# 高精細タイルドディスプレイを用いた並列ボリュームレンダリングシステムの実装

坂井陽平[†] 浅野琢也[†]
福間慎治[†] 森眞一郎[†]

本稿ではタイルドディスプレイをターゲットとした，並列ボリュームレンダリングにおける並列画像合成アルゴリズムの改良と実装結果について報告する．従来の画像合成アルゴリズムでは生成される最終合成画像が 1 台のノードに集約されるため，タイルドディスプレイへ転送する際の通信帯域幅が集約されたノード 1 台に依存し可視化のボトルネックとなっていた．改良したアルゴリズムでは中間画像の合成をディスプレイの台数まで行い，タイルドディスプレイへ送信することにより，通信帯域幅が大きくなり画像転送時間を短縮した．また，PC クラスタを用いて実装し，台数の違いによる処理時間の比較を行った．

# Implementation of parallel volume rendering system using high resolution tiled display

Yohei Sakai,[†] Takuya Asano,[†] Shinji Fukuma[†]
and Shin-ichiro Mori[†]

This paper reports the improved parallel image composition algorithm for sort-last parallel volume rendering system with tiled display system for high resolution image display and its implementation results. In the conventional system, the rendering subsystem totally composes the image into one node and then the node distributes the image to each display nodes. This image distribution process incurs the bottleneck as the number of display increases to generate high resolution image. In order to decrease this bottleneck, the proposed composition algorithm generates partially composed images such that the each image corresponds to one display in the tiled display system. Through this improvement, our system can aggregate the network bandwidth between rendering subsystem and tiled display system, and thus it could achieve higher frame rate for high resolution image.

## 1. はじめに

近年，計算機システムや計測技術の性能が向上し，取り扱うデータの大規模化や複雑化が急速に進んでいる．このような大規模かつ複雑なデータを人間が直感的に理解するために，データの可視化技術が必要になった．この可視化技術の一つとしてボリュームレンダリング処理があり，一台の計算機では実時間処理が不可能な大規模データに対しては計算機を並列に連携して大規模なデータを分割して処理する並列ボリュームレンダリング手法が用いられている．

また可視化した高精細なデータを表示する技術として，複数のディスプレイを格子状に配置し 1 つの大きなディスプレイとして利用することで高精細な画像を表示できるタイルドディスプレイがある．

我々は，従来より並列ボリュームレンダリングにより生成された高解像度の画像をタイルドディスプレイに表示するボリュームレンダリングシステムの構築を行っている．しかし，従来の並列ボリュームレンダリングにおける画像合成アルゴリズムでは，生成される最終合成画像が 1 台のノードに集約される．そのため，タイルドディスプレイへ画像を転送する際の通信帯域幅が最終合成画像を集約したノード 1 台のネットワーク性能で律速され，可視化のボトルネックとなってしまう．これを解消するために並列ボリュームレンダリングを行うアプリケーションサーバ側とタイルドディスプレイへ表示を行うディスプレイサーバ間の通信帯域幅を増やし画像転送時間を短縮する必要がある．

本論文ではアプリケーションサーバとディスプレイ

† 福井大学大学院工学研究科
Graduate School of Engineering, University of Fukui

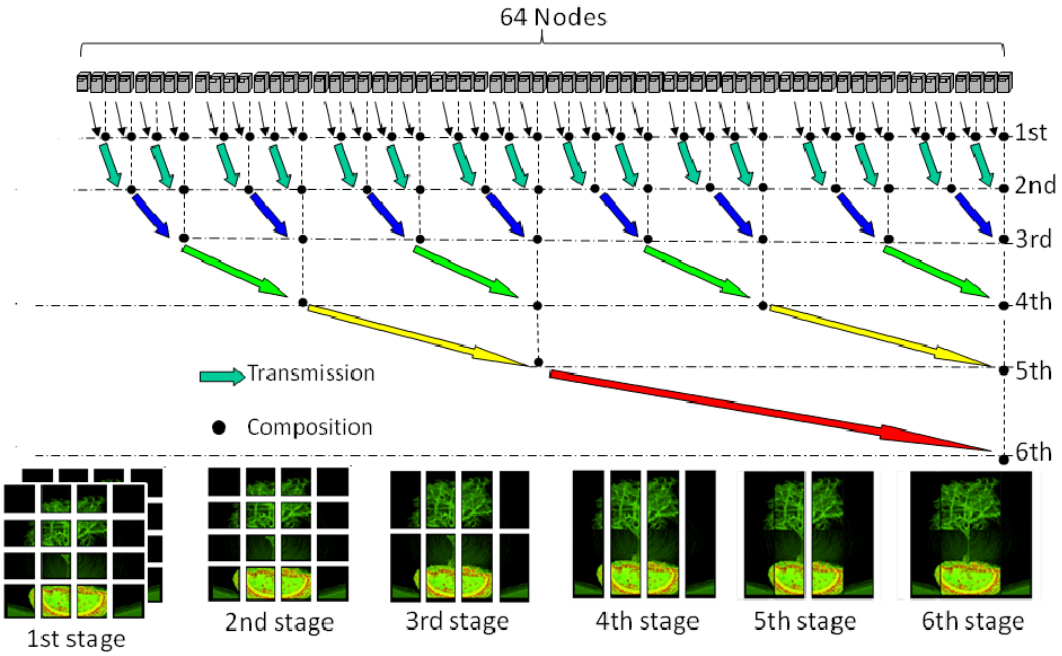


**図 1** 主軸優先木構造合成の各ステージにおける中間画像
Fig. 1 Intermediate Images generated at each stage of PAA-PTC scheme.

サーバ間における通信帯域幅を増やすために並列画像合成アルゴリズムを改良し，実装を行った結果を報告する．

## 2. 関 連 研 究

### 2.1 並列画像合成アルゴリズム

コストパフォーマンスの高い高性能 GPU の普及により，GPU 内のメモリに格納できるサイズのボリュームデータであれば，リアルタイムの高精細ボリュームレンダリングが可能となってきた．その結果，並列ボリュームレンダリングシステムにおける性能のボトルネックは，各ノードで生成された中間画像を最終画像まで合成する並列画像合成処理に移ってきた．特に，並列度が高いシステムにおいてはこの問題が顕著となってきた[1]．

並列ボリュームレンダリングにおける並列画像合成処理の高速化を目指した研究としては Binary-Swap Compositing[2] や SLIC[3] などの研究がある．Binary-Swap Compositing は全ての合成ステージで，全てのノードを利用する高い並列性を持ったアルゴリズムである．時間計算量としては良質のアルゴリズムであるが，適切なノード数でないと並列処理のオーバーヘッドが大きくなる．

SLIC は，各ノードが生成した中間画像間の重複関係を視線方向に基づいて解析し，合成の必要がない背景領域や他の中間画像と重なりのない領域を並列画像合成の対象から省くことで合成処理の演算量を削減する．合成処理の対象から省かれた領域に対応する中間画像は，必要に応じて最終画像表示ノードへ直接転送を行う．負荷の均等化に際しては，各ノードにおいてスキャンライン内での中間画像の重複回数と重複状態を求め，その重複状態が同じ範囲 (スパン) を負荷分散の単位として，スパン単位の合成処理を各ノードに静的に割当てる負荷分散方式を採用している．重複状態を考慮することによる演算量削減の効果は大きいと考えられるが，SLIC で提案されている負荷分散方式では，アルゴリズムの実装に際し，ノード間の通信パターンの不規則性が増加し，ノード間のネットワークに高いランダム通信性能が要求されることになる．

### 2.2 主軸優先木構造合成

並列画像合成アルゴリズムとして木構造合成をベースとした並列合成アルゴリズムを採用すると，合成処理が進むにつれ通信量と演算量が次第に増加していく．特に，中間画像の合成を行うノードを静的に決定する単純な木構造合成では，視線方向と合成の順番との相対的な関係により処理の早い段階で通信量が激増する可能性がある．

そこで，合成の順番を実行時に動的に判断し，中間画像間の重複が大きいものを優先して合成することで総通信量と演算量を軽減する手法として，我々は主軸

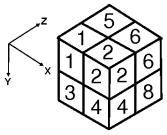


**図 2** サブボリュームと座標軸の対応
Fig. 2 sub-volumes along axes

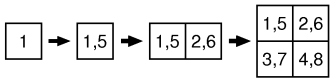


**図 3** 中間画像の合成過程: 通信コストが少ない例
Fig. 3 Footprints of intermediate images: Best case example.

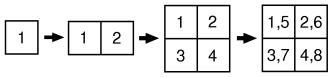


**図 4** 中間画像の合成過程: 通信コストが大きい例
Fig. 4 Footprints of intermediate images: Worst case example.

優先木構造合成アルゴリズムを提案した (図 1)[4)5)].

具体的には，2 分木構造に基づく合成処理において，合成処理に参加する各ノードは，各々のステージにおいて，視線ベクトルの絶対値が最も大きい成分 (第 1 主軸) 方向の隣接ノード間との合成を優先して行う．これにより，各ノードの合成処理において最も重複の多い中間画像間での合成が可能となり，合成結果として得られる画像サイズの不必要な増加を抑制する．また，各々のステージではノード間での通信が $X$,$Y$ あるいは $Z$ 軸のいずれか一つのみに平行な極めて規則的な通信パターンとなり，ネットワークに対するランダム通信性能の要求を軽減することが可能である．

例えば，ボリュームデータを 8 個のサブボリュームに分割している場合，図 2 のようにサブボリュームをボリュームデータ固有の座標系の $X,Y,Z$ 方向と対応させる．視線ベクトルを $(x,y,z)$ と表すとする．ボリュームデータに対して視線ベクトルが $(0,0,1)$ のとき $Z$ 方向の隣接するサブボリュームと先に合成を行うと図 3 のように中間画像が重なり画像サイズの増加を抑えている．ここで，先に $X$ 方向の隣接するサブボリュームと合成を行うと図 4 のように中間画像が全く重ならず合成の初期段階でサイズが増加してしまう．視線ベクトルの各成分の絶対値の大きさで合成する軸に優先順位をつけることで中間画像の重複部分が大きい軸方向 (主軸) から合成を行うことができる．この操作を第 1 主軸に沿って合成すべきノードがなくなるまで繰り返し，次に第 2 主軸に沿って同様の処理を行い，最後に第 3 主軸に沿った合成を行うと最終的にすべての中間画像を合成した最終合成画像が完成する．

### 2.3 タイルドディスプレイと可視化システム

コストパフォーマンスの高い複数の高解像度ディスプレイをタイル状に配置し，超高解像度のディスプレイシステムを実現する方法としてタイルドディスプレイシステムが実用化されている．タイルドディスプレイの実現方法にはマルチモニタ対応の GPU を利用した小規模なものとクラスタベースのものがある．クラスタベースのタイルドディスプレイはコモディティPC を LAN などで相互接続したものであり[6)]，コストパフォーマンスと解像度に拡張性がある．

データをディスプレイノードに送る方法としてはグラフィックス API レベルでの実装法[7)] とディスプレイマネージャレベルでの実装法[9)] がある．前者は，API を使用できるアプリケーションならば意識せず容易に対応することができるとともに，表示すべき画像のソースが複数ノードに分散して配置されている場合にも対応可能である．これらの代表として Chromium[7)] と SAGE[8)] があげられる．一方ディスプレイマネージャレベルの実装では，メニューやツールバーを含めた全てのウィンドウアプリケーションをタイルドディスプレイへ表示することが可能であるが，画像のソースがシングルノードのシステムに限定されるためスケーラビリティに問題がある．

Chromium は OpenGL の API で描画された画像データをノードのフレームバッファから取得し，各々のディスプレイに表示するグラフィックス API レベルのシステムであり，高解像度の表示装置である Hyperwall[10)]，VisWall[6)]，LionEyes Display Wall[11)] などと組み合わせて利用されている．しかし，Chromium はレンダリングした画像を 1 つのノードから全てのディスプレイノードに送信するため 1 台のノードの通信帯域幅に依存しており，広域ネットワーク (WAN) での利用に向いていない．これに対して SAGE は，複数アプリケーションの同時表示や，実行時のウィンドウ操作に対応するとともに，レンダリングノードとディスプレイノード間が WAN で結ばれた低速・高遅延の環境に対応できるという柔軟性をもっている．そこで，本研究ではタイルドディスプレイシステムの実装に当たっては，SAGE(Scalable Adaptive Graphics Environment) を利用することとした．ボリュームレンダリング結果をタイルドディスプレイに表示するためには，SAGE が提供する SAIL(SAGE Application Interface Library) ライブラリの API コードをボリュームレンダリングを行うアプリケーションプログラムに組み込めばよい．これにより SAGE 上で実行可能な SAIL アプリケーション (ノード) となる．

大規模なボリュームデータの可視化システムとして，vol-a-tile[12)] がある．時系列で出力された大規模なデータセットをボリュームレンダリングしタイルドディスプレイへ表示する．データセットは，OptiStore とい

う遠隔にあるデータストアから専用のリンクを使って，指定した部分のみボリュームデータを取得しボリュームレンダリングを行う．その際，マスターノードはカラーマップなどの可視化パラメータや視線方向の操作をインタラクティブに行うことができ，MPI を使って視線パラメータをレンダリングノードへ向けてブロードキャストを行う．GUI による操作以外はマスターノードは全く処理は行わず，命令を受けたレンダリングノードが行っている．画像データの送信には SAGE が利用されており，各レンダリングノードで生成された画像を，同期を取って交換し各ディスプレイが表示する．大規模データの解析を支援する完成度の高いシステムではあるが，SAGE と組み合わせることを前提としたアプリケーションの最適化については言及されていない．これに対して，本論文で提案するシステムは，タイルドディスプレイとの連携に際し画像生成処理自体の最適化を考慮したシステムである．

タイルドディスプレイ上での描画速度をあげる手法としては，各ディスプレイが表示すべき領域に対応する 3 次元データ（あるいは全ての 3 次元データ）を対応するディスプレイノードに事前に分配する Sort-First 型の画像合成アルゴリズムを用いるアプローチがある[13),14)]．この手法は，視点位置が固定（あるいは一定の制約条件下）の場合には画像合成処理において通信が発生しないため高速であるが，描画領域や視点位置の変更にリアルタイムで追従することが困難である．これに対して我々の研究では大規模データの可視化を目指しており，3 次元データの事前再配置が非現実的な状況化においても任意視点からのリアルタイム高精細表示を可能にすることを目指している．

## 3. 合成アルゴリズムの設計方針

この章では，前述の主軸優先木構造合成アルゴリズムを用いた並列ボリュームレンダリングシステム（以下，アプリケーションサーバ，**AS** と呼ぶ）と，SAGE を用いたタイルドディスプレイシステム（以下，ディスプレイサーバ，**DS** と呼ぶ）を連携させたタイルドディスプレイ向けボリュームレンダリングシステムを構築する手法を検討する．

### 3.1 基本合成アルゴリズム

並列ボリュームレンダリングにおいて単純で効率的な並列画像合成アルゴリズムは木構造合成である．木構造の通信は 1 段通信するたびに通信に参加するノードが半分になり，ノード数を $N$ とすると通信回数は $\log N$ 回となる．合成の終盤に近づくにつれて通信量と合成の演算量が増加する．$\log N$ 回の合成後には全ての中間画像を合成した最終合成画像が 1 台のノードに集約される．本研究では $N(=n^3)$ 台のノードを $n \times n \times n$ の 3 次元格子状に論理的に配置する．各ノードにはボリュームデータを $N(=n^3)$ 個のサブボリュームに分割した領域の 1 つを割り当て，$X, Y, Z$ 軸のいずれかと平行な方向でそれぞれ $\log n$ 回ずつ木構造合成を行う．ただし，今回の実装では後述の通りスクリーンサイズに応じて合成フェーズ後半の一部のフェーズを省略する実装となる．

### 3.2 SAGE との連携方針

主軸優先木構造合成アルゴリズムはシングルディスプレイ向けの合成アルゴリズムのため最終合成画像は 1 台のノードが持つことになる．つまり，単純に AS と DS を連携させただけでは，DS へ転送する際に集約された 1 台の通信帯域幅に依存するため通信ボトルネックが発生し高精細画像のリアルタイム表示の支障となる．これを防ぐために DS で表示すべき画像を AS から並列転送することでボトルネックを解消する手段を検討する．並列転送を行うには，最終合成画像の分割が必要になる．分割画像の生成方法として，木構造合成を最終段まで行い最終合成画像を生成した後に DS の台数に分割する方法と，木構造合成を DS の台数になるまで合成した後，分割された状態で異なる合成方法を用いて各々の画像を最終合成画像とする方法を挙げる．

本実装において，分割した画像を並列に転送するミドルウェアとして SAGE を利用する．SAGE ではディスプレイ内の指定した領域の画像転送をどの AS ノードが担当するかをあらかじめ設定する必要がある．つまり，分割した画像は，決められた AS のノードが最終的に持っていなければならない．この点と先に述べた分割画像の生成方法と合わせて検討すると，主軸優先木構造合成を最終段まで行う方法では 1 台が最終合成画像を持つため，この 1 台の画像を分割し決められた AS のノードへ転送すればよい．しかし，DS の台数まで主軸優先木構造合成を行う方法では，$X, Y, Z$ 軸の合成順が主軸の優先順位に依存するため，分割された画像を持つノードが一意に定まらない．ところが，各ノードの論理的位置 $(X, Y, Z)$ とサブボリュームの座標位置 $(x, y, z)$ の関係は相対的なものであり，この対応関係を視点位置に応じて変更すれば，常に合成画像を特定のノードに集約させることが可能になる．ただし，サブボリュームを並べ替えるのはオーバーヘッドが極めて大きいため，各ノードに初期配置されたサブボリューム (3 次元) に対して，そのボリュームレンダリングした後の中間画像 (2 次元) に対して 3 次元

的に配置を並び替えることで視点変更に伴うオーバーヘッドを軽減する．

### 3.3 合成アルゴリズムの方針

以上より AS と DS を連携方法として以下の方針が考えられる．なお，AS を構成するノード数を $N$, DS を構成するノード数を $M$ とし，AS と DS 間の通信は，AS 内通信や DS 内通信に比べて通信遅延時間が大きく，スループットも低いことを想定する．また，DS を構成するノードはタイルドディスプレイシステムとしてのコストパフォーマンスを鑑み，ディスプレイサーバとして十分な性能をもつが AS のノードと比べると処理性能が低いものとする．

**Type I　シングルヘッド主軸優先木構造合成**

1. 集約（合成）フェーズ：主軸優先木構造合成を用いて最終合成画像を生成する．
2. 分割フェーズ：1 台が持つ最終合成画像を $M$ 分割し一旦 AS 内の他の $M-1$ 台のノードに再分散を行う．
3. 転送：AS 内の $M$ 台のノードから DS 内の $M$ 台のノードへ 1 対 1 通信する．

**Type II　マルチヘッド主軸優先木構造合成 (図 5)**

1. 置換フェーズ：ボリュームレンダリング後の中間画像を視点位置に応じて並び替える．
2. 集約（合成）フェーズ：主軸優先木構造合成を最終合成が完了する以前の段階で一旦中断し AS 内の複数のノードが中間合成画像を保持する状態を作る．具体的には，DS のノード数と同じ $M$ 台のノードに中間合成画像が集まった時点で木構造合成処理を中断する．
3. 調整フェーズ：今，中間合成画像を保持するノード群を $G_M$ と呼ぶこととする．木構造合成処理を途中で終了したため，$G_M$ 内の各ノードが保持する中間画像には更なる合成が必要な領域が存在する．また，$G_M$ 内の各ノードが保持する画像と DS の各ノードでの表示位置が未だ 1 対 1 対応となっていない．そこで，$G_M$ 内のノードが保持する画像の表示領域と DS の各ノードでの表示位置が 1 対 1 となるよう $G_M$ 内のノード間で画像の交換を行うとともに，必要に応じて交換された画像の合成を行う．その結果，各ノードには $M$ 分割された最終合成画像ができあがる．
4. 転送：$G_M$ の各ノードが保持する画像を DS の各ノードに 1 対 1 通信で送信することで AS-DS 間の通信ボトルネックを解消する．

TypeI は，実装がシンプルで AS 内の内部ネットワークが AS-DS 間のネットワークに比べて高速，低遅延である場合には有効であるが，一旦合成したデータを 1 対 $M$ の通信が必要となる再分散で冗長な通信処理が発生するという問題点がある．これに対して TypeII では，木構造合成を途中で中断し一旦未完成な中間合成画像を生成するが，その後の交換処理では必要最小限のデータのみを $G_M$ 内で交換することで冗長な通信の発生を回避できている．

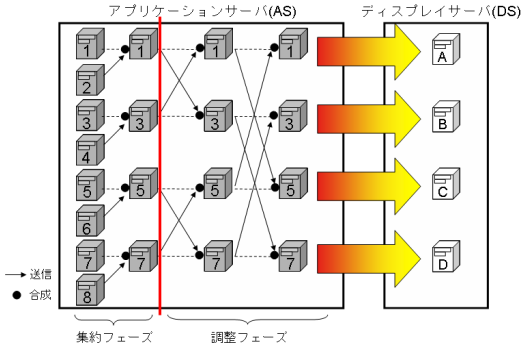


**図 5**　マルチヘッド主軸優先木構造合成
Fig. 5　Multi Head Prioritized Axis Aligned Parallel Tree Composition.

## 4. システム実装の詳細

並列ボリュームレンダリングシステムにおいて逐次的に処理を進めた場合の流れは以下の様になる．
(1)AS で，GPU によるレンダリング処理を行い中間画像を生成する（レンダリング）
(2) 中間画像を視点からの前後関係を考慮しながら合成を行い最終合成画像を得る（合成）
(3) 得られた画像データをディスプレイ送信用バッファに格納する（バッファリング）
(4)AS から DS へ送信する（画像送信）
(5)DS が画像データを受信する（画像受信）
(6) ディスプレイに表示する（表示）
以上が一連の処理である．

このとき，(1) のレンダリングは AS 内の GPU での処理であり，CPU での処理となる (2) の合成とは並列に実行可能である．また，一旦 SAGE ラインタイムシステムのバッファにバッファリングされたデータはバックグランド処理として DS へ転送される．従って，AS 側では (1)，(2)+(3)，ならびに (4) の 3 つのステージからなるソフトウェアパイプラインを構成した．(2) の処理は更にいくつかのステージに分割して高速化することが可能であるが今回の実装では 1 つのステージとして実装した．図 6 にシステム全体の処理の流れを示す．合成処理とバッファリング処理，画像転送処理の処理時間がシステム全体の処理時間に影響

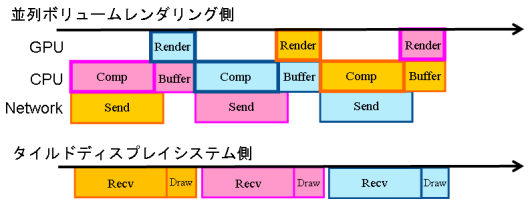


**図 6** システム全体の流れ
Fig. 6 A flow of the total system.

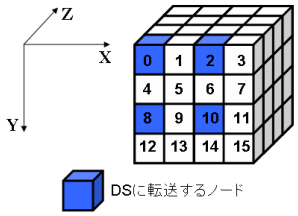


**図 7** DS へ転送する AS ノードの位置
Fig. 7 The position of the AS node to transfer to DS.

を及ぼすことが分かる．

ここでは，$N(=n\times n\times n)$ 台の計算機で構成される並列ボリュームレンダリング用アプリケーションサーバと，$M(=m\times m)$ 台のディスプレイで構成されるタイルドディスプレイに対応したアルゴリズムに改良する．なお，$n\geq m$ とし，$n,m$ は 2 のべき乗とする．アプリケーションサーバの各ノードには，ボリュームデータを同一サイズの $N(=n\times n\times n)$ 個のサブボリュームに分割された領域の 1 つを割り当てるものとする．その際，サブボリューム空間 $(i,j,k)$ は $(k\times n^2+j\times n+i)$ 番目のノードに割り当てるものとする．

以下，マルチヘッド主軸優先木構造合成の各フェーズについて説明する．

### 4.1 置換フェーズ

SAGE では指定したノードがディスプレイのどの領域を描画するかあらかじめ指定する必要がある．よって，最終合成画像を DS の $M$ 台へ並列転送する AS のノード $M$ 台を 1 対 1 で指定しなければならない．ノードの位置を，担当するサブボリューム空間と同じ配置 $(i,j,k)$ で指定すると，$(2^s i_m,2^s j_m,0)$ となる．ただし $s=\log n-\log m$，$i_m=0,1,...,\frac{n}{2^s}-1$，$j_m=0,1,...,\frac{n}{2^s}-1$ である．図 7 のように $N=64$ $(n=4)$，$M=4$ $(m=2)$ の場合では．一番手前側の $xy$ 面のノードの内 $(0,0,0)$，$(2,0,0)$，$(0,2,0)$，$(2,2,0)$ の 4 台である．この 4 台はそれぞれ左下，右下，左上，右上の画像を最終的に所持する．しかし，主軸優先木構造合成では最終合成画像を持つノードが主軸の優先順位により異なるという性質を持っている．そこで，この置換フェーズでは主軸優先木構造合成の完了後に上記の AS のノードに集約するように各ノードが持つ生成画像を $N$ 台間で交換する．方針として $(i,j,k)$ の 3 次元サブノード空間を第 1 主軸が $Z$ 軸となるようにし，さらに画像が上記で指定したディスプレイの担当描画領域に合うようにノード配置を変更する．具体的には，まず第 1 主軸が $X$ または $Y$ の場合は転置処理を行う．

$-X$ 方向の場合，$(i,j,k)$ と $(n-1-k,j,n-1-i)$
$+X$ 方向の場合，$(i,j,k)$ と $(k,j,i)$
$-Y$ 方向の場合，$(i,j,k)$ と $(i,n-1-k,n-1-j)$
$+Y$ 方向の場合，$(i,j,k)$ と $(i,k,j)$
$-Z$ 方向の場合，転置処理なし
$+Z$ 方向の場合，転置処理なし

この処理により第 1 主軸が $Z$ 軸になる．次に，画像の向きが正しければそのままで完了，正しくない場合はさらに交換を行う．

上下交換の場合，$(i,j,k)$ と $(i,n-1-j,k)$
左右交換の場合，$(i,j,k)$ と $(n-1-i,j,k)$
回転交換の場合，$(i,j,k)$ と $(n-1-i,n-1-j,k)$
転置交換の場合，$(i,j,k)$ と $(n-1-j,j-1-i,k)$
　　もしくは，$(i,j,k)$ と $(j,i,k)$

この処理により並列転送する AS が指定した担当領域と $N$ 台が所持する生成画像の向きが一致する．

以上が置換フェーズである．

### 4.2 集約（合成）フェーズ

集約（合成）フェーズでは，$N$ 台のノードが持つ中間画像をディスプレイサーバの台数と同じ $M$ 台まで，主軸優先木構造合成に基づいて合成を行う．しかし，単純に $M$ 台になるまで合成を行うのではなく，ディスプレイの配置 (指定した $M$ 台の AS) を考慮して各主軸での合成回数を変える必要がある．ここでは，$N=64$ $(n=4)$，$M=4$ $(m=2)$ の場合で，主軸の優先順位を $z,y,x$ として説明する．ノードの位置は，担当するサブボリューム空間と同じ配置 $(i,j,k)$ で指定する．

まず，$xy$ 面のノードが $+z$ 方向から $-z$ 方向へ合成を行う．1 ステップ目は，合成を行うノードを $(i,j,k)$(ただし $k$ は偶数) とすると，送信するノードは $(i,j,k+1)$ となる．2 ステップ目は，$(i,j,k)$ と $(i,j,k+2)$ で合成する．その結果，ノード位置 $(i,j,0)$(ただし $i=0,1,...,n-1$，$j=0,1...,n-1$) の 16 台 (図 7 では一番手前側の 4×4 の 16 台) に中間合成画像が集約される．一般に $s$ ステップ目の合成は，$(i,j,2^s k)$ と $(i,j,2^s k+2^{s-1})$ で行われる．ただし

$s = 1, 2, ..., \log n$，$k = 0, 1, ..., \frac{n}{2^s} - 1$ である．これを $\log n$ ステップまで行うことで第 1 主軸方向での隣接ノード内の合成が終了し，第 1 主軸に沿った $n$ 台のノードの合成結果が $n \times n$ 台に集約される．

次に，集約された $n \times n$ 台で $+y$ 方向から $-y$ 方向へ合成を行う．合成を行うノードを $(i, j, 0)$(ただし $j$ は偶数) とすると，送信するノードは $(i, j+1, 0)$ となる．この 1 ステップの合成により，8 台に集約される．一般に $s$ ステップ目の合成は，$(i,2^s j,0)$ と $(i,2^s j+2^{s-1},0)$ で行われる．ただし $s = 1, 2, ..., (\log n - \log m)$，$j = 0, 1, ..., \frac{n}{2^s} - 1$ である．これを $(\log n - \log m)$ ステップまで行うことで第 2 主軸方向での隣接ノード内の合成が終了し，ノード位置 $(i, j, 0)$(ただし $i = 0, 1, ..., n-1$，$j = 0, 1, ..., m-1$) の $n \times m$ 台に集約される．

この段階で $y$ 方向のノード数がディスプレイの $y$ 方向の台数と同じになる．

最後に，集約された $n \times m$ 台で $+x$ 方向から $-x$ 方向へ合成を行う．合成を行うノードを $(i, j, 0)$(ただし $i$ は偶数) とすると，送信するノードは $(i+1, j, 0)$ となる．この 1 ステップの合成により，ディスプレイ台数と同じ 4 台に集約される．一般に $s$ ステップ目の合成は，$(2^s i,j,0)$ と $(2^s i + 2^{s-1},j,0)$ で行われる．ただし $s = 1, 2, ..., (\log n - \log m)$，$i = 0, 1, ..., \frac{n}{2^s} - 1$ である．これを $(\log n - \log m)$ ステップまで行うことで第 3 主軸方向での隣接ノード内の合成が終了し，$m \times m$ 台に集約される．この例で集約されるノード番号は図 7 において 0，2，8，10 となる．

以上が集約フェーズである．他の主軸の順番でも同様に処理を行い，$m \times m$ 台まで合成を行う．

### 4.3 調整フェーズ

このフェーズでは以下の 2 つの問題を解消する．1 つは，AS 側の集約した $m \times m$ 台の部分画像間で，ある画素 $(i,j)$ に対応する中間画像が依然として複数存在しており，最終合成画像が完成していないという問題と，もう 1 つは，AS 側の $(k,l)$ ノードが持っている部分画像の表示領域が必ずしも DS 側の $(k,l)$ ノードの表示範囲内に存在しているとは限らない点である．

図 8 の場合，A の画像を持つノードは左上のディスプレイの表示を担当するが，担当領域には B や C，D の画像も含まれており，A 自身の表示も他のディスプレイの領域に入っている．これは，その他のノードにも同じ事がいえる．よって，正しい最終合成画像を作るための合成処理と，AS 側のノード $(k,l)$ に関して AS 側の画像表示領域を DS 側の表示領域に合わせる作業が必要となる．

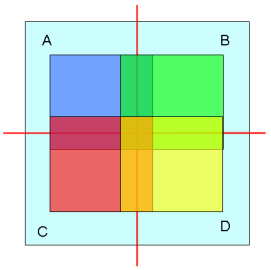


**図 8** ディスプレイと画像の位置関係
Fig. 8 The position of an images and the displays.

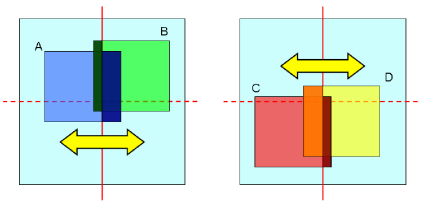


**図 9** 横方向の交換
Fig. 9 The swap of the horizontal direction.

まず横方向の隣接ノードと画像の交換を行う．図 9 において，横方向で A の表示範囲を越えている部分を B に送り合成を行う．これにより，B は A との重なりが解消する．次に，横方向で B の表示範囲を越えている部分を A に送り合成を行う．これにより，A,B 間での重なりが解消する．同様に，C と D の間でも交換する．

縦方向についても同様に交換を行う．縦方向で A の表示範囲を越えている部分を C に送り，合成を行う．このとき送信する画像は，1 つ前に合成した B の画像も含まれているため，C には A と B の画像データが送られる．同様に B と D の間でも交換する．

## 5. 評　　価

### 5.1 実 行 環 境

アプリケーションを SAGE に対応するために，プログラムに SAIL(SAGE Application Interface Library) の API コードを追加した．タイルドディスプレイに転送するノードはバッファを用意し，最終合成画像を格納し，ディスプレイサーバに送信することでタイルドディスプレイへ描画する．実行環境を表 5.1 に示す．並列ボリュームレンダリングを行うアプリケーションサーバ 8 台または 64 台，生成された画像をタイルドディスプレイに表示するためのディスプレイサーバ 4 台，タイルドディスプレイシステムを制御するための管理ノード 1 台を 1Gbps のネットワークケーブルで接続する．また，並列ボリュームレンダリングには OpenGL グラフィックライブラリ，MPICH2 通信

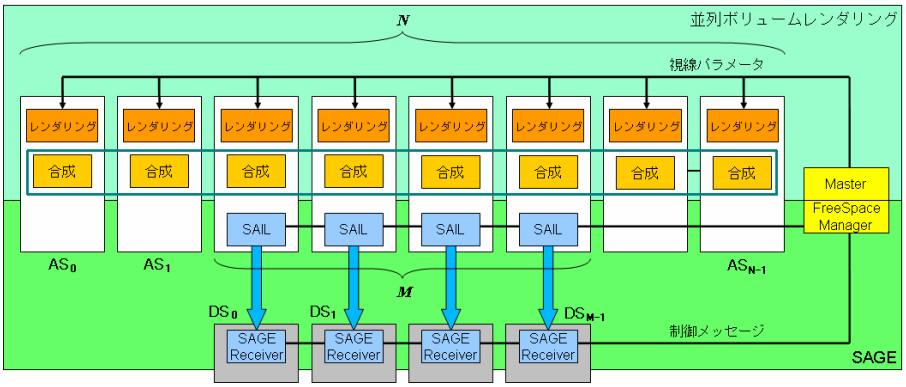


**図 10** AP8 台での実行環境の概要
Fig. 10 Experimental Environment

**表 1** サーバノードの仕様
Table 1 Hardware specification of server nodes.

| | AS | DS |
|---|---|---|
| CPU | Pentium4 3.4GHz | Pentium4 3.0GHz |
| Memory | 1GB | 1GB |
| GPU | GeForce 6800 GT | GeForce FX 5950Ultra |
| GPU Mem | 256MB | 256MB |
| OS | Fedora Core 6 | Fedora Core 6 |
| Network | 1GbE | 1GbE |

ライブラリを用いた．ボリュームデータサイズは $512^3$ で，生成する最終合成画像サイズは $2048^2$ とした☆．最悪条件での評価を行うため不透明度に関してはボクセル値に関わらず全て 1.0 とすることで全画素に有効な色を与えるとともに，ボリュームレンダリング時のアーリーレイターミネーションや画像の圧縮は行っていない．

### 5.2 実験結果

$X$ 軸についての回転を 0 度とし $Y$ 軸を中心に 5 度刻みで 360 度の回転を 10 回行ったときの平均を表 2，表 3 に示す．また，$X$ 軸を中心に 45 度回転させた状態で $Y$ 軸を中心に回転させた場合を表 4，表 5 に示す．

Type II について，置換フェーズでは 8 台と 64 台で比較すると 1 台あたりの転送する画像サイズが 64 台の方が小さいため処理時間が短くなった．集約（合成）フェーズに注目すると，Type I と比較して，木構造合成処理での通信量の多い終段の処理を省略しているため処理時間が約 1/2 となり，大きな削減効果を確認できた．また，8 台の場合に比べて 64 台での合成時間が倍以上になっているが，これは前者ではサイズが $1024^2$ クラスの画像合成 1 ステージ分のみで処理が終了するケースであるのに対して，後者では $512^2$ クラスの画像合成からスタートする 4 ステージの画像合成となるためである．更にプロセッサ台数を増やした場合の処理時間の増加は理論上は数%程度である[5]．調整フェーズについては 8 台，64 台ともに集約された 4 台間で行われるためほぼ同じ時間であることが確認

☆ 今回の実験では使用する GPU の制約からこのサイズのボリュームデータを使用しオーバーサンプリングにより高解像度の画像を生成した．最新の GPU を使用すれば，各ノードに $1024^3$ 程度のボリュームデータを分配したうえで 30fps 程度の画像生成が可能であるため，今回の実験で使用したデータサイズが提案アルゴリズムの評価に影響を与えるものではない．

**表 2** ($X$ 軸:0 度)Type I の処理時間 [msec]
Table 2 (X-axis is zero degree)Processing time of Type I.

| | 8 台 [min/max/ave] | 64 台 [min/max/ave] |
|---|---|---|
| 集約フェーズ | 112.4 / 192.2 / 158.7 | 125.2 / 217.8 / 180.0 |
| 分割フェーズ | 69.1 / 101.1 / 88.7 | 75.5 / 108.1 / 96.5 |
| 合計 | 181.5 / 293.3 / 247.4 | 200.7 / 325.9 / 276.5 |

**表 3** ($X$ 軸:0 度)Type II の処理時間 [msec]
Table 3 (X-axis is zero degree)Processing time of Type II.

| | 8 台 [min/max/ave] | 64 台 [min/max/ave] |
|---|---|---|
| 置換フェーズ | 7.06 / 43.8 / 30.9 | 4.35 / 20.6 / 12.8 |
| 集約フェーズ | 31.3 / 55.3 / 47.1 | 38.7 / 100.4 / 74.7 |
| 調整フェーズ | 1.09 / 51.9 / 28.2 | 1.38 / 48.8 / 27.1 |
| 合計 | 39.4 / 151.0 / 106.2 | 44.4 / 169.8 / 114.6 |

**表 4** ($X$ 軸:45 度)Type I の処理時間 [msec]
Table 4 (X-axis is 45 degree)Processing time of Type I.

| | 8 台 [min/max/ave] | 64 台 [min/max/ave] |
|---|---|---|
| 集約フェーズ | 178.2 / 374.7 / 291.6 | 184.9 / 434.3 / 334.7 |
| 分割フェーズ | 99.0 / 172.2 / 145.0 | 105.8 / 186.0 / 156.4 |
| 合計 | 277.2 / 546.9 / 436.6 | 290.7 / 620.3 / 491.1 |

**表 5** ($X$ 軸:45 度)Type II の処理時間 [msec]
Table 5 (X-axis is 45 degree)Processing time of Type II.

| | 8 台 [min/max/ave] | 64 台 [min/max/ave] |
|---|---|---|
| 置換フェーズ | 38.2 / 200.1 / 130.1 | 13.1 / 65.1 / 39.4 |
| 集約フェーズ | 46.4 / 89.3 / 72.7 | 63.1 / 137.3 / 99.6 |
| 調整フェーズ | 41.5 / 158.0 / 102.3 | 41.0/ 150.0 / 97.6 |
| 合計 | 126.1 / 447.4 / 305.1 | 117.2 / 352.4 / 236.6 |

できた．

## 6. ま と め

本研究では，高精細に画像を提示できるタイルドディスプレイシステムに対応した高精細ボリュームレンダリングシステムを提案し実装した．SAGE との連携を考慮した，置換フェーズの導入ならびにマルチヘッド主軸優先木構造合成の導入によりボリュームレンダリング結果としての $2048^2$ サイズの画像をタイルドディスプレイを利用し 5〜10fps の速度で表示することが可能となった．しかし，最終合成結果画像が大きくなると，従来のアルゴリズムを単純に採用した場合，1 台のノードから複数のディスプレイに転送する際に，送信する 1 台のネットワーク性能に依存するという問題があったため，タイルドディスプレイにおける合成アルゴリズムの検討を行った．

そして，今回のアルゴリズムは主軸優先木構造合成を元に改良し，画像を送信するノードを増やすことでタイルドディスプレイ向けに対応させた．集約（合成）フェーズではディスプレイの台数になるまで集約を行い，調整フェーズではディスプレイの担当表示領域間の合成を行った．したがって，任意のディスプレイ台数に合わせた並列画像合成が可能となった．本研究では SAGE を利用して，アプリケーションサーバ側で各々のディスプレイ表示範囲の画像を完成させ，ディスプレイサーバへ送信する手法を採用した．

また，送信ノード数 4 台の場合で画像転送時間を測定した．結果として画像転送時間が短くなったり，今後パイプライン処理した場合に最長のステージとなる可能性を低下させた．しかし，SAGE による送信ノードの制約のため生成画像の交換を行う置換フェーズの追加により合成時間の短縮効果とは別に，画像の単純な移動処理が必要となってしまいオーバーヘッドとなった．今後，複数のネットワークを利用した合成処理のパイプライン化を行うことで．高速化できると考えられる．

## 参 考 文 献

1) Hongfeng Yu; Chaoli Wang; Grout, R.W.; Chen, J.H.; Kwan-Liu Ma; , "In Situ Visualization for Large-Scale Combustion Simulations," Computer Graphics and Applications, IEEE , vol.30, no.3, pp.45-57, May-June 2010.
2) Kwan-Liu Ma; Painter, J.S.; Hansen, C.D.; Krogh, M.F.; , "Parallel volume rendering using binary-swap compositing," Computer Graphics and Applications, IEEE , vol.14, no.4, pp.59-68, Jul 1994.
3) Stompel, A., et al.; , "SLIC: Scheduled Linear Image Compositing for Parallel Volume Rendering," Proc. IEEE Symp. on Parallel and Large-Data Visualization and Graphics, pp.33-40, 2003.
4) T.Asano, T.Yoshimura, H.Shimada, S.Mori, S.Tomita;"Large Scale Volume Rendering on the Sensable Simulation System," Int'l Workshop on Super Visualization, June 2008.
5) 吉村知普:"体感型シミュレーションシステム Scube の構築と可視化性能の評価", 京都大学大学院情報学研究科修士論文, 2006 年 2 月.
6) VisWall High Resolution Display Wall, http://www.visbox.com/wallMain.html, 2010.
7) G. Humphreys, M. Houston, R. Ng, R. Frank, S. Ahern, P.D. Kirchner, and J.T. Klosowski, "Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters," Proc. ACM SIGGRAPH '02, pp. 693-702, 2002.
8) Byungil Jeong; Renambot, L.; Jagodic, R.; Singh, R.; Aguilera, J.; Johnson, A.; Leigh, J.; , "High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment," SC 2006 Conference, Proceedings of the ACM/IEEE.
9) Distributed Multihead X (DMX) Project, http://dmx.sourceforge. net, 2010.
10) Sandstrom, T.A.; Henze, C.; Levit, C.; , "The hyperwall," Proc. Int'l Conf. on Coordinated and Multiple Views in Exploratory Visualization 2003, pp. 124- 133, 2003.
11) LionEyes Display Wall, http://viz.aset.psu.edu/ga5in/DisplayWall.html, 2010.
12) Schwarz, N.; Venkataraman, S.; Luc Renambot; Krishnaprasad, N.; Vishwanath, V.; Leigh, J.; Johnson, A.; Kent, G.; Nayak, A.; , "Vola-Tile - A Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays," Visualization, 2004. IEEE , pp. 19p-19p, 10-15 Oct. 2004.
13) Bethel, E.W., et al., "Sort-first, distributed memory parallel visualization and rendering," PVG 2003.
14) J. Allard, et al.,"A Shader-Based Parallel Rendering Framework," IEEE Vis 2005.
15) 浅野琢也："主軸優先合成アルゴリズムを用いた並列ボリュームレンダリングの実装と高速化," 福井大学大学院工学研究科修士論文, 2010 年 2 月.
16) Nirnimesh; Harish, P.; Narayanan, P.J.; , "Garuda: A Scalable Tiled Display Wall Using Commodity PCs," Visualization and Computer Graphics, IEEE Transactions on , vol.13, no.5, pp.864-877.