Vol.2010-HPC-126 No.32
2010/8/4

タイルドディスプレイを用いた高精細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]

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伝送時間を短縮する．また，PC クラスタを用いて実装し，画像伝送時間を含めた描画速度の向上を確認した．

The implementation of the high resolution volume rendering system using tiled display

Yohei Sakai,[†1] Takuya Asano,[†1] Shinji Fukuma[†1]
and Shin-ichiro Mori[†1]

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 alleviate 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，これを汎用計算機を並列に連携して大規模なデータを分割して可視化することを目的とした並列ボリュームレンダリングが提言されている．

また，可視化した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挙げられる．

ここでは，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高精細ボリュームレンダリングシステムを構築する．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伝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伝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. 研究の背景

2.1 関連研究

ここでは，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と，その結果を描画するための大規模な表示装置，さらに，それらを組み合わせた大規模データの可視化システムについて取り上げる．

近年，専用ハードウェアないしは汎用 GPU の採用によるレンダリング時間の大幅な短縮が報告されている．しかしながら，大規模なデータの実時間可視化においてはメモリ不足

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

となるため並列環境が必要である．このような並列環境においては，中間画像の並列画像合成処理の高速化が必須である．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[1] や SLIC[2] 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．計算量のオーダー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．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[16]，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．2.3 節で当該アルゴリズムの詳細を説明する．

大規模かつ複雑なデータを人間が直観的に理解するために，データを画像にして提示する可視化技術は非常に重要であり，これまでに多くの研究が行われてきた．大規模シミュレーションにより出力された画像は非常に大きいため，現在広く使われている液晶ディスプレイの解像度では，縮小やスクロールしないと見えない部分があり，画像全体を認識するのが困難で分析の妨げとなる．プロジェクタなどで拡大した場合は，画像全体は見渡せるものの，画素数は上がらないため，細かい部分が見づらくなるという問題が起こる．そこで，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 または，Myrinet などで相互接続したものであり[3]，コストが低く，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[4] とディスプレイマネージャレベルでの実装法[6] 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[4] と SAGE[5] 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に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マスター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[7]，VisWall[3]，LionEyes Display Wall[8] 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マスターノードから全てのディスプレイノードに送信するためマスターノードの通信帯域幅に依存する．また，複数のアプリケーションの表示，ウィンドウの移動，拡大・縮小にはディスプレイマネージャレベルの DMX を組み合わせて利用するか，あるいはアプリケーションの再コンパイルが必要となる．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広域ネットワーク (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 を利用することとした．SAGE の詳細については次節で説明する．

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[9] 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 が利用されている．グラフィックス API レベルのシステムであり，レンダリングノードから API による命令で画像データをタイルドディスプレイへ送信する．その際，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．

2.2 SAGE(Scalable Adaptive Graphics Environment)

SAGE はイリノイ大学で開発された，高解像度の画像やビデオを遠隔地にネットワークを介して，リアルタイムに表示させることができるソフトウェアである．SAGE を利用することによって，複数の視覚化アプリケーションをタイルドディスプレイに同時に表示し，見ることが出来る．また，アプリケーションウィンドウは標準のウィンドウのようにサイズの変更や移動，重ねて表示する事が容易に出来る．SAGE は，FreeSpace Manager，SAGE Application Interface Library(SAIL)，SAGE Receiver，SAGE UI から構成されている．FreeSpace Manager は SAGE を管理するマネージャで，ここから SAGE 制御メッセージを他の構成要素に送ることで SAGE 上に流れるピクセルを制御している．SAIL はアプリケーションを SAGE 上で表示可能にするためのライブラリで，アプリケーションに SAIL の API コードを組み込むことで，SAGE 上で表示可能な SAIL アプリケーション (ノード) となる．SAGE Receiver はアプリケーションのピクセルデータを SAIL ノードから受け取り，タイルドディスプレイに表示する．SAGE UI はタイルドディスプレイに表示されているアプリケーションウィンドウの表示位置やサイズを自由に変更することができる GUI である．ローカルで実行されているものだけでなく，遠隔地で実行されているアプリケーションも SAGE UI を利用できるユーザならば誰でもディスプレイ上の表示に関する操作を行うことができる．

2.3 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[17]．

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 1 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 方向と対応させると，あるサブボリューム V_o に隣接するノードは 3 個である．V_o の座標を (X,Y,Z) とすると，隣接するサブボリュームの座標は ($X \pm 1$,Y,Z),(X,$Y \pm 1$,Z), (X,Y,$Z \pm 1$) となる．視線ベクトルを (x,y,z) と表すとする．

サブボリューム V_o について，Z 方向に隣接するサブボリューム V_z の投影される領域と，V_o の投影される領域が重なる部分領域の大きさは，視線ベクトルの z 成分の絶対値の大きさに依存する．例えば，視線ベクトルが (0,0,-1) のとき，どのサブボリュームの投影領域も，x,y 方向に隣接するの投影領域とは全く重ならない．しかし，Z 方向に隣接するサブボリュームの投影領域を考えると，互いに完全に重なっている (図 2)．視線ベクトルの各成分の絶対値が大きいほど，その方向に隣接するサブボリュームの投影領域の重複部分が大きくなる．

このように視線ベクトルの主軸の優先順位に基づいて合成ペアを動的に選択することで，選択された 2 つのサブボリュームに対する中間画像の重複部分を最も大きくすることができる．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．

3. タイルドディスプレイ向け合成アルゴリズムの検討

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）を連携させた高精細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．前述のとおり単純に AS と DS を連携させただけでは，AS と DS 間の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 (図 3 参照)．そこで, 本章では 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．

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

Vol.2010-HPC-126 No.32
2010/8/4

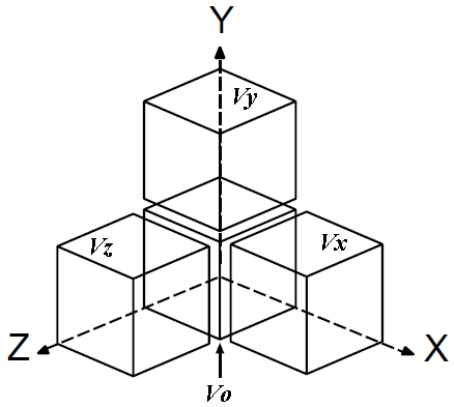

図 1 隣接するサブボリュームと軸対応
Fig. 1 Adjacent volume data and axes.

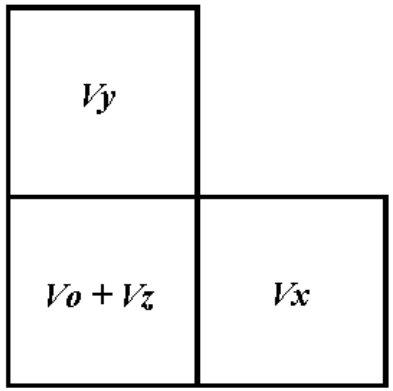

図 2 視線ベクトル (0,0,-1) の場合のサブボリュームの重複関係
Fig. 2 The degree of overlapping between sub-volumes in the case of view vector (0,0,-1).

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．

この時，AS と DS を連携させる基本方針として以下の 4 つの方針が考えられる．

Type I 主軸優先木構造合成を用いて最終合成画像を作成するが, その画像を M 分割し一旦 AS 内の他の M-1 台のノードに再分散したのち, 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．再分散の時点で 1 対 M の通信が必要となるが, 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 AS-DS 間の通信ボトルネックを解消できる可能性がある．

Type II 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．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．最後に，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．

Type III TypeII と同様に木構造合成を途中で中断し M 台のノードに中間合成画像を集約する．この時点で AS 内の M 台のノードから DS の M 台のノードに 1 対 1 で中間合成画像を送信する．TypeII と同様に木構造合成を途中で中断したため，DS に送られた画像は，未だ最終合成画像ではなく，かつ, DS での表示位置と必ずしも一致していない．そのため，最終合成処理と DS 内での位置合わせの処理を DS 内で実施する．

Type IV AS で木構造合成をまったく行わず, AS の各ノードが描画した中間画像を，それが DS のどのノードに対応するかを考慮しながら直接 DS へ送信し，DS 側で送られてきた画像の前後関係を維持しつつ画像合成を行う方法である．従来の DirectSend 型の画像合成アルゴリズムをマルチスクリーン向けに修正したものに相当する．

TypeI は，実装がシンプルで AS 内通信性能が AS-DS 間の通信性能に比べて高い場合に有効であるが，一旦合成したデータを再分散する過程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，その後の補正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．さらに，木構造合成処理を構成する処理ステージのうち，通信量の多い終段の処理を省略できることで合成処理に要する時間を削減できるという利点がある．次に，TypeIII について考えると，TypeII と比べて AS の処理を一部 DS に移すことで AS の負荷を軽減できるという利点があるが，AS が DS に送る画像データ量は必ずしも最小ではない．また，DS 側で AS 側のアプリケーションに依存した処理を行う必要があり，AS と DS を独立に設計することができなくなる．すなわち SAGE を利用することができなくなる．次に TypeIV について考える．通信スループットの最大は DS 側の M 台のノードのスループットの合計が上限となるが, AS 側の N 台のノードが DS 側の M 台に対してほぼ同時期に通信を開始するため，AS 側の送信タイミングを十分に制御できなければ輻輳が発生しスループットを低減させる可能性がある．また，AS と DS の間で受け渡しが行われるデータ量は，TypeI,II,III と比べて最大で $N^{\frac{1}{3}}$ 倍となる．なお, AS と DS が

物理的に非常に離れており，AS の 1 ノードと DS の 1 ノードとの間の実効スループットが DS の 1 ノードの物理的な最大スループットに比べて極めて低い場合，AS の複数ノードと DS の 1 ノードが通信を行うことで DS 側でのトータルなスループットが向上する可能性があるが，今回我々が想定する環境では AS と DS の距離は同一の建屋内程度を想定しており，この状況は発生しないものとする．

これらの点を鑑み，我々のシステムでは TypeII の採用を決定した．

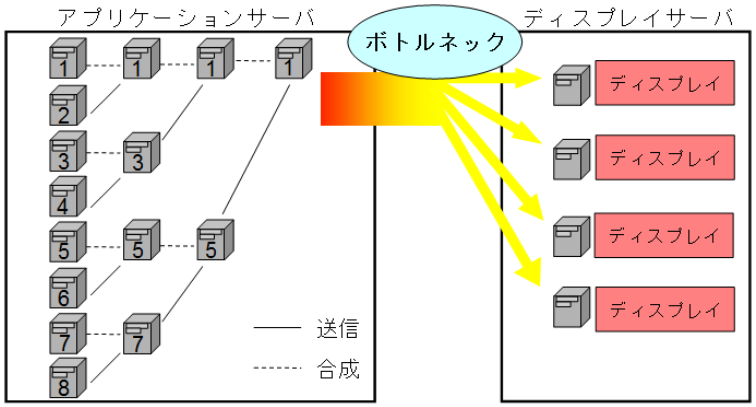

図 3 可視化のボトルネック
Fig. 3 Bottleneck of visualization in tiled display.

4. タイルドディスプレイ向け主軸優先並列木構造合成

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，$n\geq m$ であり，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．

提言するアルゴリズムは N 台のノードで生成された中間画像を M 台のノードに集約するフェーズと，集約された M 台の画像間の重なりを解消するための交換フェーズで構成する．以下，各々について説明する．

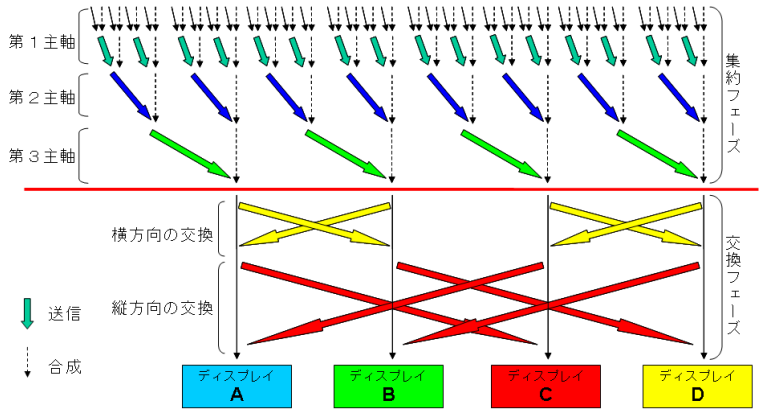

図 4 64 台での合成の流れ
Fig. 4 A flow of the composition in case of 64.

4.1 集約フェーズ

集約フェーズでは，N 台のノードで生成された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 5 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，$(i,j,2^sk)$ と $(i,j,2^sk+2^{s-1})$ 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．

Vol.2010-HPC-126 No.32
2010/8/4

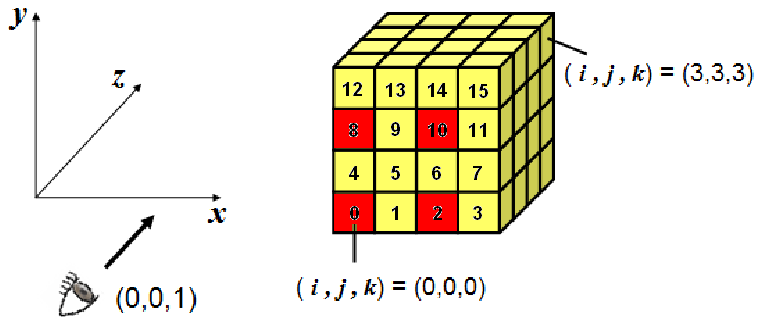

図 5 視線方向と 64 台のノード配置
Fig. 5 View vector and 64 node placement.

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 5 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．しかし，一般に最後に集約されるノードは，主軸の優先順位によって変化する．第 1 主軸が x 方向の場合 $(p\frac{n}{m} \times n + q\frac{n}{m} \times n^2)$，$y$ 方向の場合 $(p\frac{n}{m} + q\frac{n}{m} \times n^2)$，$z$ 方向の場合 $(p\frac{n}{m} + q\frac{n}{m} \times n)$ となる．ただし $p = 0,1,...,m-1$，$q = 0,1,...,m-1$ とする．

4.2 交換フェーズ

このフェーズでは以下の 3 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．最後は SAGE を利用するために発生する制約条件である．以下では，まず最初の 2 つの問題を解決する方法を説明する．

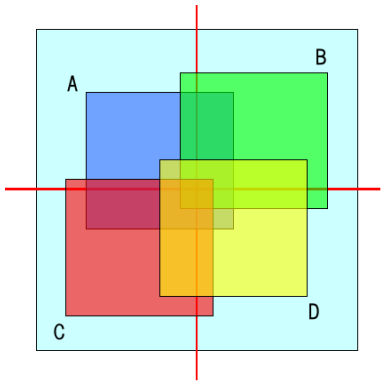

図 6 ディスプレイと画像の位置関係
Fig. 6 The position of an images and the displays.

図 6 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．

まず横方向の隣接ノードと画像の交換を行う．図 7 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．

一般には $2i$ と $2i+1$ $(i = 0,1,2,...,\frac{m}{2} - 1)$ のノードで交換し，次に，$2i+1$ と $2(i+1)$

Vol.2010-HPC-126 No.32
2010/8/4

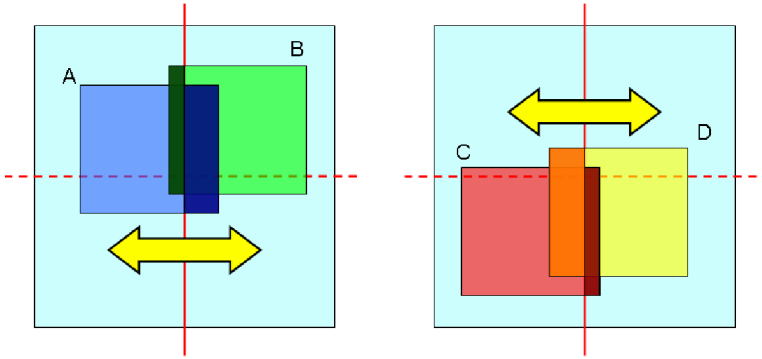

図 7 横方向の交換
Fig. 7 The swap of the horizontal direction.

$(i = 0, 1, 2, ..., \frac{m}{2} - 2)$ のノードで交換する．

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．

一般には $2j$ と $2j+1$ $(j = 0, 1, 2, ..., \frac{m}{2} - 1)$ のノードで交換し，次に，$2j+1$ と $2(j+1)$ $(j = 0, 1, 2, ..., \frac{m}{2} - 2)$ のノードで交換する．この 2 段階の交換により集約された M 台間での画像の重なりはすべて解消され最終合成画像となる．

最後に SAGE を利用する際の制約条件の充足するための補正処理を行う．現在の SAGE では，ディスプレイサーバ (DS) の各ノードが表示を担当する画素に対して，画像データを送信するアプリケーションサーバ内のノードを 1 台だけ一意に定めなければならない．そのため現在の実装では，前節の集約フェーズにおいて第 1 主軸が z 方向の場合に画像が集約されるノード M 台 $((p\frac{n}{m} + q\frac{n}{m} \times n)$ 番目のノード．ただし $p = 0, 1, ..., m-1$，$q = 0, 1, ..., m-1)$ と DS のノードを 1 対 1 に対応付けている．ところが，これらのノードは第 1 主軸が x 方向や y 方向の場合に画像が集約されるノードと異なるため，そのままでは合成画像を DS へ送ることができない状況が発生する．そこで，最終合成が終了した画像を DS と対応付けられたノードに一旦転送したのち，DS へ転送するという処理が必要となる．

5. 実　　装

5.1 実行環境

実行環境を図 8，表 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，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用ノード 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，MPI 通信ライブラリを用いた．

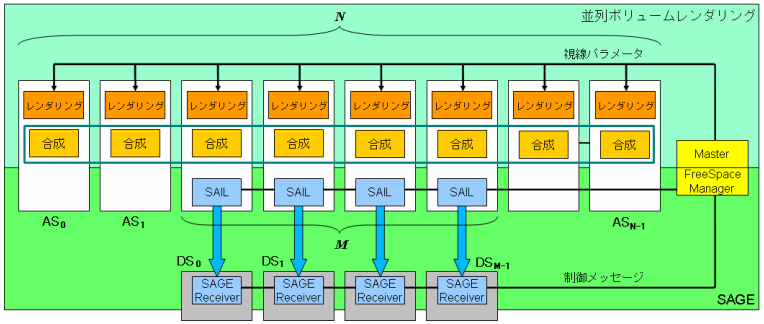

図 8 実行環境
Fig. 8 Processing components.

表 1 各サーバの環境
Table 1 Hardware specification of cluster.

サーバ	アプリケーションサーバ	ディスプレイサーバ
CPU	Core2Duo E6700 2.66GHz	Pentium4 3.00GHz
Memory	2GB	1GB
GPU Card	GeForce 8800 GTX	GeForce FX 5950 Ultra
GPU Memory	768MB	256MB
OS	Fedora Core 6	Fedora Core 6
Network	1GbE	1GbE

ディスプレイサーバへの画像送信には SAGE を用いた．アプリケーションを SAGE に対応させるために，プログラム内でアプリケーションサーバとディスプレイサーバの間で通信

を行うための sail config オブジェクトを設定する．設定は，タイルドディスプレイに表示するノード数，タイルドディスプレイに表示する際の縦横の解像度，送信するピクセルのフォーマット，ディスプレイの表示領域の指定などを行う．

また，タイルドディスプレイに送信するノードは RGB バッファを用意する．このバッファに，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される．今回の実装では，全ノードで同期を取り一斉に描画される．

改良した手法との描画速度を比較するために，以下の 2 種類を実装した．

1 つは合成アルゴリズムに主軸優先木構造合成を用いて，1 台に集約してノード 1 台から 4 台のディスプレイサーバに送信する．通信帯域幅は送信したノード 1 台分となる．もう 1 つは，4 章で提案したタイルドディスプレイ向け合成アルゴリズムを適用する．この場合，4 台のアプリケーションサーバに画像が集約し，4 台のディスプレイサーバに並列に送信する．1 台での送信と比べて通信帯域幅は単純に 4 倍になる．

ボリュームデータサイズは 512×512×512 で，生成する最終合成画像サイズは 1024×1024 と 2048×2048 とした．ディスプレイの枠幅は考慮しないため，分割した画像は各ディスプレイに 512×512(1024×1024 の場合) もしくは 1024×1024(2048×2048 の場合) で表示される．

5.2 実験結果

各画像サイズに対して処理時間内訳と描画速度を計測した結果を表 2，表 3 に．特徴的な視点位置における合成時間を測定した結果を表 4 に示す．

表 2 および表 3 は y 軸に沿って-90 度から+90 度まで 1 度ずつ回転した場合に，各ステップでの処理時間を個別に計測し平均をとったものと，180 度分の回転に要した時間から平均描画速度を求めたものである．

表 2 1024×1024 の各平均処理時間 [msec]
Table 2 Each average processing time for case of 1024^2 resolution.

	1 台で送信	4 台で送信
レンダリング	10.9	10.7
合成	42.6	27.0
画像伝送	10.7	4.3
処理全体の描画速度 [fps]	15.5	23.8

表 3 2048×2048 の各平均処理時間 [msec]
Table 3 Each average processing time for case of 2048^2 resolution.

	1 台で送信	4 台で送信
レンダリング	28.7	30.1
合成	158.7	105.3
画像伝送	41.7	13.3
処理全体の描画速度 [fps]	4.36	6.72

表 4 2048×2048 の指定した角度の合成処理時間 [msec]
Table 4 Composition processing time in a focus of angle.

y 軸について回転した角度	1 台で送信	4 台で送信
0 °	116.0	32.8
44 °	184.2	109.1
45 °	183.8	183.3
90 °	116.4	109.3

まず，画像伝送速度に関しては，スクリーンサイズ 1024×1024 の場合は約 2.5 倍，2048×2048 の場合は約 3.1 倍の性能向上が得られており AS の複数ノードから画像を送信する効果が確認できる．また，スクリーンサイズが大きいほどその効果が高いことが確認できた．合成時間についても合成アルゴリズムを改良した今回の実装の方が処理時間が短くなっており，木構造合成における終段の処理を省略できた効果が確認できた．しかしながら，合成時間に関しては回転中に第 1 主軸が変動するため，4.2 節で述べた修正処理の影響が含まれている．そこで，主軸が x 方向となる回転角 45 度，90 度ならびに主軸が z 方向となり修正処理が不要な 0 度と 44 度での合成処理時間を表 4 に基づき以下で比較検証する．

回転角が 0 度および 44 度の場合は 4 台で送信する場合にも純粋な画像合成時間のみであり，主軸優先木構造合成により最終段まで合成した場合に比べて大幅な速度改善が得られていることがわかる．一方で，45 度および 90 度では，補正処理の影響で速度改善の効果が激減している．特に 44 度と 45 度のデータを比較すると，この両者では視点移動がわずかであるため純粋な画像合成時間はほぼ等しいにも関わらず，補正処理に約 80ms の時間を要していることが分かる．同様に回転角 0 度と 90 度でも，修正処理が必要な点を除けば，合成処理の時間はほぼ同一である．これは 1 台で送信する場合の合成時間がほぼ等しいことからも確認できる．ここでも，修正処理に 80ms 弱の時間を要していることが確認できた．

この 80ms という時間について考えると，これは 1024×1024 の画像を 1GbE の回線で 2 回送る時間に匹敵する．これは 4.2 節で説明した補正処理に必要なデータ量の倍に相当する．その原因を考察した結果，今回の実験環境に起因する問題であることが確認できた．具体的には．今回の実験環境では AS のノード数が 8 と少なかったために，主軸 x の場合に画像が集約されるノードと主軸 z の場合に画像が集約されるノード（すなわち DS への画像転送を行うノード）に重なりが発生し，当該ノードにおいて自ノードが保持する画像を適切なノードに転送する処理と，他のノードから DS へ送信すべきデータを受け取る処理の 2 つが逐次的に処理されたためであることがわかった．より大規模な AS を利用した場合にはこ

の問題を回避することが可能であり，その場合には修正処理を伴う画像合成時間が約 40ms 短縮されることが推測できる．

5.3 考　　察

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もっており画像伝送時間自体は，処理全体のボトルネックになっていなかった．そのため，伝送時間の改善が全体の処理速度の改善に十分に寄与できなかった．しかしながら，合成処理を複数のステージに分割しパイプライン処理する等の最適化を行うと[18] 画像伝送時間もパイプライン周期を決定する要因となる可能性がある．実際に，合成処理をレンダリング時間と同程度のパイプラインピッチでパイプライン化できたとすると，1 台で送信した場合には画像伝送速度がボトルネックとなることが分かる．

次に，合成処理における補正処理の影響を軽減する手法について考察する．この処理は，最終画像を M 台のノードに集約する処理とは本質的に独立な処理である．そこで，この処理を本質的な合成処理と分割し，前述のパイプライン化を行うことで，実質的にこの処理の影響を軽減することが可能と考えられる．逆に，合成フェーズの通信パターンを修正し最終合成画像がすべての主軸に対して同一のノード群に集約されるアルゴリズムの検討も可能であるが，一部不規則な通信パターンが発生する可能性があり，今後十分な検討が必要である．一方で，この処理を逆に利用して画像表示の自由度をあげる試みも考えられる．具体的には，現在 SAGE の UI では実現ができない生成画像の回転処理を補正処理に組み込むことで一回の転送で両方の問題を解決する手法などを実装する可能性が考えられる．さらに，SAGE 自体の改良が可能であれば 1 画素に対して同一 AS 内の複数のノードを対応付けるとともに，透明度をもった画像表示を可能とすることで，補正処理を不要にできる可能性がある．

6. ま と め

本研究では，可視化技術の 1 つである並列ボリュームレンダリングを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させ，高精細ボリュームレンダリングシステムを実装した．その結果，従来では表示仕切れなかった 2048×2048 サイズの画像をタイルドディスプレイを利用し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伝送する際に時間がかか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，交換フェーズではディスプレイ間の合成を行った．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，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，送信ノード数が 1 台と 4 台の場合で画像伝送時間の測定を行い比較した．結果として通信帯域幅を増加させると，画像伝送時間が短くなることを確認した．しかし，SAGE による送信ノードの制約のため送信ノードと主軸が一致していない場合，合成時間の短縮効果とは別に，画像の移動処理が必要となってしまい大きなオーバーヘッドとなった．今後，集約したノードから直接ディスプレイに送信する手法や送信ノード決定のアルゴリズムを変更することで，合成時間を大きく改善できると考えられる．

参 考 文 献

1) 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.
2) Stomple, 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.
3) VisWall High Resolution Display Wall, http://www.visbox.com/wallMain.html, 2010.
4) 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.
5) 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.
6) Distributed Multihead X (DMX) Project, http://dmx.sourceforge. net, 2010.
7) Sandstrom, T.A.; Henze, C.; Levit, C.; , ”The hyperwall,” Coordinated and Multiple Views in Exploratory Visualization, 2003. Proceedings. International Conference , pp. 124- 133, 15 July 2003.
8) LionEyes Display Wall, http://viz.aset.psu.edu/ga5in/DisplayWall.html, 2010.
9) Schwarz, N.; Venkataraman, S.; Luc Renambot; Krishnaprasad, N.; Vishwanath, V.; Leigh, J.; Johnson, A.; Kent, G.; Nayak, A.; , ”Vol-a-Tile - A Tool for Interactive

Exploration of Large Volumetric Data on Scalable Tiled Displays," Visualization, 2004. IEEE , pp. 19p- 19p, 10-15 Oct. 2004.
10) 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.
11) 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.
12) Paul, B.; Ahern, S.; Bethel, E.W.; Brugger, E.; Cook, R.; Daniel, J.; Lewis, K.; Owen, J.; Southard, D.; , "Chromium Renderserver: Scalable and Open Remote Rendering Infrastructure," Visualization and Computer Graphics, IEEE Transactions on , vol.14, no.3, pp.627-639, May-June 2008.
13) Tao Ni; Schmidt, G.S.; Staadt, O.G.; Livingston, M.A.; Ball, R.; May, R.; , "A Survey of Large High-Resolution Display Technologies, Techniques, and Applications," Virtual Reality Conference, 2006 , vol., no., pp. 223- 236, 25-29 March 2006.
14) Fout, N.; Kwan-Liu Ma; , "Transform Coding for Hardware-accelerated Volume Rendering," Visualization and Computer Graphics, IEEE Transactions on , vol.13, no.6, pp.1600-1607, Nov.-Dec. 2007.
15) Xingfu Wu; Taylor, V.; , "Performance Analysis of Parallel Visualization Applications and Scientific Applications on an Optical Grid," Cyberworlds, 2008 International Conference.
16) 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.
17) 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.
18) 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.