Adaptive and Scalable Load Balancing Scheme for Sort-Last Parallel Volume Rendering on GPU Clusters

Won-Jong Lee
Dept. of Computer Science
Yonsei University, Korea

Vason P. Srinivasan
Data Flux Systems Inc., U.S.A

Tack-Don Han
Dept. of Computer Science
Yonsei University, Korea

Abstract
Sort-last parallel rendering using a cluster of GPUs has been widely used as an efficient method for visualizing large-scale volume datasets. The performance of this method is constrained by load balancing when data parallelism is included. In previous works static partitioning could lead to load imbalance when only task level parallelism is included.

In this paper, we present a load balancing scheme that adapts to the characteristic of volume dataset when data parallelism is also employed. We effectively combine the hierarchical data structures (octree and BSP tree) in order to skip empty regions and distribute workload to corresponding rendering nodes. Moreover, we also exploit a 3D clustering method to determine visibility order and save the AGP bandwidths on each rendering node. Experimental results show that our scheme can achieve significant performance gains compared with traditional static load distribution schemes.

Keywords: load balancing, octree, BSP tree, hierarchical visualization, parallel rendering, volume rendering

1 Introduction
Recent demands for high-resolution and high-quality imagery lead to increasing sizes of volumetric datasets that can be tens of gigabytes. Sort-last parallel visualization is used as a solution to scale the available memory and performance by the number of nodes in a cluster. These schemes have been widely researched with ray-casting based methods [6,12,14] and 3D texture based methods [7,9,13].

In analyzing the performance of the sort-last [8] parallel volume rendering, one of the major factors must be considered is load balancing. The speedup depends on the parallelism and any sequential part will eventually limit the speedup of parallel algorithms. Thus, well-designed load balancing scheme is indispensable in order to maximize parallelism and minimize synchronizing costs.

3D texture-based methods utilizing clusters are receiving increased attention due to advances in graphic processing unit (GPU), increased memory in GPU, and low-cost network equipments. Most of the previous research efforts [7,9,13] ignored load balancing issues, because they usually sub-divide a volume into same sized sub-volumes and distribute them statically. Although this static partitioning leads to a trivial self-balance in terms of task parallelism, this leads to load imbalance when data parallelism is added. This problem is not solved even if techniques dealing with skipping empty space [5] are utilized in each renderer. However it can make a single renderer faster, it can not contribute to balanced rendering on clusters.

We propose a load balancing scheme that effectively adapts to the characteristic of volume dataset by utilizing the combination of the hierarchical data structures such as octree and orthogonal BSP trees. Moreover, these hierarchies might be inexpensively adapted with regards to the rasterization parameter (such as changing the transfer function) during rendering time. Our scheme consists of the following steps. First, using octree the original volume data is sub-divided and only relevant regions are represented. Next, these relevant sub-volumes are distinguished and grouped hierarchically by orthogonal BSP tree. These groups of sub-volumes are assigned to leaf nodes of BSP tree and transmitted with its hierarchy to each rendering node. Finally, 3D clustering method [1] is utilized for regenerating hierarchies and to determine visibility order for valid rendering and compositing. Our proposed scheme (octree plus BSP tree) was tested on VCG cluster [10] of Kyoto University. We show that significant gains in overall parallelism and rendering performance are achieved in compared with the traditional method and the method using only octree.

The rest of this paper is organized as follows. Section 2 presents traditional static load distribution scheme and its limitations. Section 3 describes the proposed adaptive load balancing scheme. Section 4 provides experimental results. Finally, we conclude and discuss future plan in Section 5.

2 Static Load Balancing
In this section, we describe two traditional load balancing methods: static partitioning and parallel octree. Static partitioning is usually used due to the simplicity of data distribution strategy for sort-last approach [7,9,13]. In preprocessing phase the input dataset is partitioned into same sized sub-volumes depending on the number of nodes. The rendering process of each node can generate output that overlaps each other in image space, because the partitioning was performed in object space. Final image can be generated in compositing stage. This method is not efficient because it does not consider empty space, although it can serve as a cheap preprocessing and a self-balance feature.

One of the alternative methods is to employ parallelized octree. Parallel octree is used in several research projects for large data simulation and visualization [2,4]. Initially, the global octree is created in order to skip empty bricks efficiently. Next, the global octree is split down. Depending on the workload and the available rendering nodes, a subtree can be assigned to a new local octree and keeping the parent/children information. Finally, rendering and compositing are performed in each node, after the relevant sub-volumes and local octrees are distributed to the corresponding rendering nodes. Each local octree brick is rendered separately in a given traversal. Utilizing empty-space skipping allows us to accelerate rendering in
due to sub-division since it is still based on the static method.

Unfortunately, this method can not solve the imbalance formation about the tagged sub-volumes is encoded by the function axis between the created octree sub-volumes. Assume that the in-
volumes and convert them into an orthogonal BSP tree, in which
number of vertex.
The optimal coverage of sub-volumes would be achieved by assigning a sub-volume to each relevant elementary
space becomes the number of rendering nodes. In the created BSP
tree, each leaf node contains its own signature list and the positional
min/max values of bounding cube to enclose the sub-volumes in its
own space as shown in Fig. 1b. After creating this global BSP tree,
the partitioned sub-volumes, and packed global octree and BSP tree
are transmitted to the each rendering node.

To render, we need a new hierarchy describing the relationship
between the octree sub-volumes represented in a leaf node of global
BSP tree. This is because the previous relationship of child/parent
in octree hierarchy is not maintained any more. Moreover, the sub-
volumes should be rendered in visibility order. Therefore, we em-
ploy the 3D clustering algorithm proposed by Berger et al [1]. This
is an efficient and fast method used in visualizing the AMR (adap-
tive mesh refinement) dataset [3]. First, exterior zero-entries in sig-
nature lists are detected and pruned off in order to place a best-fitted
bounding box around the tagged cells. Second, any interior zero
entry in these lists indicates a potential splitting index at which the
given region is subdivided into two smaller sub-regions. Third, if
all signatures are nonzero, the, the Laplacian second derivative,
\[
\Delta_{yz}(i) = S_{yz}(i+1) - 2S_{yz}(i) + S_{yz}(i-1)
\]
(3)
similarly, for \(\Delta_{xz}(i), \Delta_{xy}(i)\) of each signature list of expression (2) for a given axis is
computed and the biggest refraction point is taken as the splitting
plane. This procedure is repeated recursively on the newly created
sub-regions until one of the following halting criteria is satisfied.
First, the sub-region exceeds some efficiency ratio, i.e., the ratio
of the number of its sub-volumes to its total number of grid cells is
greater than a pre-selected value. Second, the further subdivision
of the region would result in grid dimensions smaller than some
minimal extension. (See Fig. 1c)

We render the clustered sub-volumes by utilizing 3D texture
approach. Each part of original volume data described by global BSP
tree is loaded to each GPU as a 3D texture. To avoid artifacts caused
by discontinuities between adjacent grids during the rendering via
3D textures, it shares one row of data samples at its common bound-
ary faces. The sharing occurs only on the facets of 3D textures be-
tween the distributed rendering nodes. There is no sharing between

3 Proposed Adaptive Load Balancing Scheme

The proposed scheme is composed of four steps: generating the
cutree, generating BSP tree, 3D clustering, and rendering and com-
posing. In this section, we describe it in detail. The first step is
to generate the octree structure that contains the spatial information
of each sub-volume in a hierarchy. If the size of the volume data is
not a power of two, octree starts with the superblock, representing
the whole dataset, extended to next bigger power of two. This hier-
archy stores the min/max scalar data values within a node. In com-
bination with the transfer function, these min/max values allow us
to skip completely transparent nodes. Therefore, only subbranches
with leaf nodes that cover relevant regions of the data volume can
be inserted into the hierarchy (see Fig. 1a).

The size of the sub-volume is a crucial factor that affects ren-
dering performance. The optimal coverage of sub-volumes would
be achieved by assigning a sub-volume to each relevant elementary
cell of the dual voxel grid. Obviously, this would result in an enor-
mous number of texture and polygon coordinates to be computed
and specified since every sub-volume has to be intersected with the
slices to be rendered. This can cause an overhead in the vertex pro-
cessor of GPU and an increased AGP bandwidth. The sub-volume
size of 16x16x16 is selected as the optimal size, because it gives the
best results to us for the rendering performance. Moreover, 3D clus-
tering method [1] used in each rendering node can reduce the
number of vertex.

To partition the octree, we group a set of octree sub-

\[
F : [1,...,n_s] \times [1,...,n_y] \times [1,...,n_z] \mapsto \{0,1\}
\]  
(1)

where \(n_s\), \(n_y\), and \(n_z\) are the number of raw or column for the sub-
volumes in each axis, \(F(x,y,z) = 1\) if the sub-volume with index
\(x,y,z\) is not empty. For each slab perpendicular to the \(xy, yz,\) and
\(zx\) planes, the number of sub-volumes is computed and stored in
signature lists. First, we start for the longest axis. For example,
the entry for slab number \(x\) parallel to the \(yz\) plane is given by

\[
S_{yz}(x) = \sum_{y=1}^{n_y} \sum_{z=1}^{n_z} F(x,y,z)
\]
(2)

and, similarly, for the two other orientations. We split a node for
current axis as shown in Fig. 2. Once the position of splitting plane
is determined, the sub-volumes are partitioned into front and back
spaces of splitting plane, that is, two children. This procedure is
repeated recursively for a next axis until the number of partitioned

1: Choose an axis for splitting
2: Compute three signature lists \(S\) for current axis
3: Initialize two indexes and two summation variables
   \(\text{lower} = \text{zero}, \text{upper} = \text{the length of } S\) for current axis
   \(\text{leftsum} = S(\text{lower}), \text{rightsum} = S(\text{right})\)
4: \textbf{while} \(\text{lower}\) is less upper \textbf{do}
5: \textbf{if} \(\text{leftsum}\) is greater than \(\text{rightsum}\) \textbf{do}
6: \textbf{increase lower} and \(\text{leftsum} = \text{leftsum} + S(\text{lower})\)
7: \textbf{else}
8: \textbf{decrease upper} and \(\text{rightsum} = \text{rightsum} + S(\text{right})\)
9: \textbf{end if}
10: \textbf{end while}
11: Select \(\text{lower}\) or \(\text{upper}\) as a index for splitting plane

Figure 2: Pseudocode of splitting strategy.
The rendering order is easily determined, because all bricks are based on the BSP tree method. The traversal of local BSP tree is performed in the following way. At each local BSP node, the viewpoint is compared to the position of the splitting plane and the two child nodes are visited using in-order traversal. If the node is a leaf, the associated brick is rendered slice by slice. Similarly, by using global octree the valid compositing order for each rendered sub-image can be efficiently determined in arbitrary viewpoints. Our scheme could be adapted in the case of changing the transfer function during rendering time. By communicating only the appropriate parts of sub-volumes between the rendering nodes we can keep the load balance without full retransmission.

4 Experimental Results

To evaluate the performance of the proposed scheme, we implemented a parallel rendering program using C++ and OpenGL. Volume rendering was performed with per-fragment lighting and post-shading by NVIDIA register combiners. MPI/PM is used for all communication between nodes.

The VG-Cluster system of Kyoto University [10] was used for rendering and evaluation. This system consists of 9-PC nodes in a cluster using image-compositing hardware [11] manufactured by Mitsubishi Precision, Co. Ltd. Each PC node has an Intel Pentium 4 processor running at 2.4 GHz, 1024MB DDR SDRAM, one NVIDIA GeForce FX 5950 ULTRA GPU with 256MB Video RAM. The cluster employs a Gigabit Ethernet for the inter PC communications. We use three datasets, “Lobster” (301x324x356, courtesy of SUNY Stony Brook), “Leg” (341x341x93, courtesy of German Federal Institution), and Visible Human Male (VHM, 430x240x256, courtesy of National Library of Medicine). In this work, we assume that the datasets fit into the overall graphics memory of GPUs in the cluster. All rendered images are shown in Fig. 3. The performance of the proposed scheme based on octree and BSP tree (OCT+BSP) is compared with the case of conventional method (STANDARD) used in [7, 9, 13] and the case of using only octree (OCT) used in [2, 4].

We evaluate the performance of the proposed load balancing scheme by measuring the number of sub-volumes distributed to each rendering node. Fig. 4 shows a comparison of the number of sub-volumes distributed for OCT and OCT+BSP on the 8 rendering node cluster. To test for a larger set of rendering node arrangements, we also measured it for a cluster containing 16 rendering nodes by mean rendering was performed with per-fragment lighting and post-shading by NVIDIA register combiners. MPI/PM is used for all communication between nodes.

The VG-Cluster system of Kyoto University [10] was used for rendering and evaluation. This system consists of 9-PC nodes in a cluster using image-compositing hardware [11] manufactured by Mitsubishi Precision, Co. Ltd. Each PC node has an Intel Pentium 4 processor running at 2.4 GHz, 1024MB DDR SDRAM, one NVIDIA GeForce FX 5950 ULTRA GPU with 256MB Video RAM. The cluster employs a Gigabit Ethernet for the inter PC communications. We use three datasets, “Lobster” (301x324x356, courtesy of SUNY Stony Brook), “Leg” (341x341x93, courtesy of German Federal Institution), and Visible Human Male (VHM, 430x240x256, courtesy of National Library of Medicine). In this work, we assume that the datasets fit into the overall graphics memory of GPUs in the cluster. All rendered images are shown in Fig. 3. The performance of the proposed scheme based on octree and BSP tree (OCT+BSP) is compared with the case of conventional method (STANDARD) used in [7, 9, 13] and the case of using only octree (OCT) used in [2, 4].

We evaluate the performance of the proposed load balancing scheme by measuring the number of sub-volumes distributed to each rendering node. Fig. 4 shows a comparison of the number of sub-volumes distributed for OCT and OCT+BSP on the 8 rendering node cluster. To test for a larger set of rendering node arrangements, we also measured it for a cluster containing 16 rendering nodes by counting the sub-volumes to be sent to each virtual rendering node on the display node.

As the figure shows, OCT exhibits the tendency of imbalanced load distribution. The maximum difference between the maximum and the minimum sent is 509 (on 8 rendering nodes). In the case of Leg, specially, half of the rendering nodes are idle. This means that static sub-division method like OCT can cause inefficient utilization in spite of its ability to skip empty space. In contrast, OCT+BSP keeps the characteristics of balanced load distribution in all cases. Therefore, we are assured that the rendering on each GPU can be finished at the same time approximately.

![Figure 3: Rendered images by the proposed scheme.](image)

![Figure 4: Comparison of the number of sub-volumes distributed to 8 and 16 rendering nodes.](image)

To evaluate how well the proposed scheme can be adapted to the sort-last parallel volume rendering, we measured the substantial frame rates. The comparison of the frame rates for rendering of the test volume datasets to 512x512 sized windows with differing number of rendering nodes are shown in Fig. 5. Changing the number of rendering nodes from 1 to 8 increases the frame rates for most of the case. As we mentioned, we utilized the 3D clustering algorithm can reduce the number of vertex in each rendering node. Therefore, OCT+BSP reveals the better performance in case of one single node also. Specially, the case of generating the large number of sub-volumes (VHM), OCT+BSP increase performance by more than a factor of two because the reduction rate is relatively high.

Although STANDARD shows the increasing rates almost linearly with naive feature of balanced load distribution, its rendering performance is worse than others due to its data-unaware feature. The OCT and OCT+BSP approaches show better performance than STANDARD, because of their ability to render only relevant region. OCT+BSP outperforms OCT in all case since it makes use of features of well-balanced load distribution. OCT records 31-37 frames and OCT+BSP allows 35-40 frames. In case of Leg, specially, OCT does not utilize the advantage of increasing number of rendering nodes. Frame rates seldom change because it distributes workload to only half of the rendering nodes.

The OCT and OCT+BSP approaches do not show linearity like STANDARD at the large window resolution due to the following reason. Although a hardware image compositor is utilized, it has some physical performance limitation [10, 11]. Besides, needing time to read each rendered image from the frame buffer, additional time is needed to send each image across the PCI bus to the compositing hardware for each frame. Therefore, performance loss can not be avoided in compositing stage in case of high rendering frame rates. We measured frame rates without compositing (See Fig. 6). achieved 100-300 frames in the case of OCT+BSP and 55-150 frames in the case of OCT. As shown in the graph, our load balancing scheme itself shows almost linearity. We have concluded that a more intelligent compositor or one that employs a faster bus like PCI Express can make rendering scalable linearly by hiding the compositing time behind the rendering time.

Fig. 7 shows the comparison of actual rendering time versus cluster size for one frame at 512x512 sized window. The rendering time is decomposed into the fastest finished time between rendering nodes and the barrier time required by software synchronization (MPI Barrier). The time for compositing is ignored because it is always the same under the same window size. We simulated cluster sizes 16 to 64 by doing multiple renderings on a...
single node and the rendering time was measured for each case. Barrier time is calculated as the difference between the fastest and latest rendering times. The barrier time is extremely short in the case of STANDARD, due to naive self-balanced feature, but rendering takes longer. Although the rendering time is shorter than OCT+BSP, OCT spends longer time for synchronizing (57%-83% of total time), because of its unbalanced workload distribution. The reduced barrier time (occupies 29%-48% of total time) of OCT+BSP approach is the reason that it outperforms others.

5 CONCLUSION AND FUTURE WORK

We have developed an adaptive load balancing scheme for texture-based parallel volume rendering on GPU clusters. By adapting octree and orthogonal BSP tree, only relevant region of the original volume is effectively detected and decomposed evenly. The 3D clustering method regenerates hierarchies and determines rendering order correctly. By utilizing both empty space skipping and adaptive load distribution, we could achieve significant performance gains. Increasing parallelism and decreasing synchronization costs contributed to better performance compared to traditional static load distribution schemes.

We are looking at other ways to extend our scheme. We only considered static datasets in this work. In the case of time-variant datasets, the shapes are changed at every time step. The proposed scheme could be used if each periodic dataset is differently encoded by octree and BSP tree in the preprocessing steps. Besides, early ray termination could be integrated with empty space skipping in our scheme by utilizing the recent GPU features.

6 ACKNOWLEDGEMENTS

We deeply appreciate the valuable suggestion from Shigeru Muraki (AIST). We are grateful to Koji Koyama and Jorji Nonake (Kyoto University) for their helpful assistance in using their cluster system. We wish to thanks Issi Fujishiro (Tohoku University) for his suggestion on parallel volume rendering. We would like to thanks Xuezen Liu and Yasuo Katoana (Mitsubishi Precision) for their helpful suggestion on compositing hardware and PC clustering. Thanks also to Woo-Chan Park (Sejong University) and all the staff of the Media System Laboratory at Yonsei University for their valuable discussion and comments. We wish to thanks Sung-Bong Yang for his departmental support.

REFERENCES