情報処理学会グラフィクスとCAD研究報告, 2007-CG-128, Vol.2007, No.84, pp.65-70 (2007).

グリッドコンピューティングを用いた大規模ボリュームデータの可視化手法に関する研究

檜垣 徹† 岡部 憲史†‡ 西橋 邦彦† 玉木 徹† 金田 和文†

† 広島大学 ‡ GEヘルスケア

近年，高解像度化の進むCTやMRIから出力されるボリュームデータを，高精度にかつ高速に可視化するため，本研究ではグリッドコンピューティングを用いてボリュームレンダリングを行う手法を提案する．医療施設や研究機関に多数導入されている計算機を計算資源とするグリッドコンピューティングを用いる．計算能力の不均一な環境下において，可視性に基づき動的にタスクの投入を行うための手法を提案し，シミュレーションによりその有用性を確認した．提案手法をインプリメントし，グリッドコンピューティングを用いて大規模ボリュームデータのレンダリングを行った．

Visualization Technique based on Grid Computing for Large-scale Volume-data

Toru Higaki†, Kenji Okabe†‡, Kunihiko Nishihashi†,

Toru Tamaki†, and Kazufumi Kaneda†.

† Hiroshima University, ‡ GE healthcare.

To visualize high-resolution volumedata acquired from a recent CT or MRI, we propose a method for rendering the large-scale volume data using a grid computing. We use existing computers with non-homogeneous computing power in medical institutions. We propose a method of a dynamic scheduling for submitting tasks to agent machines based on the visibility of divided volume data in a grid computing environment. Simulation results demonstrate the usefulness of the propose method. A large scale volumedata is rendered using our grid computing system.

1. はじめに

今日，医療現場においてCT (Computed Tomography)やMRI (Magnetic Resonance Imaging)などを用いた画像診断が盛んに行われている．しかし，CTやMRIが出力するデータは一般的に断面画像であり，これを直接観察することで正確な診断を行うのはかなりの経験が必要となる．そこで連続的に撮影された断面画像を3次元データ（以降「ボリュームデータ」）として扱い，これをより直感的に観察できるよう画像化するボリュームレンダリング手法[1]が開発された．この手法によって立体的な画像が得られるだけでなく，観察対象の回転を行ったり，対象の内部の表示を行ったりといった観察が可能となった．

ボリュームレンダリングは，主としてレイキャスティング法とスプラッティング法に分類される．そのどちらについても，観察対象となるボリュームデータのサイズや，表示画像の解像度などが大きくなれば非常に計算コストが大きくなる．このためボリュームレンダリングを用いて，大規模なボリュームデータから高精度な画像を作成するには多くの計算時間がかかる．

この問題を解決するため，本研究では分散コンピューティングを用いて計算時間の短縮を行う．医療機関では，多数の計算機が導入されているため，それらを有効利用できる，グリッドコンピューティン

グを用いてボリュームレンダリングを行う．オブスタクルフラグを用いることで，可視性を考慮しつつ動的なタスクの投入を行い，効率よくレンダリングを行う方法を提案する．

2. 関連研究

ボリュームレンダリングを高速化するために，様々な高速化手法が提案されている．高速化手法は，ソフトウェア的アプローチによるものとハードウェア的アプローチによるものの2つの手法に分けられる．

前者として，Early Ray Terminationを用いた高速化が一般的に良く用いられている．また，余傳らによる，“ベクトル量子化を用いたダイレクトボリュームレンダリングの改良”[2]がある．これは，ベクトル量子化を用いてボクセルをグループ化することで，レンダリングの高速化を行っている．

後者としては， GPUなどのグラフィックスハードウェアを用いる方法と，クラスタなどの分散コンピューティングを用いるものがある．Salamaは，一般的なGPUを用いることで高速化する手法[3]を開発した．松井らは，クラスタを用いることでボリュームレンダリングを高速化する手法[4]を開発した．グリッドコンピューティングを用いるものとしては，岩渕らによる“グリッド技術を利用した分散レンダリングシステムの開発”[5]がある．これは，ボリュームレンダリングを分散処理化したものであるが，分散した処理のそれぞれが，前後関係などに依存しないケースでしか用いることができない．

本研究では，後者のアプローチであるグリッドコンピューティングを用いた手法を提案する．新たにオブスタクルフラグを提案し，可視性に基づいた動的なタスクスケジューリングを行うことにより，単純な分割法により得られた前後関係の複雑なボリュームデータ群を効率的にレンダリングする．ここでのグリッドコンピューティングとは一般的にクラスタと呼ばれるシステムとは異なり，様々な計算機が混在する中で未使用の計算資源を利用するシステムのことを指す．

3. グリッドコンピューティングシステム

本研究で提案するグリッドコンピューティングシステムは，ボリュームデータを分割しそれぞれのエージェントでレンダリングする．ボリュームデータの分割は，単純に賽の目状に行うものであり，非常に容易でかつ視線方向に依存しない．しかしそれぞれの分割されたボリュームデータ（以後グループ化カーネル）には前後関係が存在し，レンダリングの際にはこれを考慮する必要がある．

カーネル（またはグループ化カーネル）を投影しフットプリント（またはグループ化フットプリント）を作成するレンダリングを第1段レンダリング，フットプリント（またはグループ化フットプリント）どうしを合成するレンダリングを第2段レンダリングと呼ぶ．

本研究では，図1に示すグリッドシステムを構築した．clientからサーバにレンダリング要求を送信すると，serverではボリュームデータを分割し，グループ化カーネルとして可視性の高い順に各々のagentに送信する．agentでは，グループ化カーネルを第1段レンダリングし，結果であるグループ化フットプリントをserverに返送する．serverでは，返送されたグループ化フットプリントを，可視性の高い順に第2段レンダリングし結果画像を得る．

1節で述べたように，グリッドコンピューティングは計算能力が不均一なマシンで構成されており，かつマシンの利用状況によりレンダリング処理が中断されるため，タスクの結果が返ってくるまでの時間は様々な要因によって変動する．そのため，タスクを送信した順に結果が返って こない可能性を考慮し，以下の点を実現する必要がある．

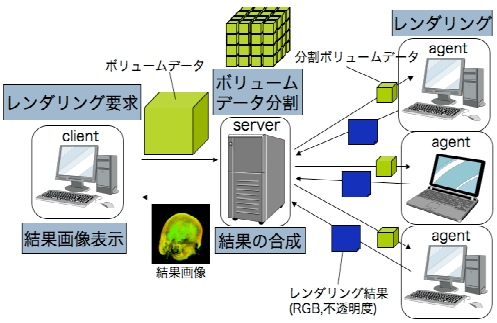


図1: システム構成

1. 可視性の高いグループ化カーネルを，動的にタスクとして送信する．

2. 結果が返ってきたと同時に第2段レンダリングが可能かチェックし，可能でなければ可能となるまで結果をメモリに保持する．

3. 第2段レンダリング可能であるかは可視性に依存し，1つのグループ化カーネルのみにより遮られていたグループ化カーネルは，そのグループ化カーネルが合成されると可視になる．このように可視であるかどうかは動的に変化する．

これらを実現するために，次節でオブスタクルフラグを提案する．

4. オブスタクルフラグ

オブスタクルフラグは可視性を判断するために用いるフラグ群であり，第2段レンダリングが可能かどうかを判断するのに用いる．図2に，グループ化カーネル群と投影面の位置関係を示す．ここでは簡単化のため2次元で説明する．

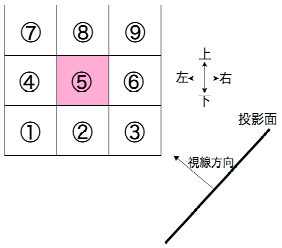


図2: 9個のグループ化カーネルとスクリーン

注目しているグループ化カーネルが第2段レンダリング可能かどうかは，辺隣接の4方向のグループ化カーネルのうち，スクリーン方向について障害となっているものが既に第2段レンダリングされているか否かで判断できる．図2の場合，各グループ化カーネルには4方向に対応したフラグ（以後「オブスタクルフラグ」）が必要となり，障害となりかつ第2段レンダリングされていないグループ化カーネルの存在する方向のフラグを1，それ以外を0とする．

表1: ⑤のオブスタクルフラグ

方向	上	右	下	左
フラグ	0	1	1	0

表1に，⑤のグループ化カーネルのオブスタクルフラグを示す．⑤に対してレンダリングの障害となるのは②と⑥であるため，それに対応する下方向と右方向に対する値が1となる．左と上に関しては，隣接するグループ化カーネルは存在するが，レンダリングの障害とならないため0となる．

オブスタクルフラグは，処理が進む毎に動的に更新される．例えば⑥のグループ化カーネルが第1段・第2段レンダリングされると，⑤の右方向の値は0に書き換わる．このように，随時オブスタクルフラグの更新を行いながらグループ化カーネルが第2段レンダリング可能であるかどうかを判断する．

5. タスクの割り振りと第2段レンダリング

本節ではagentに割り振るタスク，つまりagentに送信するグループ化カーネルを決定する処理について述べる．図3に示す送信キューと呼ぶキュー型のデータ構造を用いる．このキュー型のデータ構造には，オブスタクルフラグが全て0，すなわち第1段レンダリング結果が返ってきたとき，第2段レンダリング可能なグループ化カーネルがセットされている．

agentによる第1段レンダリング処理が終了したとき，もしくは新たなagentがグリッドコンピューティングに参加したとき，serverはagentにタスクとしてグループ化カーネルを送信する．このとき送信キューが参照される．送信キューの要素の中で最も古いもの，つまり最初に追加されたものをタスクとして送信キューから取り出し，agentに送信する．

また第1段レンダリングを行うことができるagentが存在しているにもかかわらず送信キューが空の場合には，第2段レンダリングできる可能性の高いグループ化カーネルをタスクとして送信する．

グループ化フットプリントがagentからserverに返されたとき，第2段レンダリング が可能かどうか判断し，

可能なら第2段レンダリングを行い，そうでない場合はリスト型のデータ構造に格納し，第2段レンダリング可能となるまで一時保存する．

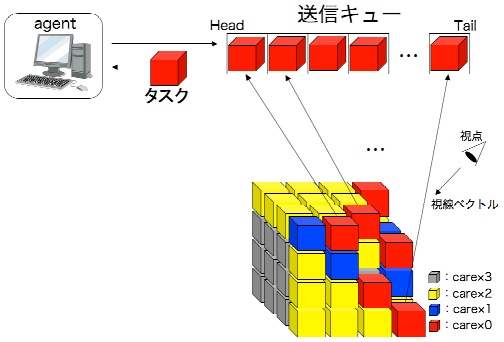


図3: タスク割り振り

agentからグループ化フットプリント が返ってくると，まず合成リストに追加する．その後合成リストを走査し，対応するグループ化カーネルの オブスタクルフラグが全て 0のもの，つまり第 2段レンダリング 可能なものがあればそのグループ化フットプリントを第 2段レンダリングする．

第2段レンダリングが終了すると，オブスタクルフラグの更新を行い，新たにオブスタクルフラグが全て0となるグループ化カーネルを探索し，オブスタクルフラグが全て0でかつ未送信のものがあれば送信キューに追加する．またオブスタクルフラグの更新により，合成リスト内に新たに第2段レンダリング可能になったグループ化カーネルが無いかを探索する．

6. シミュレーション

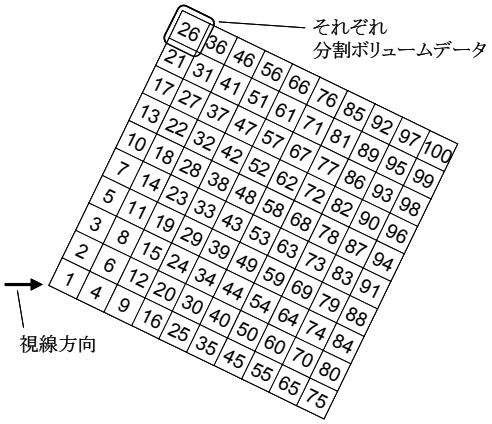
6.1. シミュレーション環境

本論文で提案するオブスタクルフラグ，及び動的なタスクスケジューリングの有用性を検証するため，ダイナミック及びシーケンシャル にタスクスケジューリングした場合の比較 シミュレーションを行った．シミュレーションする環境を表に示す．

表2: シミュレーションの際に仮定した環境

エージェント **計12台**	・高スペックマシン4台 **250sec/task**
	・中スペックマシン4台 **500sec/task**
	・低スペックマシン4台 **1000sec/task**

ただし，ネットワークの転送時間やサーバでの処理時間などは考慮せず，すべて0として取り扱う．また，ボリュームデータの分割は簡単化のために2次元とする（図4）．



（数字は，ボリュームデータの左に視線ベクトルと垂直なスクリーンを考えた時，スクリーンに近い順）

図4: シミュレーションに用いるボリュームデータ

また，エージェントは定期的に第三者により使用され，使用されている間はグリッドのサービスを停止することとする．既に投入されたタスクがあり未完の場合は，処理を一時停止し，第三者の使用が終了するのを待ち，その後再開する．それぞれのエージェントの，第三者による使用スケジュールを図 5に示す．凸状態が，第三者による使用状態を表し，凹状態が，タスク受け入れ可能状態を示す．常時 6台のエージェントがタスク受け入れ可能状態と している．

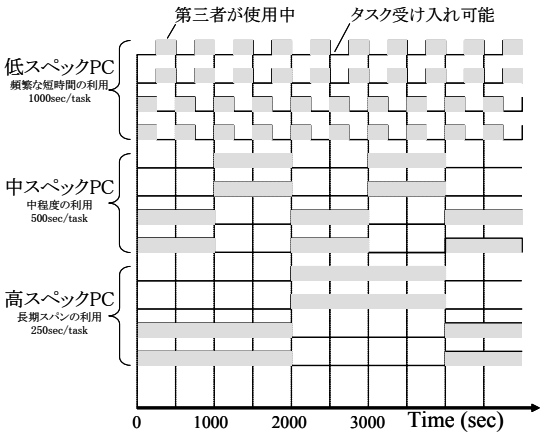


図5: エージェントの使用スケジュール

6.2. タスクの投入条件

ダイナミックタスクスケジューリングとシーケンシャルタスクスケジューリングの場合で以下の条件のもとタスク投入を行うこととする. ただし, 今回のシミュレーションでは, 提案手法の「第1段レンダリングを行うことができるagentが存在しているにもかかわらず送信キューが空の場合」のタスク投入は行わない.

シーケンシャルタスクスケジューリングは以下の条件でタスク投入を行う.

・エージェントがタスク受け入れ可能状態であると.
・視線方向に対して, 自分を覆い隠している分割ボリュームデータがレンダリング済みであること.
・未投入の分割ボリュームデータの中で一番可視性が高いこと(数字の若い順に投入する).

また, 投入されたタスクは, その時点で利用可能なもっともスペックの高いエージェントに振り分けられるとする. これらの条件のもと, 図6のタイムチャートをもとにシミュレーションを行った.

6.3. シミュレーション結果

以上をもとに, シミュレーションを行った結果を表3に示す. また, 空きエージェント利用率とレンダリング進行率の推移を図6, 図7に示す.

表3: 計算時間とエージェント利用率

	Dynamic	Sequential
計算時間	10,000 sec	15,500 sec
エージェント平均利用率	54.2 %	38.4 %

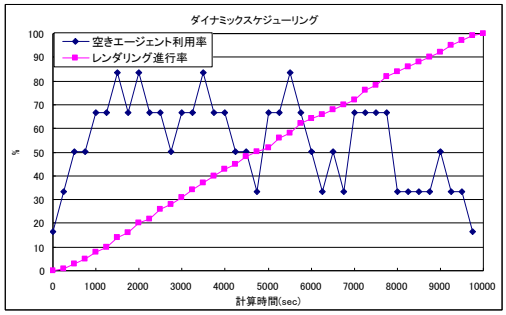


図6: 空きエージェント利用率とレンダリング進行率の推移
(提案手法)

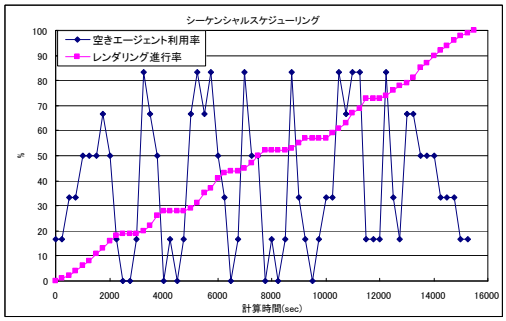


図7: 空きエージェント利用率とレンダリング進行率の推移
(シーケンシャルタスクスケジューリング)

ダイナミックタスクスケジューリングによって, 計算時間を約35%削減できている. これはタスクが実行速度の遅いエージェントに割り振られた際にでも, 処理を中断されること無く効率的にタスクスケジューリングを行った結果であると考えられる. また, 空きエージェント利用率も, ダイナミックのほうが高い数字を示しており, 効率の良いタスクの割り振りが行えている.

表4に, 今回提案したシステムで実際にボリュームレンダリングを行った結果を示す. 台数6台に対して計算時間が1/6になっていないのは, ネットワークオーバーヘッドなどが原因であると考えられる.

表4: 実行時間(実時間)

グリッドコンピューティングを用いて実行した場合※1	1CPUで実行した場合※2
113秒	439秒

※1: PowerPC 2GHz～1.6GHz 6CPU
※2: PowerPC 1.8GHz

7. 終わりに

グリッドコンピューティングを用いた大規模ボリュームデータの効率的なレンダリング手法について提案を行った. 提案手法では, agentの計算能力や利用状況の差異によって計算時間が予測できないというグリッドコンピューティングの特性を考慮し, オブスタクルフラグを用いて可視性を考慮して ダイナミックタスクスケジューリング を行う方法を開発した. また, シミュレーションにより, ダイナミックタスクスケジューリングの有効性を確認した.

今後の課題としては , 実際のグリッドシステム を用いて, 提案手法の有用性の検証などを行うことなどが挙げられる.

参考文献

[1] Robert A.Drebin , Loren Carpenter , Pat Hanrahan, "Volume Rendering", Computer Graphics 22, 4, SIGGRAPH'88, pp.65-74 (1988).

[2] 余傳洋則, 金田和文, 山下英生, "ベクトル量子化を用いたダイレクトボリュームレンダリングの改良, " 電気・情報関連学会中国支部第53回連合大会 講演論文集, p. 436, (2002).

[3] C.Rezk-Salama , "Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Texturesand Multi-Stage Rasterization" In Proc. Workshop on Graphics Hardware, (2000).

[4] 松井学, 伊野文彦, 萩原兼一, "大規模データセットを可視化するための効率の良い並列ボリュームレンダリング ", 論文誌(トランザクション)コンピューティングシステム (ACS), Vol.45, No.SIG11, pp.346-355 (2004).

[5] 岩渕栄太郎, 渡辺出, "グリッド技術を利用した分散レンダリングシステムの開発, " Visual Computing / グラフィクスと CAD 合同シンポジウム 2007 予稿集, pp.181-184 (2007).

[6] Lee Westover , "Interactive Volume Rendering , " Proc. Workshop on Volume Visualization, pp.9-18 (1989).

[7] Marc Levoy, "Efficient ray tracing of volume data, " ACM Trans. on Graphics, Vol. 9, No. 8, pp.245-261 (1990).