Efficient Pipelining Parallel Methods for Image Compositing in Sort-Last Rendering

Wei Fang[1,2], Guangzhong Sun[1], Peng Zheng[2], Tiening He[2], Guoliang Chen[1]

[1] School of Computer Science and Technology, University of Science and Technology of China, Hefei, China.
`fangwei@mail.ustc.edu.cn`,{`gzsun`,`glchen`}`@ustc.edu.cn`
[2] Institute of Computer Application, China Academy of Engineering Physics, Mianyang, China.
{`eliza_zheng`,`htn2005`}`@126.com`

Abstract. It is well known that image compositing is the bottleneck in Sort-Last rendering. Many methods have been developed to reduce the compositing time. In this paper, we present a series of pipeline methods for image compositing. Our new pipeline methods based on Direct Send and Binary Swap. However, unlike these methods, our methods overlap the rendering time of different frames to achieve high fps(frames per second) in final display. We analyze the theoretical performance of our methods and take intensive experiments using real data. The results show that our new methods are able to achieve interactive frame rates and scale well with both the size of nodes and screen resolution.

Keywords: Pipelining, parallel methods, image compositing

1 Introduction

Polygon rendering and volume rendering are important in scientific visualization. As the size of data produced by large-scale computation and simulation grow larger and larger, a natural solution for effective visualization of very large dataset is utilizing a supercomputer or a PC-cluster to parallelism rendering work. Molnar et al. [1] described a framework for parallel rendering. They proposed three types of parallel rendering algorithms: sort first, sort-middle and sort-last, depending on where the sort and redistribution of data occurs.

Sort-last algorithm is widely used in parallel rendering because of its scalability and simple task decomposition for achieving load balance. However, the image compositing stage in sort-last could be very expensive because of large amount of messages exchanged. Therefore, image compositing could become a bottleneck that affects the efficiency of the sort-last parallel rendering.

A lot of methods have been developed to composite image for sort-last rendering. Direct Send [2], Binary Tree [3], Binary Swap [4], and parallel pipeline [5] [6] are already been proved to be efficient ways for image compositing. But all the methods mentioned above have some disadvantages in communication. In this paper, we construct a tree-based pipeline system, using some additional nodes named transfer node to transfer/composite pixels, and implement Direct Send, Binary Swap, and Hybrid Partition methods on this pipeline system. Intensive experiments have been done to test these algorithms and we found that our new methods can greatly reduce the number of pixels communicated among processors and increase the frame rates of final display, especially in low bandwidth, high latency networking systems.

This paper is organized as follows: related works are introduced in section 2. We give some theoretical analysis for Direct Send and Binary Swap methods in section 3. Our new pipeline algorithms are described in detail in section 4. Section 5 is dedicated to implementation details and experimental results. Finally, concluding remarks are discussed in section 6.

2 Related Works

Image compositing is the key stage of sort-last rendering method. Most performance lost happens in this stage. So a number of parallel image compositing method have been implemented, both in software and hardware.

Software-based methods for image compositing have been proposed in the literature, and can be applied either to polygon or volume rendering. The simplest way is sending pixels directly to the node who are responsible for blending them. This is called Direct Send [2]. The disadvantage of this method is that many nodes would send pixels to the same node at the same time, which would cause link contention and hurt the performance greatly. Ma [4] proposed Binary Swap method, which in most cases is better than Direct Send. Many work have been done to improve Direct Send and Binary Swap method, such as scheduled linear image compositing [8], multiple bounding rectangle [5], run-length encoding [9], interleaved splitting [9], 2-3 swap [10] and so on. Garcia et.al [7] devised a hybrid image- and object- space partitioning algorithm to perform parallel rendering. They divide the rendering nodes into several groups, and perform object space partitioning among these groups. Within each group, they perform image space partitioning. This strategy can efficiently decrease the pixels communicated among nodes. There are also some pipeline work to reduce the time of image compositing [5] [6]. All of them implement pipeline in the process of rendering one frame.

Unlike these methods, our new pipeline algorithm implement pipeline in the process of rendering several frames. Contiguous frames flow in the pipeline at the same time. The way we used to improve the fps in final display is overlapping the time of compositing different frames, not reducing the time of compositing one frame. This is the major difference between our methods and the other pipeline methods. We will describe it in detail in section 4.

3 Theoretical Analysis

We give some theoretical analysis for Direct Send and Binary Swap methods, which are the most representative and commonly used.

In this paper, we assume that every frame should be displayed in display node. And for the current generation of network device, the network usually supports full-duplex send and receive operations. So, sending and receiving can be overlapped. Some notations used in this paper list below.

- n : the number of rendering nodes.
- $T(M)$: the reciprocal of fps, using method M.
- $T_{rendering}$: the time of rendering one image and reading back color buffer(and depth buffer in polygon rendering) from GPU by rendering nodes.
- T_{comm} : the communication time between rendering nodes.
- $T_{blending}$: the time of blending pixels.
- P_{xy} : the size of pixels need to be transferred, where x is the screen width and y is the screen height.
- $T_{display}$: the time of displaying P_{xy} pixels in the display node.
- L : the sum start-up time of a communication channel and latency of sending and receiving. (Assume sending and receiving have the same latency)
- T_c : the data transmission time per byte. To simplify our model, we assume T_c is constant between one node and any other node.
- T_b : the blending time per byte. Usually T_b is much smaller than T_c because blending two images is much faster than transfer one image, and T_b can also be speeded up by sse instructions provided by CPU.

3.1 Direct Send

The Direct Send method is simple: the screen is divided into n fractions, and every rendering node deal with one of them. In compositing stage, each node sends pixels direct to the node who is responsible for blending that portion. At the end of compositing stage, every rendering node has $\frac{1}{n}$ part of final image.

Let T_{rn} be the time of rendering nodes generate one frame, and let T_{dn} be the time of display node display one frame. Obviously, there are some overlap time between T_{rn} and T_{dn}: when rendering nodes sending pixels to display node, display node is receiving these pixels. Let $T_{rn\_dn\_comm}$ be this overlap time. Thus

$$T(DS) = T_{rn} + T_{dn} - T_{rn\_dn\_comm} \tag{1}$$

where

$$\begin{aligned} T_{rn} &= T_{rendering} + T_{comm} + T_{blending} + T_{rn\_dn\_comm} & (2)\\ &= T_{rendering} + (n-1)(\frac{1}{n}P_{xy}T_c + L) + \frac{n-1}{n}P_{xy}T_b + \frac{1}{n}P_{xy}T_c^{'} + L & (3)\end{aligned}$$

and

$$T_{dn} = T_{rn\_dn\_comm} + T_{display} \tag{4}$$

here $T_c^{'} \neq T_c$, because of link contention. In this condition, we assume every node has the same transferring rate. Thus $T_c^{'} \approx nT_c$, finally,

$$T(DS) \approx T_{rendering} + T_{display} + \frac{2n-1}{n}P_{xy}T_c + \frac{n-1}{n}P_{xy}T_b + nL \tag{5}$$

3.2 Binary Swap

In Binary Swap algorithm, compositing one frame needs $log_2 n$ stages. In the ith stage, every rendering node needs to send and receive $\frac{1}{2^i}P_{xy}$ pixels, and blend $\frac{1}{2^i}P_{xy}$ pixels. Thus

$$\begin{aligned} T_{rn} &= T_{redering} + T_{comm} + T_{blending} + T_{rn\_dn\_comm} && (6) \\ &= T_{rendering} + T_{rn\_dn\_comm} + \sum_{s=1}^{\log_2 n} (\frac{P_{xy}T_c}{2^s} + \frac{P_{xy}T_b}{2^s} + L) && (7) \\ &\approx T_{rendering} + T_{rn\_dn\_comm} + \frac{n-1}{n} P_{xy}(T_c + T_b) + L\log_2 n && (8) \end{aligned}$$

$T_{rn\_dn\_comm}$ and T_{dn} are the same as in Direct Send method, so

$$\begin{aligned} T(BS) &= T_{rn} + T_{dn} - T_{rn\_dn\_comm} && (9) \\ &\approx T_{rendering} + T_{display} + \frac{2n-1}{n} P_{xy}T_c + \frac{n-1}{n} P_{xy}T_b + L\log_2 n && (10) \end{aligned}$$

We can see that Direct Send and Binary Swap have the same complexity of communication. But reported by many researches, Binary swap has better runtime performance. Two reasons can explain this, the one is that Direct Send method needs globe network operations, which would become a disaster when the number of processors increase; the other is that Binary Swap has less network function calls.

4 Pipeline Methods

The key idea of our methods is employing additional nodes(named transfer node) to construct tree-based pipeline, which could reduce the number of pixels need to be transferred among nodes. We try to totally overlap the rendering time, blending time, and the time of sending/receiving pixels. Unlike Direct Send and Binary Swap method, our methods is in an effort to improve the fps in final display, not reduce the time of render one image.

In this section we will propose three new algorithms: Pipeline based on Direct Send(PDS), Pipeline based on Binary Swap(PBS) and Hybrid Partition Pipeline(HPP).

Notice that most improvement for Direct Send and Binary Swap, such as compression, interleaving and run-level encoding, can be applied in our methods. We don't use them in this paper in order to face the worst case situation.

4.1 Pipeline based on Direct Send

The simplest method using transfer node to implement pipeline is displayed in Figure 1. We use m additional nodes as transfer nodes. The screen is divided into m fractions. Each transfer node takes charge of one fraction. Every rendering node communicates with all the transfer nodes, sending one fraction to the

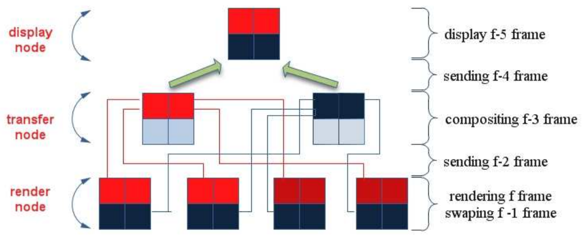

Fig. 1. Pipeline based on Direct Send, m=2, n=4

transfer node who is responsible for that portion. Transfer nodes receive pixels, then blend them, then send them to the display node.

While rendering the fth frame, the rendering node is sending the $(f-1)$th frame to the transfer node. At the same time, the transfer node is receiving the $(f-1)$th frame from all the rendering nodes, blending the $(f-2)$th frame, and sending the $(f-3)$th frame to display node.

PDS algorithm overlap the rendering time, blending time and communication time. In every step, rendering node sends P_{xy} pixels, transfer node receives $\frac{n}{m}P_{xy}$ pixels and sends $\frac{1}{m}P_{xy}$ pixels, display processor receives P_{xy} pixels. So

$$T(PDS) = \max\{T_{rn}, T_{tn}, T_{dn}\} \tag{11}$$

In rendering node, we overlap rendering time and communication time.

$$T_{rn} = \max\{\ T_{rendering}, P_{xy}T_c + (n-1)L\} \tag{12}$$

transfer node needs to blends $\frac{1}{m}P_{xy}$ pixels $(n-1)$ times, thus

$$T_{tn} = \max\{\frac{n-1}{m}P_{xy}T_b, \frac{1}{m}P_{xy}T_c^{'} + L, \frac{n}{m}P_{xy}T_c + nL\} \tag{13}$$

$$\approx \max\{\frac{n-1}{m}P_{xy}T_b, P_{xy}T_c + L, \frac{n}{m}P_{xy}T_c + nL\} \tag{14}$$

Here $T_c^{'} \approx mT_c$ as we discussed in equation (5), and

$$T_{dn} = \max\{T_{display}, P_{xy}T_c + mL\} \tag{15}$$

Usually $T_b \ll T_c$, $nL \ll P_{xy}T_c$, $T_{display} < T_{rendering}$. Thus

$$T(PDS) = \max\{T_{rendering}, \frac{n}{m}P_{xy}T_c + nL, P_{xy}T_c + \max\{n, m\}L\} \tag{16}$$

Compared with equation (5) and (10), equation (16) is a better result. When $T_{rendering} < P_{xy}T_c$ and $m \geq n$, $T(PDS)$ is bounded by $P_{xy}T_c + mL$. This is the upper bound of the model. But we could hardly get this performance using this method in real world, because the link contention would be very heavy.

4.2 Pipeline based on Binary Swap

Now we use $n-2$ additional transfer nodes to build a binary tree to implement pipeline. In this hierarchical connection architecture, all the leaves are rendering nodes. We pair up these rendering nodes, each takes charge of rendering different half of screen. Every rendering node first swaps half of screen pixels with its partner, then blend them and send pixels of its partition to transfer node, who is responsible for compositing that portion. Every transfer node receives pixels from two children, then blend them, then send them to its parent(another transfer node or display node). Figure 2 shows at a certain moment this algorithm behaves.

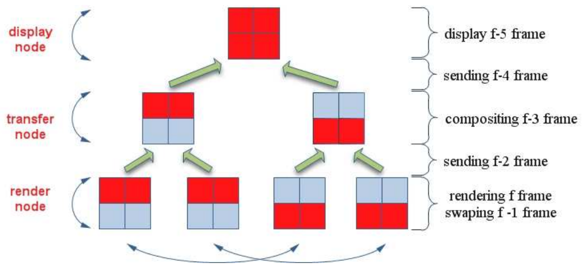

Fig. 2. Pipeline based on Binary Swap, n=4

Every rendering node need to send and receive $\frac{1}{2}P_{xy}$ pixels to/from its partner, and blend $\frac{1}{2}P_{xy}$ pixels. It also needs to send $\frac{1}{2}P_{xy}$ pixels to transfer node. Every transfer node needs to receive P_{xy} pixels, blend $\frac{1}{2}P_{xy}$ pixels and send $\frac{1}{2}P_{xy}$ pixels to its parent. Let $T_{rn\_tn\_comm}$ be the communication time between rendering node and transfer node, then

$$T_{rn} = \max\{T_{rendering}, T_{rn\_tn\_comm}, T_{comm} + T_{blending}\} \quad (17)$$

$$= \max\{T_{rendering}, \frac{1}{2}P_{xy}T_c^{'} + L, \frac{1}{2}P_{xy}T_c^{'} + L + \frac{1}{2}P_{xy}T_b\} \quad (18)$$

Here $T_c^{'} \neq T_c$, because the sending operation happens at the same time in each pair. We assume $T_c^{'} \approx 2T_c$, then

$$T_{rn} \approx \max\{T_{rendering}, P_{xy}T_c + L + \tfrac{1}{2}P_{xy}T_b\} \quad (19)$$

$$T_{tn} = \max\{P_{xy}T_c + L, \tfrac{1}{2}P_{xy}T_b, \tfrac{1}{2}P_{xy}T_c + L\} = \max\{P_{xy}T_c + L, \tfrac{1}{2}P_{xy}T_b\} \quad (20)$$

T_{dn} is the same as in equal(15). Thus

$$T(PBS) = \max\{T_{rendering}, P_{xy}T_c + L + \frac{1}{2}P_{xy}T_b\} \quad (21)$$

Using approx n transfer nodes in PDS algorithm, equation (16) shows a better performance than equation (21). But consider that PBS method has no link contention and $T_b \ll T_c$, this algorithm should have better performance than PDS method in real world. The disadvantage of this method is that there is about $\log_2 n$ frames latency in final display.

4.3 Hybrid Partition Pipeline

In this section we present hybrid partition pipeline method which employs the idea from hybrid image- and object- space partitioning method [7].

We use kn rendering nodes to render images. There are another $k(n-2)$ nodes to be transfer nodes. Rendering nodes are divided into n groups, and each group has k rendering nodes. Datasets are also divided into n groups, and every node in the same group has the same data. In every group, we divide the screen into k partitions, and each node takes charge of different part. We pair up the groups. And a rendering node's partner is the node who draws the same screen partition in the paired group. Thus each rendering node only needs to swap $\frac{1}{2k}P_{xy}$ pixels with its partner. Easy to see that, using this method the number of pixels communicated among nodes will be greatly reduced. Figure 3 shows a certain moment this algorithm behaves, where $k = 2$ and $n = 4$.

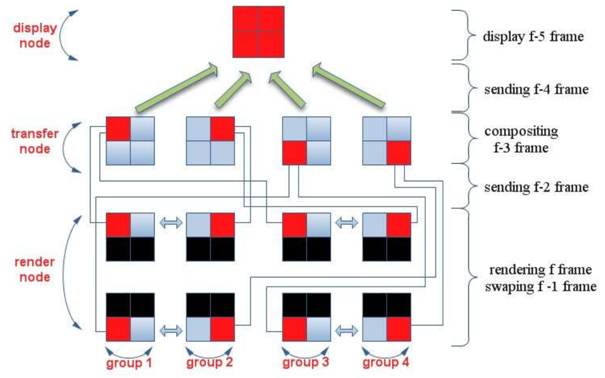

Fig. 3. hybrid partition pipeline, k=2,n=4

As we did in PBS method

$$T_{rn} = \max\{T_{rendering}, T_{rn\_tn\_comm}, T_{comm} + T_{blending}\} \tag{22}$$

$$\approx \max\{T_{rendering}, \frac{1}{k}P_{xy}T_c + L + \frac{1}{2k}P_{xy}T_b\} \tag{23}$$

$$T_{tn} = \max\{\frac{2}{k}P_{xy}T_c + L, \frac{1}{k}P_{xy}T_b, \frac{1}{k}P_{xy}T_c + L\} \tag{24}$$

$$= \max\{\frac{2}{k}P_{xy}T_c + L, \frac{1}{k}P_{xy}T_b\} \tag{25}$$

T_{dn} is the same as in equal(15), and $T_b \ll T_c$. Thus

$$T(HPP) \approx \max\{T_{rendering}, \frac{2}{k}P_{xy}T_c + L, P_{xy}T_c + 2kL\} \tag{26}$$

When k increases, the communication time of T_{rn} and T_{tn} will be reduced. Equation (26) gets better performance than PBS method. But the communication time of T_{dn} is still the same. And as we discussed in section 4.1, this is the best we can do in this model. Of course, T_{dn} can be improved using other approaches, as many researchers have done a lot of works before.

4.4 Summary

Better than PDS, PBS and HPP have no link contention, and only need a few network function calls in every step. Even when rendering nodes increase, the number of communicated pixels and network function calls are still stay constancy. This is a huge advantage in large-scale parallel rendering.

However, this is based on the assumption that we have enough nodes. If limited nodes can be used, $T_{rendering}$ may be larger than communication time, and our methods could be worse than DS or BS, because our pipeline methods need some nodes to be transfer nodes and less rendering nodes could cause render time even longer. Let T_{total} be the time of one node renders total graphic primitives. Let N be the number of nodes can be used and define function $g(N : M)$ be the number of rendering nodes using method M. Namely, $g(N : PDS, m) = N - m$, $g(N : PBS) = \frac{N+2}{2}$ and $g(N : HPP, k) = \frac{N}{2k} + 1$. If $\frac{T_{total}}{g(N:M)} \gg \frac{T_{total}}{N} + 2P_{xy}T_c$, this means we lack of nodes to render images, Direct Send or Binary Swap is a better choice than our pipeline methods in this situation.

5 Experiment Results

The application program renders polygon-data, written in C using OpenGL. We conducted our experiments on a PC-cluster of 64 nodes. Each node has dual-Xeon 3.4GHz CPU and 8GB of memory. All the nodes are connected by 1G Ethernet.

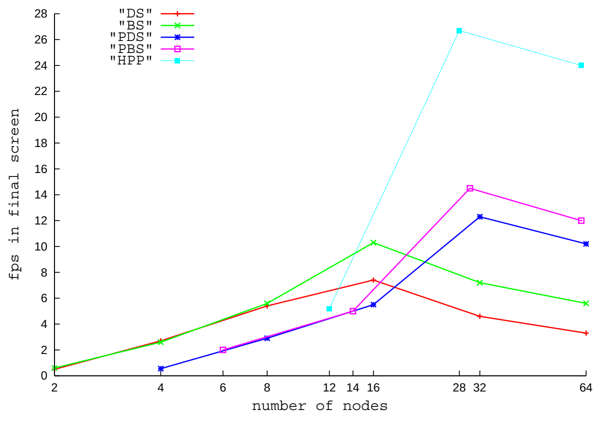

Fig. 4. The fps performance of DS,BS,PDS,PBS and HPP.

Figure 4 shows the DS, BS, PDS, PBS and HPP methods' performance of rendering 1,201,287,716 polygons with 800 × 500 screen resolution. The y axis represents the fps in final display, and the x axis represents the number of nodes, including transfer nodes in PDS, PBS and HPP methods. We adopted $n = m$ for

PDS method. When the number of rendering nodes is small, all the methods's fps are less than 6, because rendering polygons dominates the whole time. All the methods achieved the highest fps when the number of rendering nodes is 16, and the performance dropped while the number of rendering nodes increasing, because more rendering nodes means more images need to be composited. HPP method has the highest fps than other methods, and needs the most auxiliary nodes correspondingly. It achieved 26 fps using 16 rendering nodes and 12 transfer nodes.

Table 1. The scalability with the number of rendering nodes

	number of rendering nodes				
	2	4	8	16	32
PDS(1:1)	20.9	18.7	10.2	12.3	10.2
PDS(1:2)	24.6	20.1	24.5	21.4	-
PBS	-	16.3	14.5	14.5	12.0
HPP	-	27.2	26.7	24.0	-

Table 1 shows our methods' scalability with the number of rendering nodes. Each rendering node renders a small set of polygons to make sure the bottleneck is network transformation. PDS(1:a) represents $n = am$ in PDS method. The HPP have the best scalability, when the number of rendering nodes changing 4 to 16, the fps just lost 10%. PDS method's behavior was better than we expected, but was unstable both in PDS(1:1) and PDS(1:2). PBS also shows nice scalability, and have better fps performance than PDS(1:1).

Table 2. Performance predictions and measurements for different screen resolution

		PDS(1:1)		PBS		HPP	
	T(s)	M(s)	P(%)	M(s)	P(%)	M(s)	P(%)
800x500	31.2	10.2	33	12.0	38	24.0	77
1000x800	15.6	6.6	42	8.0	51	12.8	82
1280x1024	9.5	1.7	18	5.5	58	7.9	85

The inter-nodes connection of our PC-cluster has a bandwidth about 100MBs. In order to test our methods' scalability for rendering different screen size, we run our pipelining methods using as many nodes as we can to test fps performance of rendering 800×500, 1000×800 and 1024×1080 screen size. That is to say, PDS(1:1) use 32 rendering nodes and 32 transfer nodes, PBS use 32 rendering nodes and 30 transfer nodes and HPP use 16 rendering nodes and 48 transfer nodes. We calculated the upper bound of fps and measured real fps of our pipeline method for different screen size. The theoretical results($T = P_{xy}T_c$) and experimental results(M) as well as the percentage($P = T/M$) comparisons are showed in table 1. When screen size increase, the fps of PBS and HPP method are getting closer to theoretical results, that means we use the system more effective, profiting from simple topology structure and a few communication mates for each node. The fps of PDS method, on the other hand, dropped significantly when screen size increase, due to the globe networking operation.

6 Conclusion

In this paper, we proposed three tree-based pipeline algorithms for image compositing, which reduce the cost of sort-last parallel rendering. We analyzed the theoretical performance and measured the real performance of our methods, and compared with Direct Send and Binary Swap. The experiments shows that using transfer nodes to reduce communication messages and to avoid link contention, the PBS and HPP method have better behaviors than other methods. Moreover, these methods are highly scalable because when the number of rendering nodes increase, every node still have constant number of communication mates.

Acknowledgement

This work is supported by Chinese Universities Scientific Fund (CUSF), National Natural Science Foundation of China (No.60873210) and National High Technology Research and Development Program of China (No.2009AA01A134).

References

1. Molnar, S., Cox, M., Ellsworth, D., Fuchs, H.: A Sorting Classification of Parallel Rendering. IEEE Computer Graphics and Applications. 14, 23–32 (1994)
2. Neumann, U.: Communication Costs for Parallel Volume-Rendering Algorithms. IEEE Computer Graphics and Applications. 14, 49–58 (1994)
3. Shaw, C., Green, M., Schaeffer, J.: A VLSI architecture for image composition. Advances in Computer Graphics Hardware III, pp. 183–199. Springer-Verlag, New York (1991)
4. Ma, K., Painter, J., Hansen, C., Krogh, M.: Parallel Volume Rendering Using Binary-Swap Compositing. IEEE Computer Graphics and Applications. 14, 59–67 (1994)
5. Lee, T., Raghavendra, C., Nicholas, J.: Image Composition Schemes for Sort-Last Polygon Rendering on 2D Mesh Multicomputers. IEEE Transactions on Visualization and Computer Graphics. 2, 202–217. (1996)
6. Cavin, X., Mion, C., Filbois, A.: COTS Cluster-Based Sort-Last Rendering: Performance Evaluation and Pipelined Implementation. In: Proceedings of IEEE Visualization 2005, pp. 111–118. IEEE Press, New York (2005)
7. Garcia, A., Shen, H.: An Interleaved Parallel Volume Renderer with PC-clusters. In: Proceedings of Eurographics Workshop on Parallel Graphics and Visualization, pp. 51–59. Eurographics Association, Aire-la-Ville (2002)
8. Stompel, A., Ma, K., Lum, K., Ahrens, J., Patchett, J.: SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering. In: Proceedings of IEEE Symposium on Parallel and Large-Data Visualization and Graphics, pp. 33–40. EEE Press, New York (2003)
9. Takeuchi, A., Ino, F., Hagihara, K.: An improved binary-swap compositing for sort-last parallel rendering on distributed memory multiprocessors. Parallel Computing. 29, 1745–1762. Elsevier, Amsterdam (2003)
10. Yu, H., Wang, C., Ma, K.: Massively parallel volume rendering using 2-3 swap image compositing. In: Proceedings of the 2008 ACM/IEEE conference on Supercomputing, pp. 1–11. IEEE Press, New York (2008)