情報処理学会論文誌　コンピューティングシステム　Vol.5　No.3　124–136 (May 2012)

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[1]　浅野 琢也[1]　福間 慎治[1]　森 眞一郎[1,a)]

受付日 2011年10月7日, 採録日 2012年1月30日

概要：本論文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．

キーワード：ボリュームレンダリング，タイルドディスプレイ，可視化，並列処理，PC クラスタ

Implementation of Parallel Volume Rendering System Using High Resolution Tiled Display

Yohei Sakai[1]　Takuya Asano[1]　Shinji Fukuma[1]　Shin-ichiro Mori[1,a)]

Received: October 7, 2011, Accepted: January 30, 2012

Abstract: 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.

Keywords: volume rendering, tiled display, visualization, parallel processing, PC cluster

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 1 つとしてボリュームレンダリング処理があり，1 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．得られた高精細なデータを表示する技術としては，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（**図 1** 参照）．しかし，従来の並列ボリュームレンダリ

[1] 福井大学大学院工学研究科
Graduate School of Engineering, University of Fukui, Fukui 910–8507, Japan
[a)] moris@u-fukui.ac.jp

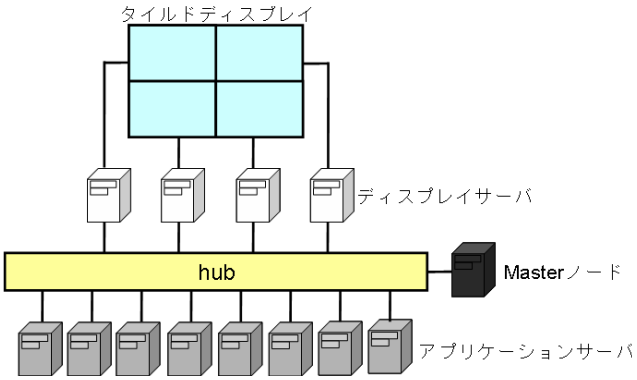


図 1　システム構成の概念図
Fig. 1　Overview of the hardware component.

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．これを解消するには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に関する関連研究

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 はすべての合成ステージで，すべ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り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．

これに対し，我々は中間画像の重複を考慮して合成処理過程での中間合成画像サイズの増大を軽減し，さらに合成処理過程で発生するノード間の通信パターンに規則性を持たせ適切な通信のスケジューリングによりネットワーク上の輻輳を軽減可能な主軸優先木構造合成アルゴリズムを提案しており [4]，本研究ではこの並列画像合成アルゴリズムの利用を前提とする．アルゴリズムの詳細については 2.4 節で説明する．

2.2 タイルドディスプレイの実現法に関する関連研究

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（Scalable Adaptive Graphics Environment）[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すべ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

つのノードからすべ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持っている．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あたっては，SAGE 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．

2.3　可視化システムレベルの関連研究

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ノードはまったく処理は行わず，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，同期をと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上げる手法としては，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（あるいはすべての 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．たとえば，文献 [14] のシステムでは，ボリュームデータ再配置に要する時間を除いて，画像圧縮可能な実画像のレンダリングを 4,096 × 3,072 のスクリーンに 2〜5 fps で描画可能である．しかし，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そもそも不可能あるいは非現実的な状況下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．そのため，視点に追従したボリュームデータの再配置が不要な Sort-Last 型のアプローチをとっている．

2.4　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，我々は主軸優先木構造合成アルゴリズムを提案した（**図 2**）[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 1 つのみに平行なきわ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．

たとえば，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 3** 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 4** 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 5** のように中間画像がまった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．

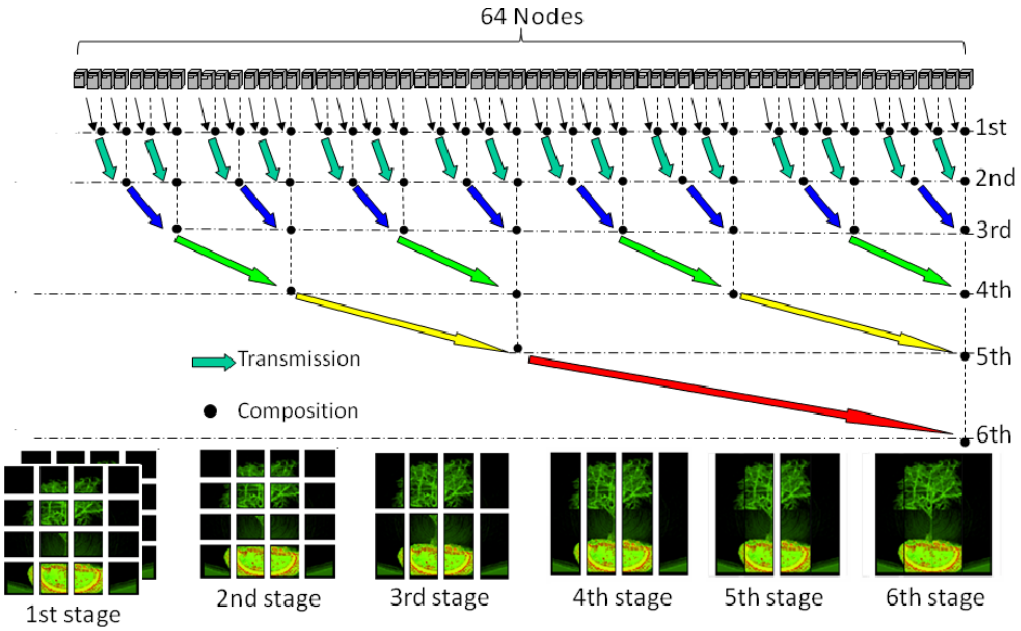


図 2　主軸優先木構造合成の各ステージにおける中間画像
Fig. 2　Intermediate images generated at each stage of PAA-PTC scheme.

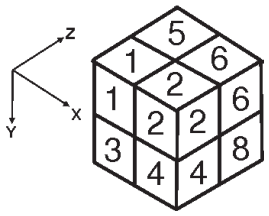


図 3　サブボリュームと座標軸の対応
Fig. 3　Sub-volumes along axes.

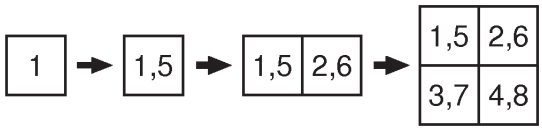


図 4　中間画像の合成過程：通信コストが少ない例
Fig. 4　Footprints of intermediate images: Best case example.

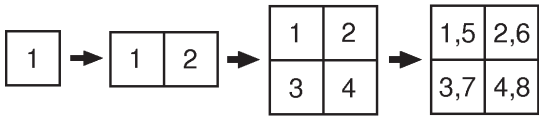


図 5　中間画像の合成過程：通信コストが大きい例
Fig. 5　Footprints of intermediate images: Worst case example.

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．$\log N$ 回の合成後にはすべ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とお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あげる．

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，つね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ヘッドがきわめて大きいため，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 次元的に配置を並べ替えることで視点変更にともなうオーバヘッドを軽減する．この手法では，すべてのノードが 2 次元画像の再配置に寄与するため総トラフィックは大きいが，再配置を行うノード対で交換されるデータはボリュームレンダリング直後の小さな画像であるため，通信時間そのものを低減させることが可能である．

3.3　合成アルゴリズムの方針

以上より AS と DS の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考え，ディスプレイサーバとして十分な性能を持つが 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，いったん 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 6）

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べ替える．
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いったん中断し 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さら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いまだ 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．

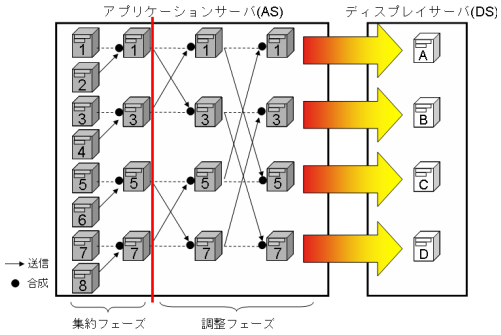


図 6　マルチヘッド主軸優先木構造合成

Fig. 6　Multi head prioritized axis aligned parallel tree composition.

Type I 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，いったん合成したデータに対して 1 対 M の通信が必要となる再分散を行う必要があり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 Type II では，木構造合成を途中で中断し，いったん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．

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の実装において，一連の処理を逐次的実装すると以下の 6 ステップを順次実行することになる．

(1) AS で，GPU によるレンダリング処理を行い中間画像

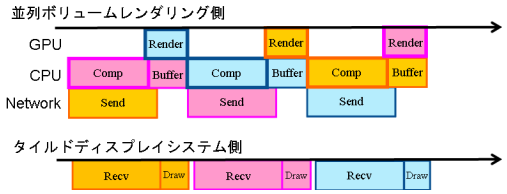


図 7　システム全体の流れ
Fig. 7　A flow of the total system.

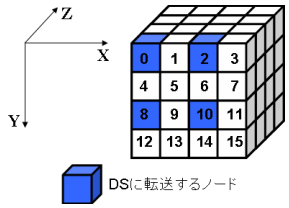


図 8　DS へ転送する AS ノードの位置
Fig. 8　The position of the AS node to transfer to DS.

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）．

このとき，(1) のレンダリングはその大半が AS 内の GPU で処理が行われるため，CPU にはほとんど負荷がかからない．そこで，連続する 2 フレームの画像に対して GPU でのレンダリング処理と CPU でのその他の処理をパイプライン処理することでスループットの向上が可能である．また，いったん SAGE ラ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．したがって，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) の処理はさら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 7** 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を及ぼすことが分かる．なお，明示的な同期は (1) と (3) の終了を待って (2) と (4) を開始するための一カ所のみに挿入されている．

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した領域の 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 $s = \log n - \log m$，$i_m = 0, 1, \ldots, \frac{n}{2^s} - 1$，$j_m = 0, 1, \ldots, \frac{n}{2^s} - 1$ である．**図 8** のように $N = 64$（$n = 4$），$M = 4$（$m = 2$）の場合では，一番手前側の xy 面のノードのうち $(0,0,0)$，$(2,0,0)$，$(0,2,0)$，$(2,2,0)$ 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 の場合は以下の転置処理を行う．

$-X$ 方向の場合，(i, j, k) と $(n-1-k, j, n-1-i)$ の置換

$+X$ 方向の場合，(i, j, k) と (k, j, i) の置換

$-Y$ 方向の場合，(i, j, k) と $(i, n-1-k, n-1-j)$ の置換

$+Y$ 方向の場合，(i, j, k) と (i, k, j) の置換

次に，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異なる場合には，必要に応じて以下の交換を行う．

上下交換の場合，(i, j, k) と $(i, n-1-j, k)$ の置換

左右交換の場合，(i, j, k) と $(n-1-i, j, k)$ の置換

回転交換の場合，(i, j, k) と $(n-1-i, n-1-j, k)$ の置換

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) の置換

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 $i = 0, 1, \ldots, n-1$，$j = 0, 1, \ldots, n-1$）の 16 台（図 8 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 $s = 1, 2, \ldots, \log n$，$k = 0, 1, \ldots, \frac{n}{2^s} - 1$ 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 $s = 1, 2, \ldots, (\log n - \log m)$，$j = 0, 1, \ldots, \frac{n}{2^s} - 1$ 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 $i = 0, 1, \ldots, n-1$，$j = 0, 1, \ldots, m-1$）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 $s = 1, 2, \ldots, (\log n - \log m)$，$i = 0, 1, \ldots, \frac{n}{2^s} - 1$ 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 8 において 0，2，8，10 となる．

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という点である．

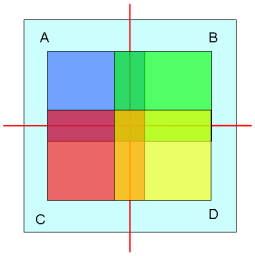


図 9　ディスプレイと画像の位置関係
Fig. 9　The relative position of images and the displays.

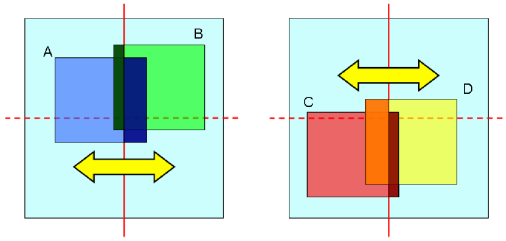


図 10　横方向の交換
Fig. 10　The swap of the horizontal direction.

図 9 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ことがいえる．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．

まず横方向の隣接ノードと画像の交換を行う．**図 10** 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．実行環境の概要を**図 11** ならびに**表 1** に示す．並

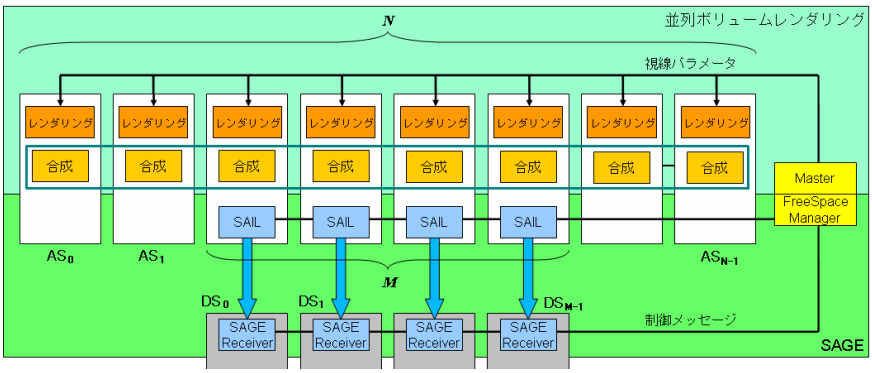


図 11　実行環境の概要
Fig. 11　Experimental environment.

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

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

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 1 Gbps 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 $2{,}048^2$ とした[*1]．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かかわらずすべて 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．これは**図 12** で使用したような実データを用いた表示では，有意なデータが表示される画素数がボリュームデータに依存して変動するとともに，ボリュームレンダリングに特有な透明度の設定によっても表示に必要な有効画素数を低減できてしまうためである．

5.2　実験結果

X 軸についての物体の回転を 0 度とし，視点を Y 軸を中心に 5 度刻みで 360 度回転させたときの処理時間を**表 2** および**表 3** に示す．また，X 軸を中心に物体を 45 度回転させた状態で，視点を Y 軸を中心に回転させた場合を**表 4** ならびに**表 5** に示す．以下では両者を区別するために，それぞれの測定条件を X0 の場合，ならびに X45 の場合と表

[*1] 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 $1{,}024^3$ 程度のボリュームデータを分配したうえで 30 fps 程度の画像生成が可能であることを確認しており，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．

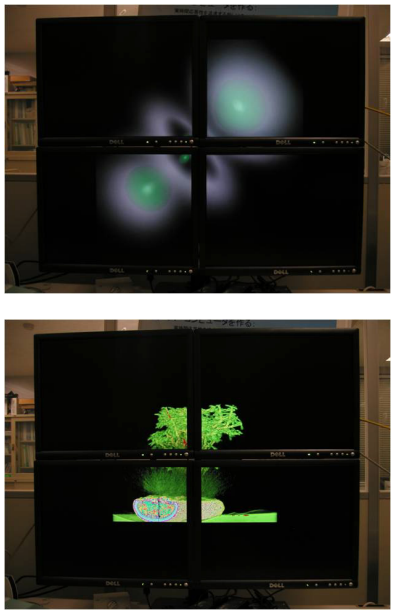


図 12　ボリュームレンダリング結果の表示例
Fig. 12　Example of volume rendering image on tiled display.

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.

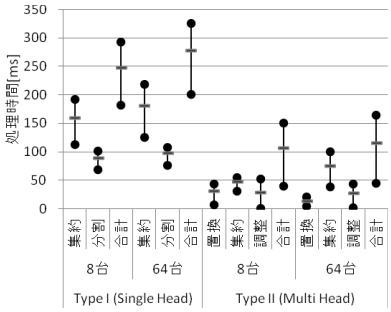
	8 台 [min/max/ave]	64 台 [min/max/ave]
置換フェーズ	7.06 / 43.8 / 30.9	4.35 / 20.6 / 12.8
集約フェーズ	31.3 / 55.3 / 47.1	38.7 / 100.4 / 74.7
調整フェーズ	1.09 / 51.9 / 28.2	1.38 / 48.8 / 27.1
合計	39.4 / 151.0 / 106.2	44.4 / 169.8 / 114.6

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.

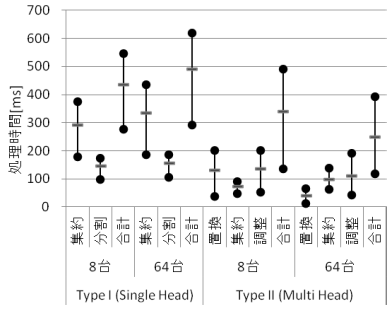
	8 台 [min/max/ave]	64 台 [min/max/ave]
集約フェーズ	178.2 / 374.7 / 291.6	184.9 / 434.3 / 334.7
分割フェーズ	99.0 / 172.2 / 145.0	105.8 / 186.0 / 156.4
合計	277.2 / 546.9 / 436.6	290.7 / 620.3 / 491.1

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	51.4 / 202.4 / 136.1	41.3/ 190.0 / 110.0
合計	130.0 / 491.8 / 338.9	117.5 / 392.4 / 249.0



(a) 物体回転なし



(b) 物体回転あり（X 軸：45 度）

図 13　処理時間の概要
Fig. 13　Execution time.

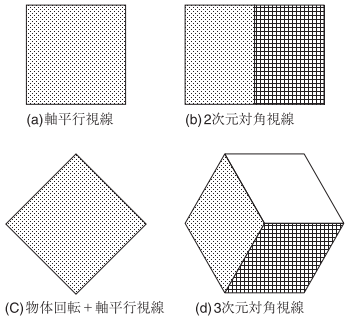
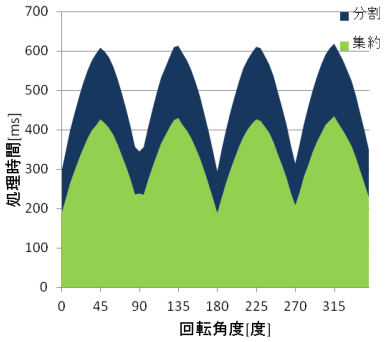
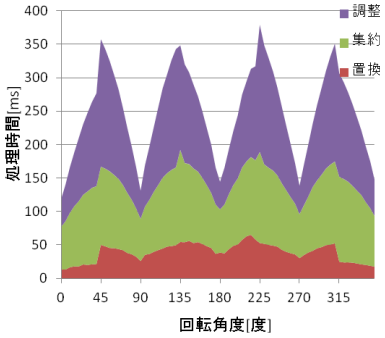


図 14　代表的な視点における投影面の概形
Fig. 14　Typical projection footprint.



(a) Type I (Single Head)



(b) Type II (Multi Head)

図 15　処理時間の視点依存性（64 台構成，X 軸回転：45 度）
Fig. 15　View dependency of execution time.

現する．**図 13** は表 2 から表 5 のデータをグラフ表示したものである．以下の表における処理時間の min，max は 72 方向のうち最も処理時間が最小ならびに最大となる角度での処理時間，avg は 76 方向すべての平均処理時間である．

それぞれの軸回りの回転では，回転角によって描画ならびに合成に要する時間が変動する [15], [16]．変動の主要因は表示される最終画像の投影面積（**図 14** 参照）の変動であり，次が置換フェーズと調整フェーズにおける視点依存の処理，そしてメモリアクセスパターンの違いに起因する実効メモリアクセス速度の変動である．**図 15** は AS を 64 台使用時で X45 の状態におけるそれぞれの視点位置での処理時間とその内訳を示した図である．いずれの視点位置，物体回転角度に対しても本論文で提案する Type II (Multi Head) で合成処理時間が短縮されていることが分かる．以下，個々の状況に対する結果の解析を行う．

まず図 15 から，全体的な傾向として 45 度ごとに極大，極小が交互に繰り返して現れることが分かる．これは投影面積の変動と対応しており，物体回転なし（X0）の場合もほぼ同様の傾向となる．図 15 (a) では 180 度を中心に左右対称の傾向が，0 度から 180 度の範囲に注目すると，90 度を中心に再び左右対称の傾向が見られる．これは，視線ベクトルの各成分の絶対値の大小関係が 90 度ごとに切り替わることに対応している．一方，図 (b) ではさらに 45 度ごとに傾向が変わっていることが観測できる．これは置換フェーズで必要となる処理が切り替わるためである．Type I (Single Head) の場合にも，Type II (Multi Head) の調整フェーズと同等の処理が必要であるが，Type I においては最終画像が集約された 1 ノード内でのメモリアクセ

スパターンの変更のみ（メモリアクセス時の配列添字の変更のみ）で等価な処理が完了するため処理時間の差（平均メモリアクセス時間の変動）は分割のための通信時間に完全に隠蔽されている．なお，表 2 から表 5 における合計時間の最大，最小はそれぞれのフェーズにおける最大値あるいは最小値の合計であり，必ずしもこれらが同じ視点位置で発生しているとは限らない．そのため各視点位置での実測値での最大値，最小値とはわずかに異なっている．

Type I，Type II ともに存在する集約フェーズに注目すると，木構造合成処理での通信量の多い終段の処理を省略できる Type II は処理時間が Type I の約 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 台での集約時間が倍以上になっているが，これは前者ではサイズが $1{,}024^2$ 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．

Type I での分割フェーズについてみると，AS の台数によらずほぼ同程度の数値である（表 2，表 4 参照）．物体回転角度の違いによる差は投影面積の差が主要因であるが，合成過程における重複度が低下することによる通信データ量の影響も受けている．その結果，最終的な投影面積が等しい状態（集約処理時間が min となる視点位置）においても X45 の場合，X0 の場合に比べて多くの処理時間を費している．

Type II の置換フェーズでは 8 台と 64 台で比較すると，それぞれの AS が転送する画像サイズはノード数が N 倍になると $N^{-\frac{2}{3}}$ 倍となるためこのフェーズに要する時間は理論上は 1/4 となる．X45 の場合で画像サイズが最大となる視点に対してはこの傾向が観測できるが，平均でみると 2〜3 倍程度の速度向上にしかなっておらず，画像サイズが小さい場合には通信・同期オーバヘッドの影響が見えているものと考える（表 3，表 5 参照）．

Type II の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できた．しかし，X45 の場合は視線方向によっては集約フェーズよりも処理時間が長くなった．これは Type I における集約フェーズの最終段階での処理が Type II では調整フェーズに移るためであり，特に視線方向が物体の 3 次元対角方向と一致する場合にこの影響が最大となることが確認できた（表 5，図 15 (b) 参照）．

5.3　考察

まず最初に，画像生成からタイルドディスプレイへの表示までのトータルなシステムで考えた考察を行う．4 章の図 7 で示したようなパイプラインステージを考えた場合，AS と DS が物理的に非常に離れた場所にある場合や AS–DS 間で十分なネットワーク帯域が得られない場合は，DS 側の処理がパイプラインピッチを規定することとなる．今回の実装では，AS と DS 間のネットワークが AS 内や DS 内のネットワークとほぼ同程度の性能を持っており画像転送時間自体は，ほとんどの視点において処理全体の大きなボトルネックになっていなかった．実際には AS–DS 間の転送時間は約 35 ms であり，バッファリング時間は約 13 ms であった．また，GPU での画像生成速度は視点によって変動するが 50〜100 ms であった．使用した GPU がすでに旧世代のものであり，現在流通している最新の GPU を 64 台使用できればこの時間を 1/10 以下に低減させることが可能である．さらに，近年のマルチコアプロセッサの利用を前提とすれば，合成処理の複数ステージ化，バッファリング処理の別ステージ化によりさらなる高速化が期待できる．しかしながら，使用した CPU が旧世代のシングルコア CPU であるため，合成処理の複ステージ化は行っていない．ただし，合成処理における処理時間の大半は通信時間であり CPU の利用率は 10〜20%程度であるため，バッファリング処理は別ステージとして容易に実装可能である．

詳細解析は別論文 [17] に譲るが，典型的な視線方向に対して合成処理フェーズで必要となる通信量だけに注目して提案アルゴリズムを理論的に解析したものを付録に添付する．AS の台数として 512 台までの考察を行っているがプロセッサ台数を増やした場合にも理論上は合成時間増加を抑えることが可能であることが分かった．今回の実験結果では理論値ほどは高速化できていないが，AS を 8 台から 64 台に増加してもほとんどの視点位置において処理時間の増加を抑えることができている．理論値からのずれの要因としては，計算時間が完全には無視できない点，シングルコアの CPU で複数ステージのパイプライン処理を行わせている点，同期処理のコストが無視されている点などが考えられる．

次にさらなる高解像度モニタを実現する場合の合成処理時間について考察を行う．DS の数を増やさずに個々のモニタの解像度を上げた場合は，合成処理時間は基本的には今回の実験データを基に外挿が可能である．一方で，解像度を上げるために DS の数を増やすと，集約ステージ（狭義の合成ステージ）の段数が減少し，その分調整ステージの処理が増加する．その結果，今回提案したアルゴリズムそのままでは調整ステージの処理が増加すると大きなサイズの画像授受が増加し性能低下が発生する．この問題に対しては，アルゴリズムがハードウェアに要求するネットワークバンド幅やトポロジの制約が強くなるという欠点はあるものの，集約ステージにおける計算順序の入れ換えにより調整ステージで必要となる通信量を低減させるアルゴリズムを検討しており，別論文として提案する予定である．

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．マルチヘッド主軸優先木構造合成アルゴリズムの採用により，木構造合成アルゴリズムの欠点である合成ステージ後段での通信量の増加が抑制でき，それとともにDSへの並列転送による転送時間も軽減され，両者の相乗効果が得られた．また，タイルドディスプレイシステムとして広く用いられているSAGEとの連携を考慮した中間画像置換フェーズの導入により，大容量なボリュームデータの再配置なしで任意視点からの高精細ボリュームレンダリングを可能とするフレームワークの構築に成功した．その結果，ボリュームレンダリング結果としての$2{,}048^2$サイズの画像を4台構成のタイルドディスプレイを利用し2.5〜20 fpsの速度で任意視点から表示することが可能となった．

今後は，AS内のネットワークトポロジを考慮した合成アルゴリズムの最適化 [5] ならびに合成ステージ自体のパイプライン化によるさらなる高速化を行っていく予定である．

謝辞　本研究は一部，科学研究費補助金（基盤（S）16100001ならびに基盤（C）22500044）の補助による．本研究の実施にあたり活発な議論に参加いただいた森・福間研究室各位に感謝いたします．また，日頃より各種クラスタ環境の運営・整備に貢献いただいている松山幸雄技術職員なしでは本研究は実施できなかった．松山氏に心より感謝いたします．

参考文献

[1] Yu, H, Wnag, C., Grout, R.W., Chen, J.H. and Ma, K.-L.: In Situ Visualization for Large-Scale Combustion Simulations, *IEEE Trans. Computer Graphics and Applications*, Vol.30, No.3, pp.45–57 (2010).

[2] Ma, K.-L., Painter, J.S., Hansen, C.D. and Krogh, M.F.: Parallel volume rendering using binary-swap compositing, *IEEE Trans. Computer Graphics and Applications*, Vol.14, No.4, pp.59–68 (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] Asano, T., Yoshimura, T., Shimada, H., Mori, S. and Tomita, S.: Large Scale Volume Rendering on the Sensable Simulation System, *Int'l Workshop on Super Visualization* (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).

[6] VisWall High Resolution Display Wall, available from ⟨http://www.visbox.com/wallMain.html⟩ (accessed 2012-03-05).

[7] Humphreys, G., Houston, M., Ng, R., Frank, R., Ahern, S., Kirchner, P.D. and Klosowski, J.T.: Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters, *Proc. ACM SIGGRAPH'02*, pp.693–702 (2002).

[8] Jeong, B., Renambot, L., Jagodic, R., Singh, R., Aguilera, J., Johnson, A. and Leigh, J.: High-Performance Dynamic Graphics Streaming for Scalable Adaptive Graphics Environment, *Proc. ACM/IEEE Supercomputing Conference*, DOI: 10.1145/1188455.1188568 (2006).

[9] Distributed Multihead X (DMX) Project, available from ⟨http://dmx.sourceforge.net⟩ (accessed 2012-03-05).

[10] Sandstrom, T.A., Henze, C. and 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, available from ⟨http://viz.aset.psu.edu/ga5in/DisplayWall.html⟩ (accessed 2012-03-05).

[12] Schwarz, N., Venkataraman, S., Renambot, L., Krishnaprasad, N., Vishwanath, V., Leigh, J., Johnson, A., Kent, G. and Nayak, A.: Vol-a-Tile – A Tool for Interactive Exploration of Large Volumetric Data on Scalable Tiled Displays, *Proc. IEEE Visualization*, DOI: 10.1109/VISUAL.2004.126 (2004)

[13] Bethel, E.W. et al.: Sort-first, distributed memory parallel visualization and rendering, *Proc. Parallel Visualization and Graphics*, DOI: 10.1109/PVGS.2003.1249041 (2003).

[14] Allard, J. et al.: A Shader-Based Parallel Rendering Framework, *Proc. IEEE Visualiztion*, pp.127–134 (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).

[16] 丸山悠樹ほか：汎用グラフィクスカードを用いた並列ボリュームレンダリングシステム，情報処理学会ACS論文誌，Vol.45, No.SIG11(ACS7), pp.332–345 (2004).

[17] 坂井陽平：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，福井大学大学院工学研究科修士論文 (2011).

付　　録

A.1　通信データ量の理論的考察

並列ボリュームレンダリングにおいて画像合成にともなうノード間通信は必須であり，処理の中で多くの時間を占めている．したがって，ここでは3.3節で提案したアルゴリズムの通信コストについて検討する．

通信データ量と通信時間が比例関係にあるとすると，3.3節で提案したアルゴリズムの通信コストはDSが最終画像を表示するまでの総通信時間と考えられる．総通信時間は，全体の処理時間からレンダリング時間，合成の演算時間，バッファリング時間を除いた時間である．ここでは，N（$= n^3$）のサブノードによる並列ボリュームレンダリングからM（$= m^2$）台のDSにS^2サイズの解像度で描画する場合を考え通信データ量を求める．なお，通信量は視点によって変化するため，特徴的な視点のみ考えるものとし，ディスプレイ台数が$M = 4$でアプリケーションサーバが$N = 8, 64, 512$の場合を示す．導出式の説明 [17] には多くのページ数が必要なため以下では結果のみを示す．

視線ベクトルが軸平行$(-1, 0, 0)$の場合の総通信量は次式で与えられ，具体的なパラメータを代入した値を**表 A·1**

表 A·1　通信データ量
Table A·1　Communication cost.

視線ベクトル	ノード数	シングルヘッド	マルチヘッド
$(-1,0,0)$	8 台	$1.75S^2$	$0.75S^2$
	64 台	$1.81S^2$	$0.43S^2$
	512 台	$1.78S^2$	$0.18S^2$
$(-1,0,-1)$	8 台	$2.34S^2$	$1.41S^2$
	64 台	$2.42S^2$	$1.32S^2$
	512 台	$2.37S^2$	$1.24S^2$
$(-1,1,-1)$	8 台	$3.05S^2$	$2.20S^2$
	64 台	$3.88S^2$	$1.80S^2$
	512 台	$4.43S^2$	$1.60S^2$

に示す．

$$T_{\mathrm{I-para}} = \log N^{\frac{1}{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{S^2}{N^{\frac{2}{3}}} + \frac{2S^2}{N^{\frac{2}{3}}} + \frac{4S^2}{N^{\frac{2}{3}}} + \cdots + \frac{S^2}{2} + \frac{(M-1)S^2}{M} \tag{A.1}$$

$$T_{\mathrm{II-para}} = \frac{S^2}{N^{\frac{2}{3}}} + \frac{S^2}{N^{\frac{2}{3}}} + \log N^{\frac{1}{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{S^2}{N^{\frac{2}{3}}} + \frac{2S^2}{N^{\frac{2}{3}}} + \frac{4S^2}{N^{\frac{2}{3}}} + \cdots + \frac{2(\log n - \log m)S^2}{N^{\frac{2}{3}}} \tag{A.2}$$

視線ベクトルが 2 次元対角 $(-1,0,-1)$ の場合の総通信量は次式で与えられ，具体的なパラメータを代入した値を表 A·1 に示す．

$$T_{\mathrm{I-2Ddiag}} = (n-1+\log n) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + (n\log n + n - 1) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + 2n(n-1) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{(M-1)S^2}{M} \tag{A.3}$$

$$T_{\mathrm{II-2Ddiag}} = \sqrt{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \sqrt{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + (n-1+\log n) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \left(n(\log n - \log m) + \left(\frac{n}{m} - 1\right)\right) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + 2n\left(\frac{n}{m} - 1\right) \cdot \frac{\sqrt{2}}{2} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{\sqrt{2}}{4} \cdot S^2 \tag{A.4}$$

視線ベクトルが 3 次元対角 $(-1,-1,-1)$ の場合の総通信量は次式で与えられ，具体的なパラメータを代入した値を表 A·1 に示す．

$$T_{\mathrm{I-3Ddiag}} = (2^{p+2} - 2^2 + 2p) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + (2^{2p+2} + p2^{p+1} - 2^{2p+1} - 2) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + (2^{2p+2} - 2^{p+2} + p2^{2p+1}) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{(M-1)S^2}{M} \tag{A.5}$$

$$T_{\mathrm{II-3Ddiag}} = \frac{3}{\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{3}{\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + (2^{p+2} - 2^2 + 2p) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \left(2n\left(\frac{n}{m} - 1\right) + (p-q)2^{p+1} + 2^{p-q+1} - 2)\right) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + n\left(\frac{n}{m} - 1\right) \cdot \frac{1}{2\sqrt{3}} \cdot \frac{S^2}{N^{\frac{2}{3}}} + \frac{25}{16\sqrt{3}} \cdot S^2 \tag{A.6}$$

坂井 陽平　（正会員）

1986 年生．2009 年福井大学工学部情報・メディア工学科卒業．2011 年同大学大学院工学研究科情報・メディア工学専攻修士課程修了．現在，NTT データ北陸勤務．

浅野 琢也　（正会員）

1986 年生．2008 年福井大学工学部情報・メディア工学科卒業．2010 年同大学大学院工学研究科情報・メディア工学専攻修士課程修了．現在，金沢エンジニアリングシステムズ勤務．

福間 慎治

1971 年生．1994 年長岡技術科学大学工学部電子機器工学課程卒業．1996 年同大学大学院工学研究科電子機器工学専攻修士課程修了．1999 年同研究科情報・制御工学専攻博士後期課程修了．博士（工学）．同年長岡技術科学大学工学部助手．2000 年島根大学総合理工学部助手．2004 年福井大学工学部情報・メディア工学科講師．2011 年福井大学大学院工学研究科情報・メディア工学専攻准教授．信号処理ならびに信号処理応用システムの研究に従事．電子情報通信学会，計測自動制御学会，IEEE 各会員．

森 眞一郎　（正会員）

1963 年生．1987 年熊本大学工学部電子工学科卒業．1986 年九州大学大学院総合理工学研究科情報システム学専攻修士課程修了．1992 年同専攻博士後期課程単位取得退学．同年京都大学工学部情報工学科助手．1995 年同助教授．1998 年同大学大学院情報学研究科助教授．2006 年福井大学大学院工学研究科情報・メディア工学専攻教授．博士（工学）．並列処理，可視化，計算機アーキテクチャの研究に従事．電子情報通信学会，IEEE-CS，ACM 各会員．