COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2016; **27**:385–393
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1711

SPECIAL ISSUE PAPER

Pipelining image compositing in heterogeneous networking environments

Ning Liu[1,2]*, Dengming Zhu[1], Zhaoqi Wang[1], Hong Qin[3], Jianfeng Zhan[1] and Jinzhu Gao[4]

[1] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] Stony Brook University, Stony Brook, NY, USA
[4] University of the Pacific, Stockton, CA, USA

ABSTRACT

Because of intensive inter-node communications, image compositing has always been a bottleneck in parallel visualization systems. In a heterogeneous networking environment, the variation of link bandwidth and latency adds more uncertainty to the system performance. In this paper, we present a pipelining image compositing algorithm in heterogeneous networking environments, which is able to rearrange the direction of data flow of a compositing pipeline under strict ordering constraint. We introduce a novel directional image compositing operator that specifies not only the color and α channels of the output but also the direction of data flow when performing compositing. Based on this new operator, we thoroughly study the properties of image compositing pipelines in heterogeneous environments. We develop an optimization algorithm that could find the optimal pipeline from an exponentially large searching space in polynomial time. We conducted a comprehensive evaluation on the ns-3 network simulator. Experimental results demonstrate the efficiency of our method.

KEYWORDS

pipeline; image compositing; heterogeneous networking environment; optimization algorithm

***Correspondence**

Ning Liu, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China.
E-mail: liuning01@ict.ac.cn

1. INTRODUCTION

Parallel visualization has been widely used to analyze large-scale data sets. It typically consists of three stages: data distribution, local rendering, and image compositing. In a sort-last parallel visualization system, image compositing must be carried out every time the rendering parameters change. Bentes [1] studied the overhead components involved in a parallel volume visualization system and showed that the network communication incurred the largest cost. In a heterogeneous environment, the variation of networking characteristics adds more uncertainty to the system performance. Consequently, image compositing could easily become a bottleneck of parallel visualization systems.

In the past, researchers have proposed various image compositing algorithms, such as direct send [2], binary swap [3], 2-3 swap [4], radix-k [5], and pipelining image compositing [6]. They are all designed for homogeneous networking environments. The main issue caused by the homogeneity assumption is bad load balancing among the nodes. Bad load balancing usually leads to poor use of the networking and computing resources of the underlying environment.

An image compositing algorithm uses compositing operators to perform actual pixel blending. The most commonly used operator is *over*, which was introduced in [7]. It specifies the color and α channels of the output but puts no restrictions on the direction of data flow when performing compositing. So it is not an ideal operator to model the image compositing problem in heterogeneous environments where the direction of data flow has significant impact on the overall compositing efficiency.

In this paper, we propose a pipelining image compositing algorithm in heterogeneous networking environments. It is able to rearrange the direction of data flow of a compositing pipeline under strict ordering constraint. We introduce a novel directional compositing operator that specifies not only the color and α channels of the output but also the direction of data flow when performing compositing. Based on this operator, we thoroughly study the properties of compositing pipelines in heterogeneous environments.

It turns out that the number of all valid pipelines is exponentially large with respect to the number of nodes. Using a brute-force solution to find the optimal pipeline is obviously not feasible. We propose a dynamic programming algorithm that can find the optimal pipeline in polynomial time. To demonstrate the efficacy of our algorithm, we conduct a comprehensive evaluation on the ns-3 network simulator and study our algorithm from various perspectives. Experiment results demonstrate the efficiency of our method.

2. RELATED WORK

Parallel visualization has been proven to be an effective technique to analyze large-scale data sets. Molnar [8] classified the parallel rendering algorithms into three categories: sort-first, sort-middle, and sort-last. Among these categories, the sort-last algorithm has gained most attentions for its simplicity, good load balancing, and high scalability. Image compositing is the last stage of a sort-last parallel visualization system and is usually the bottleneck of all stages. Many algorithms have been proposed to address this problem.

Group-based methods. Group-based methods mainly include direct send [2], binary swap [3], 2-3 swap [4], and radix-k [5]. Nodes are divided into groups, and only nodes in the same group can exchange data with each other. The grouping scheme usually changes as the compositing process proceeds from one stage to another. Thanks to grouping, the number of data exchanges among the nodes is effectively reduced. Direct send partitions the image space evenly among the nodes. Each node takes care of one portion of the final image. The rendering results of all nodes within the corresponding region are sent directly to this node. This algorithm requires n messages to be transmitted. Binary swap reduces the number of exchanged messages per node from n to $\log n$, but it restricts the number of nodes to be a power-of-two. Yu [4] extended binary swap and proposed a new image compositing algorithm named 2-3 swap that enjoys similar benefits as that in binary swap but allows an arbitrary number of nodes. Peterka [5,9] proposed a configurable compositing algorithm called radix-k, which embodies and unifies binary swap and direct-send. As reported in [10,11], radix-k has lower cost than existing algorithms. It can overlap data transmission and computation without causing network congestion.

Pipeline-based methods. Pipeline-based mainly consist of the pipelining image compositing algorithm [6]. It arranges all participating nodes in a linear order. At first, each node holds a partial image. These partial images are then partitioned into a set of parts, which will flow through the nodes in a predetermined order. Once the pipeline is fully set up, it can composite one part of the final image at every time step. It has lower cost than binary swap with respect to the final image display time.

All these algorithms assume the environment is homogeneous. This assumption holds for traditional parallel computing environments, such as clusters and supercomputers. However, the networking environment is becoming more and more heterogeneous. These algorithms could no longer guarantee the optimal cost in a heterogeneous environment.

Table I. Notations.

Notation	Meaning
$\triangleright$	Directional compositing operator
s	A segment
p	A node
$p(s)$	The node hosting segment s
$\mathbb{P}$	A node set
$P(\boldsymbol{\pi})$	A pipeline
$\tilde{P}(\boldsymbol{\pi})$	Compositing result of $P(\boldsymbol{\pi})$
$t(P(\boldsymbol{\pi}))$	Compositing cost of $P(\boldsymbol{\pi})$
$B(p_a, p_b)$	Link bandwidth from p_a to p_b
$L(p_a, p_b)$	Link latency from p_a to p_b
$C(p_a)$	Computing power of p_a

3. PROBLEM FORMULATION

In this section, we firstly explain the basic concepts of image compositing, such as *segments* and the *over* operator. After that, we introduce the novel directional image compositing operator to address the limitations of the *over* operator. Based on this novel operator, we then thoroughly study the properties of compositing pipelines in heterogeneous environments. The notations that will be used in this paper are listed in Table I.

3.1. Segments

A *segment* is basically a collection of pixels with color and α channels. The involved segments will be blended together by the compositing operator to form the final result. A segment must reside on a node. The node hosting segment s is denoted as $p(s)$.

Given a viewpoint, there is an ordering among the hosting nodes. Each node is assigned a unique *rank* such that nodes with lower ranks are in front of the ones with higher ranks. At first, each segment inherits the rank from the hosting node. During compositing, segments with different ranks are blended together to produce a new segment. The newly formed segment has the contributions from different segments. Instead of storing the ranks of all these contributing segments, we choose to store only the lowest and highest ranks for space efficiency. In our work, a segment s is defined as

- $c(s)$: color channel;
- $\alpha(s)$: α channel;

DOI: 10.1002/cav

- $\overline{r}(s)$: the highest rank;
- $\underline{r}(s)$: the lowest rank; and
- $p(s)$: the node hosting s.

For two segments s_a and s_b, $s_a < s_b$ means

$$\overline{r}(s_a) < \underline{r}(s_b) \tag{1}$$

that is, s_a is in front of s_b. Two segments s_a and s_b are considered *comparable* if

$$s_a < s_b \text{ or } s_b < s_a \tag{2}$$

Note that all operators discussed in this paper are order-sensitive and should only operate on comparable segments.

3.2. *over* Operator

The most commonly used image compositing operator is *over*, which was introduced in [7]. Given two segments s_a and s_b ($s_a < s_b$), their compositing result $s_o = s_a \otimes s_b$, where $\otimes$ represents the *over* operator, is given by

$$\begin{aligned}
\alpha(s_o) &= \alpha(s_a) + (1 - \alpha(s_a))\alpha(s_b),\\
c(s_o) &= c(s_a) + (1 - \alpha(s_a))c(s_b),\\
\underline{r}(s_o) &= \min(\underline{r}(s_a), \underline{r}(s_b)),\\
\overline{r}(s_o) &= \max(\overline{r}(s_a), \overline{r}(s_b))
\end{aligned}$$

The color channel of each segment is premultiplied by the α channel. Originally, the *over* operator only handles the color and α channels. In our work, the compositing result also includes the lowest and highest ranks of contributing segments. The *over* operator does not specify the data flow direction when performing compositing. So $p(s_o)$, the node hosting the composting result, s_o, is undefined. It can be either $p(s_a)$ or $p(s_b)$.

3.3. Directional Image Compositing Operator

In a heterogeneous networking environment, the direction of data flow will in fact have a significant impact on the overall compositing efficiency. However, the conventional *over* operator puts no restrictions on the direction of data flow when performing image compositing. When using the *over* operator, it is possible that either the node in the front sends data to the node in the back or the node in the back sends data to the node in the front. This uncertainty makes it difficult to use *over* operator to formalize the compositing process in a heterogeneous networking environment. To address this problem, we introduce a directional image compositing operator denoted as $\triangleright$ that takes the direction of data flow into account. Given two segments, s_a and s_b, to compute $s_o = s_a \triangleright s_b$, $p(s_a)$ must send data to $p(s_b)$, but not vice versa. More precisely, $s_o = s_a \triangleright s_b$ is defined as

$$\begin{aligned}
s_o &= \begin{cases} s_a \otimes s_b & \text{if } s_a < s_b \\ s_b \otimes s_a & \text{if } s_b < s_a \end{cases},\\
p(s_o) &= p(s_b)
\end{aligned}$$

Note that we use the *over* operator $\otimes$ as the building block and modify $p(s_o)$ afterwards to ensure the directional property of the $\triangleright$ operator.

The $\triangleright$ operator has the following properties: Firstly, the $\triangleright$ operator is not commutative,

$$s_a \triangleright s_b \neq s_b \triangleright s_a. \tag{3}$$

Secondly, the $\triangleright$ operator is not associative,

$$((s_a \triangleright s_b) \triangleright s_c) \neq (s_a \triangleright (s_b \triangleright s_c)) \tag{4}$$

Thirdly, given a chain of $\triangleright$ operators with m segments, these $\triangleright$ operators are evaluated in a *left-to-right* order, that is,

$$s_{\pi_1} \triangleright s_{\pi_2} \triangleright \cdots \triangleright s_{\pi_{m-1}} \triangleright s_{\pi_m} \tag{5}$$

is a shorthand for

$$(((s_{\pi_1} \triangleright s_{\pi_2}) \triangleright \cdots \triangleright s_{\pi_{m-1}}) \triangleright s_{\pi_m}) \tag{6}$$

3.4. Compositing Pipelines

3.4.1. Pipeline Definition.

Given a set of n nodes, $\mathbb{P} = \{p_1, p_2, \cdots, p_n\}$. Initially, p_i holds a segment s_i and $\overline{r}(s_i) = \underline{r}(s_i) = i$. As mentioned before, $s_i < s_j$ if $i < j$. A *compositing pipeline* is an ordered sequence of n nodes. It could be characterized by a permutation π on set $\{1, 2, \cdots, n\}$, that is,

$$P(\pi) = p_{\pi_1} p_{\pi_2} \cdots p_{\pi_n} \tag{7}$$

where π_i is the ith element of permutation π. A pipeline specifies the *data flow direction* among the nodes. The data will flow from the upstream of the pipeline to the downstream.

3.4.2. Pipeline Cost.

Once the data flow direction is determined, the *cost* of a pipeline $P(\pi)$ is computed as

$$t(P(\pi)) = \max_{1 \le k < n} \left(L(p_{\pi_k}, p_{\pi_{k+1}}) + \frac{I}{nB(p_{\pi_k}, p_{\pi_{k+1}})} + \frac{I}{nC(p_{\pi_{k+1}})} \right)$$

where $L(p_a, p_b)$ and $B(p_a, p_b)$ are the link latency and bandwidth between p_a and p_b, respectively, $C(p_a)$ is the computing power of p_a, I is the size of the whole image space, and n is the number of participating nodes. Before compositing begins, each node holds a partial image, which has the same size as the whole image space. The partial images are then evenly partitioned into n segments, and each segment is of size I/n. These segments will flow in the pipeline and be blended together to form the final result. The total cost between two connecting nodes p_a and p_b in the pipeline consists of three parts:

- the inherent latency cost, which is determined by the physics law and independent of the segment size;
- the transmission cost, which is equal to the segment size I/n divided by the link bandwidth $B(p_a, p_b)$; and
- the computing cost, which is equal to the segment size I/n divided by the computing power $C(p_b)$.

3.4.3. Pipeline Compositing Result.

The compositing result $\tilde{P}(\pi)$ of pipeline $P(\pi)$ can be expressed by the $\triangleright$ operator, that is,

$$\tilde{P}(\pi) = s_{\pi_1} \triangleright s_{\pi_2} \triangleright \cdots \triangleright s_{\pi_n} \tag{8}$$

where s_{π_i} is the segment held by the node p_{π_i}.

To simplify the notations, $P(\pi)$, $\tilde{P}(\pi)$, and $t(P(\pi))$ will be written as P, $\tilde{P}$, and $t(P)$, respectively. Also, note that $\mathbb{P}$, P, $\tilde{P}$, and $t(P)$ support *slicing*. For example,

$$\mathbb{P}_{i:j} = \{p_i, p_{i+1}, \cdots, p_j\} \tag{9}$$

and

$$P_{i:j} = p_{\pi_i} p_{\pi_{i+1}} \cdots p_{\pi_j} \tag{10}$$

3.5. Valid Pipelines

As described in Section 3.1, an order-sensitive image compositing operator like $\triangleright$ can only operate on comparable segments; that is, there must be a relative ordering among the involved segments. Given n nodes, there are $n!$ possible pipelines in total, but not all of them are *valid*. A pipeline P is said to be valid if the $\triangleright$ operator only operates on comparable segments during the compositing process. This fact is highlighted in Lemma 1:

Lemma 1. *Given a valid pipeline* $P = p_{\pi_1} p_{\pi_2} \cdots p_{\pi_n}$, *for* $\forall k \in [1, n)$, $\tilde{P}_{1:k}$ *is comparable with* $s_{\pi_{k+1}}$.

3.5.1. Valid Pipelines Construction.

Having Lemma 1, we can determine the rank of the kth node π_k in the pipeline recursively. Consider the rank of the last node π_n. Because $\tilde{P}_{1:n-1}$ is comparable with s_{π_n}, we have

$$\pi_n < \underline{r}(\tilde{P}_{1:n-1}) = \min\{\pi_1, \pi_2, \cdots, \pi_{n-1}\}$$

or

$$\pi_n > \overline{r}(\tilde{P}_{1:n-1}) = \max\{\pi_1, \pi_2, \cdots, \pi_{n-1}\}$$

which implies

$$\pi_n = \min\{\pi_1, \pi_2, \cdots, \pi_n\} = 1 \tag{11}$$

or

$$\pi_n = \max\{\pi_1, \pi_2, \cdots, \pi_n\} = n \tag{12}$$

When π_n is determined, the same reasoning process can be applied to pipeline $P_{1:n-1}$. Using this method, we can compute all the valid pipelines.

3.5.2. Searching Space Size.

Let $f(n)$ denote the total number of valid pipelines for n participating nodes. From Equations (11) and (12), we could immediately determine π_n because it must be either 1 or n. Once π_n is determined, the number of valid pipelines of the remaining nodes is $f(n-1)$. As a result,

$$f(n) = 2f(n-1) \tag{13}$$

Considering the base case $f(1) = 1$, we have

$$f(n) = 2^{n-1} \tag{14}$$

Figure 1 lists all valid pipelines for nodes p_1, p_2, p_3, and p_4. Figure 2 illustrates the compositing processes of several different pipelines out of all valid ones. It can be seen that all these pipelines yield the same correct final result. In a heterogeneous networking environment, the compositing order among the nodes plays an important role to determine the compositing cost. One nice feature of pipeline-based image compositing methods is that they support progressive display of the final compositing result, as shown in Figure 3. The upper part of the image space completes its compositing process before the lower part. This feature may improve user experiences if the compositing takes a long time to finish.

3.6. Objective Function

Suppose $\mathbb{V}$ stands for the set of all valid pipelines of nodes $\mathbb{P}$. The goal of our algorithm is to find the optimal pipeline $Q(\mathbb{P})$ with minimal compositing cost from $\mathbb{V}$, that is,

$$Q(\mathbb{P}) = \arg\min_{P \in \mathbb{V}} t(P) \tag{15}$$

4. OUR METHOD

In this section, we present our pipelining image compositing algorithm in heterogeneous networking environments. As seen in Equation (14), the number of all valid pipelines is exponentially large. In order to quickly find the optimal pipeline, we propose a dynamic programming algorithm with $O(n^3)$ time complexity. The main idea is that the last node has only two possible ranks: the smallest one or the largest one. Once the rank of the last node is determined, the problem could be reduced to a smaller size. By adopting this strategy recursively, we could find the optimal pipeline in polynomial time.

4.1. A Dynamic Programming Solution

Given a node set $\mathbb{P} = \{p_1, p_2, \cdots, p_n\}$. For $\mathbb{P}_{i:j}$, according to Equations (11) and (12), the last node of a valid pipeline can be either p_j or p_i. We use $\overline{Q}(\mathbb{P}_{i:j})$ to denote the optimal pipeline with p_j being the last node and $\underline{Q}(\mathbb{P}_{i:j})$ to denote the optimal pipeline with p_i being the last node. The optimal pipeline can then be computed by

DOI: 10.1002/cav

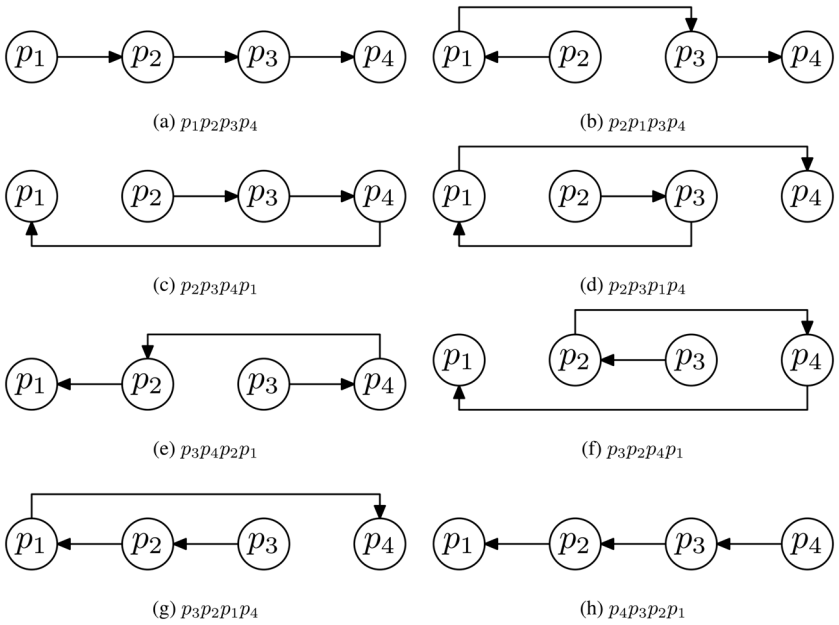


Figure 1. All valid pipelines for four nodes.

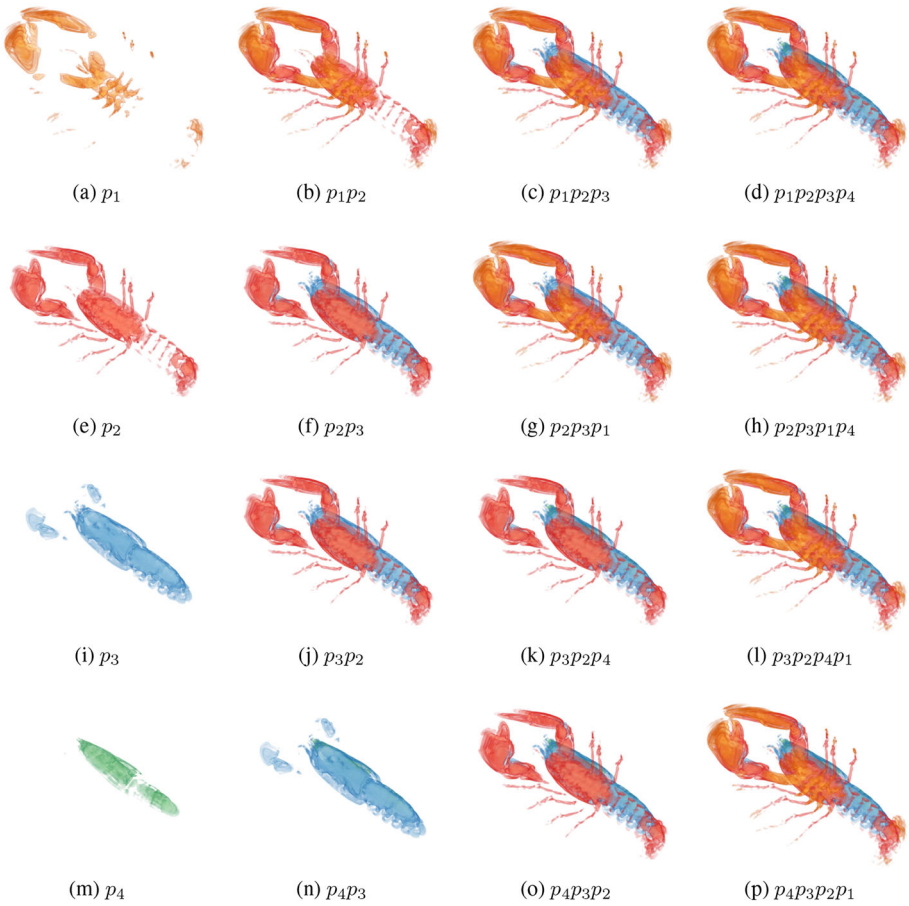


Figure 2. The compositing processes of different valid pipelines.

Figure 3. Progressive display of the final compositing result.

$$Q(\mathbb{P}_{i:j}) = \min(\overline{Q}(\mathbb{P}_{i:j}), \underline{Q}(\mathbb{P}_{i:j}))$$

where min() returns the pipeline with lower cost.

Suppose we have processed $\mathbb{P}_{i:j}$ and now move on to $\mathbb{P}_{i:j+1}$. Firstly, let us consider the case for $\overline{Q}(\mathbb{P}_{i:j+1})$ when p_{j+1} is the last node. There are two ways for $\mathbb{P}_{i:j}$ to be connected to p_{j+1}, either from $\overline{Q}(\mathbb{P}_{i:j})$ to p_{j+1} or from $\underline{Q}(\mathbb{P}_{i:j})$ to p_{j+1}. Therefore,

$$\overline{Q}(\mathbb{P}_{i:j+1}) = \min(\overline{Q}(\mathbb{P}_{i:j})p_{j+1}, \underline{Q}(\mathbb{P}_{i:j})p_{j+1})$$

where $\overline{Q}(\mathbb{P}_{i:j})p_{j+1}$ means concatenating the pipeline $\overline{Q}(\mathbb{P}_{i:j})$ with node p_{j+1}.

Secondly, let us consider the case for $\underline{Q}(\mathbb{P}_{i:j+1})$ when p_i is the last node. In this case, we could enumerate the position of p_{j+1} in the target pipeline P. Suppose p_{j+1} is at the kth position in P, that is, $\pi_k = j+1$. For any node with position $k' > k$, its rank can be determined directly:

$$\pi_{k'} = \min\{\pi_i, \pi_{i+1}, \cdots, \pi_{k'}\} = i + j - k'$$

This is because $\pi_{k'}$ cannot take the largest rank in $P_{i:k'}$ because p_{j+1} has taken position k. Consequently, it has to take the smallest rank instead. The ranks of the nodes before p_{j+1} (i.e., $\mathbb{P}_{j-k+2:j}$) can be computed recursively. Combining these two facts, we have

$$\begin{aligned}\underline{Q}(\mathbb{P}_{i:j+1}) = \min(&\\ &\overline{Q}(\mathbb{P}_{j-k+2:j})p_{j+1}p_{j-k+1}\cdots p_i,\\ &\underline{Q}(\mathbb{P}_{j-k+2:j})p_{j+1}p_{j-k+1}\cdots p_i)\end{aligned}$$

Finally, as we know, $Q(\mathbb{P}_{1:n})$ is the optimal pipeline for $\mathbb{P}$. By storing related information in the history matrix, we could reconstruct the optimal pipeline.

4.2. Time Complexity Analysis

Suppose the number of participating nodes is n. We first compute the time cost when moving from $\mathbb{P}_{i:j}$ to $\mathbb{P}_{i:j+1}$. In the case of $\overline{Q}(\mathbb{P}_{i:j+1})$, it just takes $O(1)$ time. In the case of $\underline{Q}(\mathbb{P}_{i:j+1})$, there are $O(n)$ possible positions for p_{j+1}. At any position k, it requires $O(1)$ time to compute the pipeline cost if we use some kind of caching mechanism. Therefore, the time cost of this single step is $O(n)$. After taking into account the $O(n^2)$ iterations in the outer loop (for i and j to vary from 1 to n, respectively), we can see that the algorithm's time complexity is $O(n^3)$.

5. RESULTS

5.1. Experiment Setup

Our experiment environment is built on *ns-3*, with the number of participating nodes ranging from 8 to 64. Given the participating nodes, each pair of nodes is directly connected through a point-to-point link. The latency and bandwidth of each link are randomly picked from given ranges. The computing power of each node is randomly picked from a given range. The main parameters used in the experiments are listed in Table II.

For each given parameter setting, we may carry out many rounds of image compositing simulations to reduce the impact of randomness. For R rounds of simulations, the *average speedup* is

$$\frac{1}{R}\sum_{i=1}^{R}\frac{t_i(other)}{t_i(our)} \tag{16}$$

where $t_i(other)$ is the compositing time of the algorithm to be compared with and $t_i(our)$ is the compositing time of our algorithm in round i, respectively.

We will also study the load-balancing problem among the nodes in a specific compositing stage. For m participating nodes, the *imbalance level* is defined as

$$1 - \frac{\sum_{i=1}^{m} t_i}{m \max_{i=1}^{m} t_i} \tag{17}$$

where t_i is the compositing cost of the node with rank i. The imbalance level is within range 0 and 1, where 0 implies the load among the nodes is perfectly balanced and 1 means the load is extremely imbalanced.

5.2. Comparison with Binary Swap and Radix-k

Binary swap and radix-k are group-based image compositing algorithms, whose compositing schemes are different

Table II. Experiment-controlling parameters.

Parameter	Meaning
N	Number of nodes
I	Image space size
CR	Computing power range
LR	Link latency range
BR	Link bandwidth range

DOI: 10.1002/cav

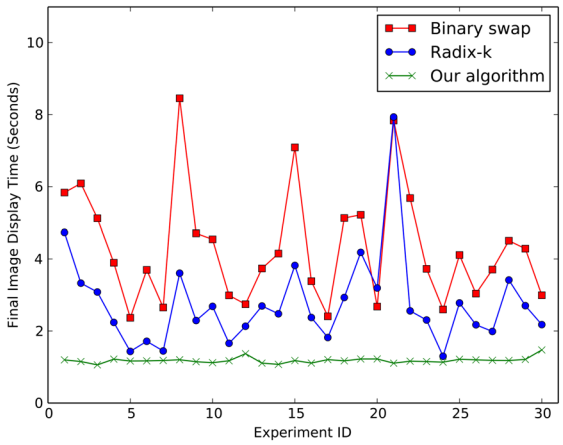


Figure 4. Final image display time of binary swap, radix-k, and our algorithm. $N = 32$, computing power range = $(1, 10^3)$, link bandwidth range =$(1, 10^2)$, and link latency range = $(10^{-5}, 10^{-3})$. Radix-k uses the configuration [4, 4, 2].

from that of pipeline-based compositing algorithms. As stated in [6], their proposed algorithm has advantage over binary swap with respect to the *final image display time*. Here, we will also compare these algorithms from this aspect. We will also study the load-balancing problem for conventional algorithms in heterogeneous networking environments.

5.2.1. Final Image Display Time.

The comparison results on final image display time of binary swap, radix-k, and our algorithm are shown in Figure 4. Radix-k usually outperforms binary swap because it is able to overlap data transmission and computation. However, in heterogeneous networking environments, it is possible that the cost of radix-k is larger than that of binary swap because they use different communication patterns, which can lead to different transmission costs.

The final image display time of our algorithm is lower than that of binary swap and radix-k. It is remarkable that the shape of our algorithm's results is much smoother than its competitors. This is because our algorithm can avoid the slow links if possible while binary swap and radix-k cannot. As a result, our algorithm can often find a good compositing scheme in the changing environments.

5.2.2. Load Balancing.

We study the load-balancing problem of binary swap, radix-k, and our algorithm, and the results are shown in Figure 5. The load of each node is the overall compositing cost in a stage, which is the sum of network communication cost and the pixel blending cost. For binary swap and radix-k, we pick the node costs from the first stage. For our algorithm, because not all nodes participate in every stage, we choose the first stage when the pipeline is fully set up.

Binary swap and radix-k are not aware of underlying heterogeneity. They are vulnerable to the slow links in the heterogeneous networking environments. Consequently, the imbalance levels of these two methods are higher than that of our algorithm.

5.3. Comparison with Wu's Algorithm

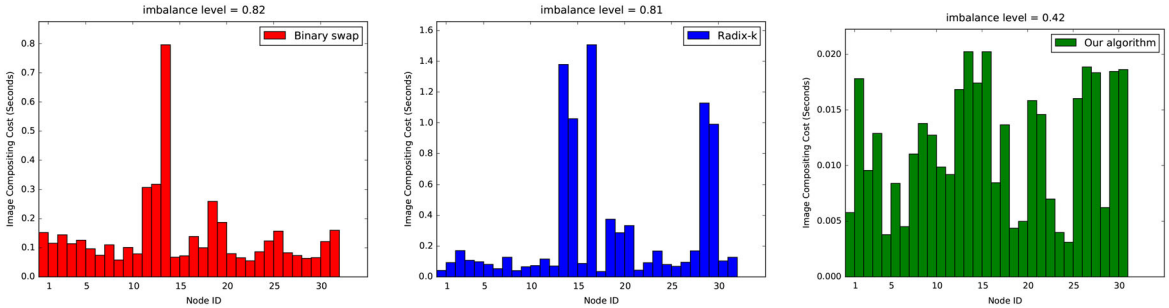
Wu [6] propose the conventional pipelining image compositing algorithm in homogeneous networking environments. Given N ordered nodes, the adopted pipeline is just the node sequence arranged in the natural front-to-back order:

$$p_1 p_2 \cdots p_N \tag{18}$$

As for a pipeline, the overall compositing cost is bounded by the slowest link. There are at most $N - 1$ links in the pipeline for N participating nodes, so it is easy to be affected by slow links in heterogeneous networking environments.

5.3.1. Image Compositing Cost.

The results on image compositing cost of Wu's algorithm and our algorithm are shown in Figure 6. It can be seen that our algorithm gains lower compositing cost than that of Wu's algorithm. Besides, the variance of the costs of our algorithm is lower than that of Wu's algorithm. Our algorithm manages to compute the optimal pipeline under the given environment and is able to avoid slow links



(a) Binary swap. (b) Radix-k. (c) Our algorithm.

Figure 5. Load-balancing results of binary swap, radix-k, and our algorithm. $N = 32$, computing power range $= (1, 10^3)$, link bandwidth range $= (1, 10^2)$, and link latency range $= (10^{-5}, 10^{-3})$.

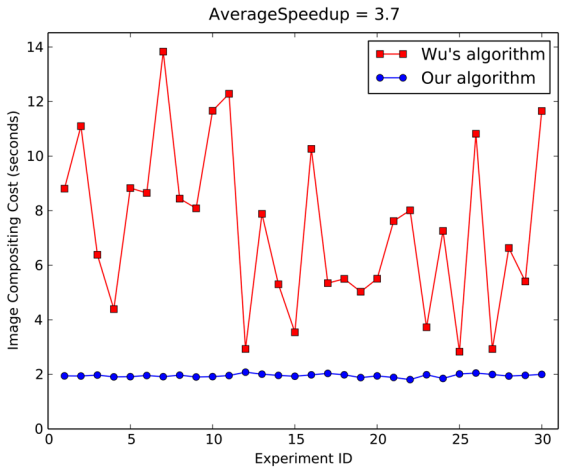


Figure 6. Image compositing costs of Wu's algorithm and our algorithm. $N = 64$, computing power range = $(1, 10^3)$, link bandwidth range = $(1, 10^2)$, and link latency range = $(10^{-5}, 10^{-3})$.

whenever possible. So it is more robust under the changing environments.

5.3.2. Different Types of Bandwidth.

In this section, we study the results of different types of link bandwidths. The computing power of each node is set to a large value, and the latency of each link is set to a small value, so that their impacts on the compositing cost are negligible. The results are shown in Figure 7.

It can be seen that as the bandwidth range increases, the speed gap between Wu's algorithm and our algorithm becomes larger. This is because when the bandwidth range is larger, the variance of the link bandwidths tends to grow. As a result, the networking environment will become more heterogeneous. When the environments become more heterogeneous, Wu's algorithm will be more likely to be affected by slow links , while our algorithm has more room to do optimizations and achieve lower compositing cost. So the speedup of our algorithm compared with Wu's algorithm tends to increase.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a pipelining image compositing algorithm for heterogeneous networking environments. The goal is to avoid slow links while satisfying the ordering constraint among the nodes. To achieve this goal, we introduce a novel directional image compositing operator. It specifies the direction of data flow during compositing, which is different from the conventional *over* operator. We develop an algorithm to find the optimal pipeline with $O(n^3)$ time complexity. To demonstrate the efficacy of our solution, we conducted a comprehensive evaluation on ns-3, with the number of participating nodes varying from 8 to 64, and provided a thorough discussion of the results.

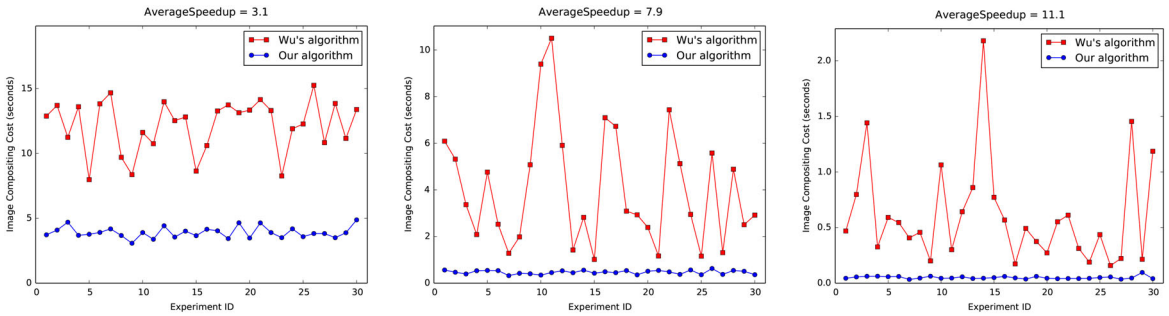
Here, we mainly focus on pipeline-based image compositing methods. In the future, we will study the potential of incorporating the ideas of other image compositing algorithms, such as the group-based ones, in heterogenous networking environments to further enhance the efficiency of our method. Because of stability issues, we only carried out experiments on a software simulator rather than on physical machines. It will be beneficial to test our algorithm in real-world environments if possible.

ACKNOWLEDGEMENTS

We would like to thank the anonymous reviewers for their valuable suggestions. This work is funded by the National Natural Science Foundation of China (Grant Nos. 61173067, 61379085, 61532002) and the National High Technology R&D Program of China (Grant No. 2015AA016401).

REFERENCES

1. Bentes C, Labronici BB, Drummond LM, Farias R. Towards an efficient parallel raycasting of unstructured



(a) $BR = (1, 10)$. (b) $BR = (1, 100)$. (c) $BR = (1, 1000)$.

Figure 7. The results of different types of link bandwidths. $N = 32$, computing power range = $(10^9, 10^9)$, and link latency range = $(10^{-9}, 10^{-9})$.

DOI: 10.1002/cav

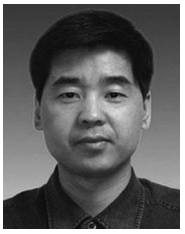
volumetric data on distributed environments. *Cluster Computing* 2014; **17**(2): 423–439.

2. Neumann U. Parallel volume-rendering algorithm performance on mesh-connected multicomputers. In *Proceedings of the 1993 Symposium on Parallel Rendering*, New York, NY, USA, 1993; 97–104.
3. Ma KL, Painter J, Hansen C, Krogh M. Parallel volume rendering using binary-swap compositing. *IEEE Computer Graphics and Applications* 1994; **14**(4): 59–68.
4. Yu H, Wang C, Ma KL. Massively parallel volume rendering using 2-3 swap image compositing. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, Piscataway, NJ, USA, 2008; 48:1–48:11.
5. Peterka T, Goodell D, Ross R, Shen HW, Thakur R. A configurable algorithm for parallel image-compositing applications. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*. ACM, New York, NY, USA, 2009; 4:1–4:10.
6. Wu Q, Gao J, Chen Z, Zhu M. Pipelining parallel image compositing and delivery for efficient remote visualization. *Journal of Parallel and Distributed Computing* 2009; **69**(3): 230–238.
7. Porter T, Duff T. Compositing digital images. *11th Annual Conference on Computer graphics and Interactive Techniques (SIGGRAPH '84)* 1984 January; **18**: 253–259.
8. Molnar S, Cox M, Ellsworth D, Fuchs H. A sorting classification of parallel rendering. *IEEE Computer Graphics and Applications.* 1994 July; **14**: 23–32.
9. Peterka T, Ross R. Versatile communication algorithms for data analysis. In *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*. Springer-Verlag, Berlin, Heidelberg, 2012; 275–284.
10. Kendall W, Peterka T, Huang J, Shen HW, Ross R. Accelerating and benchmarking radix-k image compositing at large scale. In *Proceedings of the 10th Eurographics Conference on Parallel Graphics And Visualization*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2010; 101–110.
11. Moreland K, Kendall W, Peterka T, Huang J. An image compositing solution at scale. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, New York, NY, USA, 2011; 25:1–25:10.

AUTHORS' BIOGRAPHIES

Ning Liu is a PhD student at the Institute of Computing Technology, Chinese Academy of Sciences. He received his BS degree from Shandong University in 2009. His research interests include scientific visualization and volume rendering.

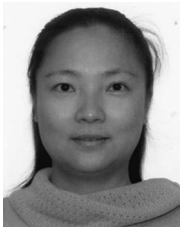
Dengming Zhu is an associate researcher at the Institute of Computing Technology, Chinese Academy of Sciences. He received his PhD degree from the Chinese Academy of Sciences in 2009. His research interests include fluid simulation, scientific visualization, and volume rendering.



Zhaoqi Wang is a researcher and doctoral supervisor at the Institute of Computing Technology, Chinese Academy of Sciences. He received his PhD degree from Beihang University in 1999. His research interests include virtual reality and intelligent human-computer interaction.

Hong Qin is a professor at the Computer Science Department at Stony Brook University. He received his PhD degree from the University of Toronto in 1995. His research interests include computer graphics, geometric and physics-based modeling and data mining.

Jianfeng Zhan is a researcher at the Institute of Computing Technology, Chinese Academy of Sciences. He received his PhD degree from the Chinese Academy of Sciences in 2002. His research interests include distributed and parallel systems.



Jinzhu Gao is an associate professor at the Computer Science Department at the University of the Pacific. She received her PhD degree from the Ohio State University in 2004. Her research interests include big data analysis and visualization.