Applied Physics and Computational Mathematics, Beijing 100094, China

Abstract In order to visualize large-scale scientific data effectively, a parallel preintegration volume rendering algorithm based on adaptive sampling is presented in this paper. The algorithm sets sampling points adaptively by detecting the extremal points of a data field along the rays, so it can grasp the data variation exactly. After the data field is sampled distributedly on the processing cores, the resulting sampling points are sorted by piecewise packing orderly sampling points and then composited along each ray using the preintegration technique. In the algorithm, a static load balancing scheme based on information entropy is also proposed to balance the loads of both data reading and ray sampling. In addition, a mixed logarithmic quantization scheme is suggested to quantize depth distance so as to shorten the preintegration table while preserving the rendering quality. It is demonstrated that the presented algorithm can show inner features in a data field clearly and achieve a rendering speedup ratio between 1.8–4.4, compared with the traditional parallel volume rendering algorithm.

Keywords Parallel volume rendering, adaptive sampling, preintegration, static load balancing, information entropy

1 Introduction

With the development of supercomputers and computational simulation, thousands of CPU cores are needed to perform three-dimensional (3D) highly-refined simulations in applied science fields, e.g. inertial confinement fusion and supernova explosion, so as to obtain simulation results of high scientific confidence. The ceaseless increase of computing scale consequentially results in the quick increase of output data amount: the output data usually have several gigabytes at one time step while the total time-varying data may reach the terabyte scale. Moreover, the variables in these data fields may vary violently and behave very differently in different sub-regions. It is a great challenge to quickly and clearly visualize such massive scientific data so as to enable scientists to observe and analyze scientific phenomena effectively.

Volume rendering is one of the kernel algorithms for visualizing 3D data fields [1-4], because it can directly and effectively exhibit inner data features. Quality and speed are two key factors for volume rendering. On the one hand, inner data features need to be truly shown in high quality, and on the other hand, the interactive speed is a demand of visual analysis. In theory, we may render refined data features clearly by increasing the sampling density ceaselessly. However, both computing and compositing of sampling points are time-consuming, whose computational cost is proportional to the number of sampling points. Therefore, increasing massive sampling points will inevitably bring tremendous cost of time and memory and thus decrease the performance of a rendering system greatly.

In order to render scientific data of increasing scale efficiently, we propose an adaptive sampling based parallel preintegration volume rendering algorithm. We give an adaptive sampling method so that we can effectively...
sample a data field with less points. We further realize the parallelization of the rendering algorithm so that we can deal with large-scale data on many cores.

2 Related Work

In order to improve rendering efficiency, several adaptive sampling methods were proposed, such as empty space jumping [2], hierarchical adaptive sampling [5-6], detail-directed sampling [7] and gradient-based sampling [8]. Kraus et al. presented a GPU-based volume rendering algorithm, where the sampling distance is adaptively chosen according to an oracle [9]. Although these methods can be used to reduce sampling points, it is difficult for them to exactly grasp the variable variation and thus avoid rendering artifacts. To overcome the problem, Marchesin and Verdière proposed a GPU-based cell-projection volume rendering algorithm with adaptive sampling [10], and their sampling method can effectively grasp all monotony intervals of the rendered variable. To deal with TB-scale time-varying data output in large-scale scientific simulations, it is indispensable to develop parallel rendering algorithms in distributed-memory environments [11], but it is difficult to integrate adaptive sampling into parallel rendering due to arbitrary sampling positions.

The uniform sampling method is still used widely in main parallel rendering systems including VisIt [12]. Though the algorithms in [5] and [6] are parallel, they use a simple adaptive scheme depending on only the resolutions of AMR subgrids. In parallel rendering systems, there are multiple factors to influence the rendering efficiency, such as I/O, data structure, load balance, etc. Childs et al. presented a contract-based mechanism allowing many optimizations about these factors to be applied to each pipeline execution [13]. Moloney et al. proposed an effective dynamic load balancing strategy by pre-estimating the rendering cost of each pixel [14].

The preintegration technique is often introduced to improve the visual quality of volume rendered scenes [15]. Lum et al. proposed a quick algorithm for computing the preintegration lookup table [16], and Hajjar et al. further improved the preintegration precision by a second order polynomial interpolation [17]. Recently, Guetat et al. integrated the shading component into the preintegration table with a non-linear gradient interpolation and accordingly improved the rendering effect [18].

3 Parallel Volume Rendering Algorithm

In ray-casting volume rendering, a ray is casted from the viewpoint to each pixel in the screen, along which the data field is resampled. All sampling points will be mapped to color and opacity values from transfer functions. Compositing all sampling points on each ray according to the depth order, we then obtain the color for the pixel.
corresponding to the ray. Aiming at massive datasets produced in large-scale numerical simulations, we propose a parallel preintegration volume rendering algorithm based on adaptive sampling, as shown in Fig. 1. It is a sort-last rendering algorithm according to the classification of Molnar et al. [19]. In detail, the algorithm is described as follows:

1) In a parallel computing environment, the volume rendering engine divides original input data into many small patches and distributes them to all processing cores averagely.

2) On each processing core, the volume rendering engine adaptively samples all distributed data patches on the rays that intersect them: determining positions of sampling points and interpolating values of variables.

3) In the compositing process, the rendering image is partitioned into small blocks and distributed to all processing cores averagely, and each core takes charge of collecting all sampling points and sorting them in depth order for each responsible ray.

4) According to transfer functions and lights, each processing core calculates color and opacity values of sampling points for each responsible ray, and then composites them front-to-back relatively to the viewpoint by using the preintegration method.

5) The main engine collects all image blocks and thus obtains the whole volume rendering image.

4 Static Load Balancing

In parallel rendering, it is very time-consuming to communicate massive data between the computing nodes, thus the static load balancing strategy is usually adopted. The data must be divided and distributed to all nodes at first, and then loaded into the memory of each node. In detail, the divided data patches, which often have the same size, are averagely assigned to all processing cores in a traditional scheme, thus the time of data reading is almost the same between all cores. However, for adaptive volume rendering, the computing time of ray sampling may differ much for each data patch, and ray sampling takes a prominent ratio of the rendering time, therefore the load of ray sampling may be very uneven, which will decrease the performance of parallel rendering greatly.

In a local region, rich data information means that there are abundant features, or the data change violently, thus we need to sample the region with a high density correspondingly. Contrariwise, the data can be sampled sparsely in a smooth region containing few features. Therefore, we introduce the concept of information entropy [20] to depict the abundance degree of data features, or the degree of violent variation of the data. We regard the information entropy of a data patch as a coarse estimation of the computational amount of ray sampling so as to guide the design of an effective load balancing. In existing work, based on information entropy, Wang et al. identify and present the most essential aspects of time-varying data [21], and Wu et al. select important time steps and sub-regions or optimal viewpoints in the visualization process [22].

The information entropy is a general measure of information content contained in a dataset, which is defined as

\[ H(X) = -\sum_{i=0}^{N-1} p(x_i) \log p(x_i), \]

where the dataset is regarded as a series of occurrences of a random variable \( X \), which takes values from \( \{x_0, x_1, \ldots, x_{N-1}\} \), and \( p(x_i) \) denotes the probability of \( X = x_i \). In actual applications, a large-scale data field is usually divided into a number of data patches. For each data patch, we can establish the probability distribution function
based on a histogram, as shown in Fig. 2. We partition the variable extent into $N$ intervals, and take the variable falling into an interval as a random event. We then traverse the whole data patch and count the total number of such events for each interval. After normalizing these numbers, we obtain the probability of the variable locating in each interval. From the probability distribution, we then compute the information entropy of the data patch.

![Fig. 2. Histogram of data distribution.](image)

To enhance the rendering efficiency, we propose a static load balancing scheme based on information entropy. We take the information entropy introduced above as the pre-estimated load of ray sampling for each data patch. The information entropy is computed in advance and saved in the dataset as a part of its metadata. When assigning the data patches to the processing cores, the actual data field has not been loaded into the memory, but its metadata has been loaded above all, thus we can obtain the pre-computed information entropy and use it in load balancing. At first, we sort all the data patches in a descending order according to their pre-estimated loads of ray sampling. Then, we have the load assigning algorithm as shown in Fig. 3:

1) **Initial assignment.** Take the first $k$ data patches and assign them to all processing cores respectively in a reverse order, where $k$ is the total number of the cores.

2) **Recursive assignment of the remainder.** Assign the first $k$ data patches in the remainder to the cores respectively in turn, i.e. the data patch of bigger load to the core with smaller total load, and then sort the cores in an ascending order according to their total loads.

![Fig. 3. Sketch map of data assignment.](image)

5 Adaptive Sampling
Ray-casting volume rendering exhibits the interior information of a data field by casting rays from the viewpoint to all screen pixels and resampling the data field on these rays. Traditional parallel volume rendering methods generally adopt equal depth spacing to sample a data field while ignoring non-uniform physical variation, and accordingly they often do redundant sampling computation in smooth regions but lose important information in volatile regions. To sample a data field effectively, we need to grasp its internal variation and place sampling points only in key positions so that we can use as few sampling points as possible to depict data features clearly.

Let us firstly consider the case that a ray passes through a single hexahedral cell. Limited in the cell, the data field is defined as trilinear interpolation of its eight vertices:

\[
s(x,y,z) = s_{000}(1-x)(1-y)(1-z) + s_{100}x(1-y)(1-z) + s_{010}(1-x)y(1-z) + s_{001}(1-x)(1-y)z + s_{101}xy(1-z) + s_{110}xyz,
\]

where \( s_{ijk}, i,j,k=0,1 \) are data values of the eight vertices and \( x,y,z \in [0,1] \) are linear parameters in a 3D cell.

Denote by \((x_0,y_0,z_0)\) and \((x_1,y_1,z_1)\) the entry and exit points of the ray passing through the cell. By simple cyclic permutation of \((x,y,z)\) one can suppose \( x_0 < x_1 \), and as a result, the ray can have a simple parametric equation as follows:

\[
x = t, \quad y = a + b \cdot t, \quad z = c + d \cdot t,
\]

where \( a = y_0 - bx_0, b = (y_1 - y_0)/(x_1 - x_0), \) \( c = z_0 - dx_0, \) and \( d = (z_1 - z_0)/(x_1 - x_0) \). Putting Equation (2) into Equation (1), we then obtain the variable expression on the ray:

\[
s(t) = C_3t^3 + C_2t^2 + C_1t + C_0, \quad x_0 \leq t \leq x_1,
\]

where \( e = 1-a, f = 1-c, g = b-a, h = e+b, \)

\[
C_0 = ef s_{000} + af s_{010} + cf s_{001} + ac s_{011},
C_1 = -(de + ch) s_{000} - af s_{110} + ac s_{111} + ef s_{100} + (fg - ad) s_{010} - (de - ch) s_{001} + (ad + cg) s_{011},
C_2 = (dh + bf) s_{000} + (dg - bc) s_{011} - (df + bh) s_{001} + (bc + ad) s_{111} + (bc - dh) s_{001} - (bf + de) s_{100} + (de - bc) s_{101} + (bf - ad) s_{110},
C_3 = bd s_{111} + s_{001} + s_{100} + s_{010} - s_{000} - s_{011} - s_{101} - s_{110}.
\]

The above formula is essentially equivalent to but much simpler than the variable expression presented in [10] because it contains much less arithmetical operations such as multiplication and addition.

The variable function in Equation (3) is a polynomial of degree 3 at most, and thus has at most three monotone
intervals (see Fig. 4), which are separated by extremal points. Let \( s'(t) = 0 \), then one can obtain \( 3C_3t^2 + 2C_2t + C_1 = 0 \), from which all extremal points can be solved easily.

When a ray passes through a data patch, it will intersect a series of cells successively, and the length of one ray segment may differ much from each other even if the grid of the patch is uniform, as shown in Fig. 5. Since we have known all local monotone intervals within each cell, we can obtain the global monotone intervals in the patch by merging successive increasing intervals or decreasing ones. We will check the endpoints of the local monotone intervals one by one along the ray. An endpoint can be removed if the next interval has the same monotone property as the previous one. For instance, all blue endpoints in Fig. 5 can be removed, and accordingly we set the sampling points at the remaining red endpoints. In this way, we can grasp all global monotone intervals and effectively set the sampling points on the ray.

![Fig. 5. Determining monotone intervals from individual cells to a whole patch.](image)

In order to preserve the quality of preintegration volume rendering, we still need to add some constraints when removing an endpoint (or merging an interval). Firstly, the total length of the resulting interval should be not more than a threshold value, \( \text{tol} \_\text{depth} \). Secondly, the number of the merged intervals should be limited, e.g. not more than \( \text{tol} \_\text{num} \). Thirdly, the data variation range is expected to be not more than a value, \( \text{tol} \_\text{range} \). If any one of the constraints is violated for the resulting interval, the endpoint under consideration will not be allowed to be removed.

6 Sampling Points Sorting

In our adaptive sampling algorithm, sampling points are flexibly set according to data variation on the rays, thus they may have much different numbers and arbitrary positions on different rays, which prevents them from being accessed via depth-based indices in memory. That is to say, the resulting sampling points are unstructured and very difficult to handle, especially in a parallel computing environment. In contrast, the uniform sampling algorithm can use simple 3D arrays to manage sampling points, and moreover its integer-depth-based access mode ensures a natural view order.

Considering the flexibility of adaptive sampling, we let each ray manage its own sampling points (including positions and values) and dynamically allocate memory to store them. Because the sampling points on each ray are calculated patch by patch on multiple cores, their sequence relative to the viewpoint can not be ensured; thus
we need to sort these sampling points according to their depths after collecting them. In detail, we propose an efficient algorithm for handling sampling points as follows (see Fig. 6):

1) For each ray, there may be many data patches intersecting it, which are located on multiple cores. We sample each data patch cell by cell in the direction of the ray, and then put the resulting orderly sampling points together as a package of orderly sampling points (OSPP).

2) On each core, we collect all OSPPs for each ray assigned to it, according to the image partition.

3) For each ray, we sort the OSPPs belonging to it in depth order, and accordingly obtain the whole orderly sampling points on the ray.

![Fig. 6. Collecting and sorting OSPPs for each ray.](image)

### 7 Ray compositing

The volume rendering equation is well known as

$$I = \int_0^1 c(s(t)) e^{-\int_0^t \tau(s(u)) du} dt,$$

where $I$ is the final intensity, $c(\cdot)$ is the color transfer function and $\tau(\cdot)$ is the opacity transfer function. Note that the transfer functions are usually given in discrete forms and the data will also be quantized accordingly in software implementation. For example, $s(t_1)$ is quantized to $i$, and the corresponding color and opacity are $C_i$ and $\alpha_i$, respectively, while $s(t_2)$ is quantized to $j$. Then, in the interval $[t_1, t_2]$, the basic integral

$$\int_{t_1}^{t_2} c(s(t)) e^{-\int_{t_1}^t \tau(s(u)) du} dt$$

is reduced as

$$I_{i,j} = A_i C_i + A_{i+1} C_{i+1} + \cdots + A_j C_j,$$

where

$$A_k = (1 - \alpha_i)(1 - \alpha_{i+1})\cdots(1 - \alpha_{k-1})\alpha_k, \ k=i, i+1, \ldots, j.$$  

Before computing the discrete integral (5), the opacity correction must be applied to all $\alpha_k$:

$$\alpha_i = 1 - (1 - \overline{\alpha}_i)^{\frac{D}{p}},$$
where $D$ is the world-space depth distance of the ray in $[t_1, t_2]$ and $\alpha_k$ comes from the transfer function.

In the preintegration technique, all basic integrals like (5) are computed in advance and made as a lookup table. And then, the final intensity is obtained in real-time rendering by directly looking up these integrals and compositing them along the ray, so the rendering efficiency is enhanced. Because the depth distance of two successive sampling points can be any value in our adaptive sampling method, we can not have a unified opacity correction. Thus, we have to quantize the depth distance and take it as one dimension of the preintegration table. For example, the preintegration table is denoted by $\text{PreIntTable}[r][i][j], 0 \leq r \leq q - 1, 0 \leq i, j \leq n - 1$, where $n$ is the scale of the transfer functions (usually equal to 256) and $q$ is the quantization scale of the depth distance. Note that the preintegration table can be calculated in linear time [16].

The quantization scale of the depth distance influences the system costs directly in pre-computing the preintegration table. Big $q$ will inevitably result in long computing time and much traffic. Thus, we need to reduce $q$ as many as possible while preserving the rendering quality. Note that the depth distance takes value in $[0, 1]$ after the depths of the near and far view planes are normalized to 0 and 1, respectively. Because the depth distance can vary freely and sometimes becomes very small in adaptive sampling, we need to adopt a logarithmic mapping instead of a linear mapping. A 2-based logarithmic mapping was used previously for 3D preintegration in [23]. It is further found that the depth distance is seldom greater than 1/8 and located in a little interval near 0 in the most cases. Therefore, we divide $[0, 1]$ into two intervals: In the interval containing 1 we use the 2-based logarithmic, while in the other interval we use the 10/9-based logarithmic, which varies much more laggardly and accordingly is able to provide better precision. In detail, our mixed logarithmic quantization scheme is given as follows:

$$
    r = \begin{cases} 
    \left\lfloor \log_u D \right\rfloor, & \text{if } D > u^{-s}, \\
    q - 1, & \text{if } D \leq u^{-s} p^{r+1-6}, \\
    s + \left\lfloor \log_p D u^{r} \right\rfloor, & \text{otherwise,}
    \end{cases} \tag{6}
$$

where $u=2, s=3, p=10/9$. As a result, we can decrease the quantization scale $q$ to 64 without loss of the rendering quality. In accordance, when computing the preintegration table, we use the following depth distances:

$$
    D_r = \begin{cases} 
    \left( u^{-r} + u^{-r-1} \right)/2, & \text{if } r < s, \\
    u^{-s} p^{r+s} / 2, & \text{if } r = q - 1, \\
    u^{-s} (p^{r+s} + p^{r+s-1}) / 2, & \text{otherwise.}
    \end{cases} \tag{7}
$$

When computing the preintegration table, we use the following parallelization strategy: The three-dimensional preintegration table can be regarded as $q$ two-dimensional sub-tables, i.e. $(\text{PreIntTable}[r][i][j], 0 \leq r \leq q - 1, 0 \leq i, j \leq n - 1, r = 0, 1, \ldots, q-1$. We assign the $q$ computing jobs to the cores averagely, where a job is to compute one sub-table and the assigned sub-tables on each core are asked to be successive. And then, the jobs are performed on each core, and sequentially the resulting sub-tables are broadcasted to all other cores using the MPI_Bcast function. On each core, since the sub-tables occupy continuous memory, one time of broadcasting is enough and moreover no auxiliary memory is needed in either sending or receiving the sub-tables. It is worth to point out that we have also tried a parallelization strategy using the MPI_Alltoallv function but obtained a poor performance result.

Note that the computation of the preintegration table is a preprocessing step, which will be performed only when a user modifies the transfer functions interactively. The preintegration table will keep constant when all
other parameters are changed, such as the rendering window, viewing angles, datasets and time steps. Thus, we usually do not take it into account when counting the volume rendering time. After decreasing the quantization scale of depth distance and parallelizing the computation of the preintegration table, the computing time of the preintegration table can be reduced to 0.5 second or so. It is acceptable for the user in interactive design of the transfer functions.

8 Experiments

Using the proposed adaptive sampling based rendering algorithm, we experiment with the output data of the 3D laser-plasma interaction hydrodynamic simulation, where the grid size is $512 \times 512 \times 1024$, the cell number is about 0.27 billion and there are 150 time steps. As shown in Fig. 7 (Top), the resulting image for the adaptive rendering algorithm can clearly reveal inner physical phenomena in the data field. Based on the uniform sampling method, a rendered image of near quality is also shown in Fig. 7 (Bottom), where a much higher sampling density has to be used.

![Fig. 7. Rendered images with the adaptive (Top) and uniform (Bottom) sampling methods.](image)

Figure 8 shows the performance of the proposed parallel rendering algorithm in different parallel scales. In the experiment, the number of the computing nodes varies from 1 to 20, and eight cores are used in each node. With the increase of the computing nodes, the minimal and maximal sampling times on the cores decrease quickly, and
accordingly the volume rendering time decreases considerably, as shown in Fig. 8 (Top). Comparing the proposed rendering algorithm with the traditional rendering algorithm based on uniform sampling, one can see from Fig. 8 (Bottom) that the maximal sampling times of the former are obviously less though the minimal sampling times are very approximative. Since the maximal sampling time over cores influences the performance of volume rendering directly, the comparison means that the proposed rendering algorithm has a better performance.

![Fig. 8. Computing times in parallel rendering.](image)

![Fig. 9. Rendering times for two volume rendering methods.](image)

In fact, we have checked the rendering times for both the traditional uniform volume rendering method and our adaptive volume rendering method, as shown in Fig. 9. Relative to the traditional rendering method, the proposed rendering method can achieve a rendering speedup ratio between 1.8–4.4.
In addition, we have also compared the rendering times when using our entropy-based static load balancing (LB) scheme and the traditional LB scheme, e.g. load balancing with contiguous blocks together. Relative to the traditional scheme, our load balancing scheme can achieve a rendering speedup ratio between 1.5–4.9, as shown in Fig. 10. In our scheme, the time for assigning data blocks among cores is always very tiny and thus can be neglected in the rendering process.

![Fig. 10. Rendering times for two load balancing schemes.](image)

9 Conclusions

Aiming at large-scale time-varying datasets output in scientific simulations, we propose in this paper a parallel preintegration volume rendering algorithm based on adaptive sampling. The algorithm samples a data field adaptively according to its extremal points on the rays, thus we can exactly depict its variation with less sampling points. In the parallel rendering framework, we give an effective method to manage and sort the unstructured sampling points obtained by adaptive sampling. We finally composite the sampling points along each ray by the preintegration technique. In order to improve load balance, we further design a static load balancing scheme based on information entropy. In a preprocessing step, we reduce the time for computing the preintegration table by improving the quantization scheme of depth distance and parallelizing the computation. The experiments demonstrate that the proposed volume rendering algorithm can be used to clearly and quickly visualize the physical features contained in large-scale scientific data, especially when these features are distributed spatially unevenly.

References


Dr. Huawei Wang is currently an Associate Professor of Institute of Applied Physics and Computational Mathematics, Beijing. He worked as a research associate at City University of Hong Kong from 2007 to 2009 as well as Hong Kong University of Science and Technology from 2006 to 2007. He was a post-doctor at Tsinghua University, Beijing from 2004 to 2006. He received his PhD and MEng degrees in Computer Science and Technology from Tsinghua University in July 2004, and his BSc in Applied Mathematics and BEng in Computer Science and Technology also from Tsinghua University in July 1998. His research
interests include scientific visualization, computer graphics, computer aided geometric design, wavelet analysis, etc.

**Zhiwei Ai** is an Associate Professor of Institute of Applied Physics and Computational Mathematics, Beijing. He obtained his B.S in Automatic Control from Xian University of Technology (1996), and his M.S in Computational Mathematics from Graduate School of CAEP (2005), both in China. His research focuses on the scientific visualization algorithms design, and software development for large-scale scientific and engineering computation, including parallel and scalable visualization algorithms, large-scale data visualization for real applications.

**Yi Cao** is an Associate Professor of Institute of Applied Physics and Computational Mathematics, Beijing. He obtained his B.S in Computer Science from Lianzhou University (1998), and his M.S in Computational Mathematics from Graduate School of CAEP (2007), both in China. His research focuses on the scientific visualization algorithms design and software development for large-scale scientific and engineering computation, including hardware-accelerated algorithms, parallel and scalable visualization algorithms, large-scale data visualization for real applications.

**Li Xiao** is a Professor of Institute of Applied Physics and Computational Mathematics, Beijing. She obtained her B.S in Industrial Electric Automation from Dalian Railway Institute (1993), and her M.S in Computer Application from Harbin Engineering University (1996), both in China. Her research focuses on the scientific visualization software design and development for large-scale scientific and engineering computation, including parallel visualization pipeline, large-scale visualization framework for real applications.