修士論文

超高速単方向リンクを用いた
並列ボリュームレンダリングシステム

指導教員　　富田 眞治 教授

京都大学大学院情報学研究科
修士課程通信情報システム専攻専攻

岡村 大

平成18年2月3日

超高速単方向リンクを用いた
並列ボリュームレンダリングシステム

岡村 大

内容梗概

近年の計算機性能の急速な向上に伴い，大規模かつ高精度な数値シミュレーションへの期待が高まっている．中でも実時間のシミュレーションの可視化技術は，大規模な三次元データを必要とする医療などの分野において，高度な技術が要求されている分野である．

我々は，そのような大規模データの可視化を実現する並列ボリュームレンダリング環境の構築に向けた研究を行っている．具体的には 1024^3 のボリュームデータを，1024^2 の投影スクリーンに 15fps で出力することを，当面の大規模データの可視化における目標に掲げている．

本稿ではこの目標の実現へ向けた，専用ハードウェアによる並列ボリュームレンダリングシステムの構成について述べる．専用ハードウェアによる大規模データの可視化では，並列化の方法と並列ノード間のデータの転送方法が問題となる．

並列化した個々のノードにおけるサブボリューム単位での可視化処理について，キューボイド順レイ・キャスティング法の原理を応用した，DRAM のアクセスとの親和性の高いレイ・キャスティング法について実装の検討を行った．

また，高速な単方向リンクを用いた並列ボリュームレンダリングシステムの提案を行った．このシステムは，隣接するサブボリューム同士の合成順序を工夫することで，従来提案してきた ReVolver/C40 や VisA クラスタの特徴であったボリュームデータの三重化する必要がなく，一次元に並ぶ計算ノードによって中間画像の合成が可能であり，重畳方向に関する制約に対する計算と通信方法の工夫することにより単方 向リンクで実現可能である，という特徴を持つ．

さらに，高速な単方向リンクの実装対象として DVI インタフェースを紹介し，このインタフェースを通信路として用いることで，提案するシステムの実装が十分に可 能であることを説明する．

そして，最後に高速な単方向リンクを利用したインタラクティブ性や負荷分散への対応に関し考察を行った．

A Parallel Volume Rendering System implemented with High-Speed Link

Dai OKAMURA

Abstract

The rapid improvement of computer performance in these days makes large-scale numerical simulations possible and very useful in the various fields. Realtime visualization is one of the most suitable use of large-scale numerical simulations, and in a medical field we need higher and higher perfomance of them to visualize the large scale three-demensional medical data.

We research and develop a parallel volume rendering environment to visualize such a large scale data set. Our concrete goal is to visualize 1024^3 data set and to output 15 frames per second on the screen of 1024^2 size.

In this thesis, we propos a new concept of a hardware parallel volume rendering system. When we try to visualize a large scale data set with a hardware parallel volume rendering system, we have to think about how to visualize with parallel calculation hardware and how to transfer the data between the knode.

We introdudce an optimized ray-casting which inspired by the CPU-cache based ray-casting "cuboid". It is friendly with the process of DRAM access. We use it in visualization of voxels in sub-volumes.

And, we introduce a new parallel volume rendering system implemented with high-speed DVI Link. With an idea to composite the result images of neighbor sub-volumes first, our system's parallel calculation knodes ,that stand in a line, can visualize the data set without redundant of volume memory which ReVolver/C40 and VisA cluster have.

We introduce DVI interface which transfer the result images generated by the calculation knode in our systems.

And we consider the way to satisfication of interactive volume rendering and dynamic load balancing.

超高速単方向リンクを用いた
並列ボリュームレンダリングシステム

目次

第1章　はじめに

近年の計算機性能の急速な向上に伴い，大規模かつ高精度な数値シミュレーションへの期待が高まっている．なかでも，実時間数値シミュレーションの可視化技術は，大規模な3次元データの処理を伴う医療などの分野において，現在研究が進められている．このような次世代のシミュレーション環境では，オペレータによるシミュレーション対象へのインタラクティブな操作に対応して，実時間でシミュレーションを行うとともに，即刻その結果を可視化などの手段により提示することが求められる．

処理量の増大に対し単一の計算機またはハードウェアを用いた実装には速度上限界があり，その解決法としては複数台の計算機やハードウェアによる並列処理が有効である．我々は，個人あるいは小規模な組織単位で占有利用可能なPCクラスタを用いて，インタラクティブな数値シミュレーション及びその可視化を実時間処理する環境について研究を行っている．

処理量の増加に伴い生ずる結果表示迄にかかるタイムラグをどのように隠蔽するかという計算面と，即時更新可能でかつ全範囲に高速アクセス可能な三次元データをどのように扱えばよいのかというアーキテクチャ面の双方から可視化の実現可能性を探るため，我々は複数のアプローチをとっている．本稿ではそれらのうち，可視化専用のハードウェアを用いた実現方法について述べる．

1台では実時間処理が不可能な大容量の数値データを分散処理し，再度集約し表示機器に出力するという一連の処理の性質上，高速な伝送路と実時間処理可能な可視化機構が不可欠である．

上記の可視化機構を実現可能なハードウェアアーキテクチャとして，我々はRevolver/C40[1, 2] やVisAを過去に提案し，VisAのプロトタイプとして処理の手順をそのままにボリュームデータの処理可能サイズを半分に制約したVisA Proカードを開発した[3, 4].

VisAアーキテクチャはボリュームレンダリングのひとつの有効な実装方法であるが，ボリュームデータを三重化して保持しているという欠点が存在する。

我々はVisAのアーキテクチャからは一旦離れ，ボリュームレンダリングのレイ・キャスティング法の工夫と，重畳処理の一筆書き処理を用い，視点依存による三重化が不要となる可視化機構を提案する．また，大規模データでのフレームレート向上のために，DVIインタフェースを単方向の高速通信路として

利用した，並列分散処理の高速化について検討した．

本稿では，提案するレンダリングシステムの紹介とシミュレーションにより行った評価について述べる．以下，第 2 章でボリュームレンダリングの概要について説明し，VisA について仕組みとアーキテクチャ面での特徴を紹介する．第 3 章で各種レイ・キャスティング法の説明との今回ハードウェア化・並列化を行う上でサブノード単位の中間画像合成処理に採用した手法について説明する．第 4 章ではサブノード間の重畳合成の仕組みとして，我々が提案するの一次元単方向リンクによる中間画像の合成方法について述べ，5 章でその具体的な実装対象として DVI インタフェースを取り上げ，その通信路としての性質と提案手法の実現について述べる．そして，第 6 章でまとめと，提案システムの拡張についての考察を行う．

第2章　背景

本章では，まずボリュームレンダリングの数値処理の詳細を説明する．そして我々がこれまで手掛けてきたVisAについてその構成と特徴を述べる．

2.1　ボリュームレンダリング

ボリュームレンダリングとは，三次元のスカラー場やベクトル場をボクセルの集合として表現し，二次元平面へ投影することにより，複雑な内部構造や動的特性を可視化する手法である．ボリュームレンダリングは大別して，すべてのサンプル点の寄与を計算して全体を表示する直接法と，前処理によって表示する情報を抽出してデータの一部分を表示する間接法の二種類に分類され，通常ボリュームレンダリングという場合は直接法のことを指す．

直接法のボリュームレンダリングでは，数値シミュレーションなどにより得られた三次元空間上の数値データを，可視化したい情報に重み付けした色と透明度を対応付け，三次元空間内部のデータの布状況を二次元のスクリーンに投影する（図2.1）．(投影の方法には2種類あるが後述する）

処理手順として，まず可視化対象の三次元空間（ボリューム空間と呼ぶ）を小さな立方体格子で区切り（個々の格子をボクセルと呼ぶ），個々の格子の情報に色Cと透明度Tに対応づける．次に可視化する視点の位置を定め，その視点からボリューム空間を眺めたとき投影スクリーン上の各ピクセルを通る視線を計算する．すべての視線上のデータを以下に述べる累積計算を行い，投影スクリーンの1枚分の画像情報として可視化結果を出力する．

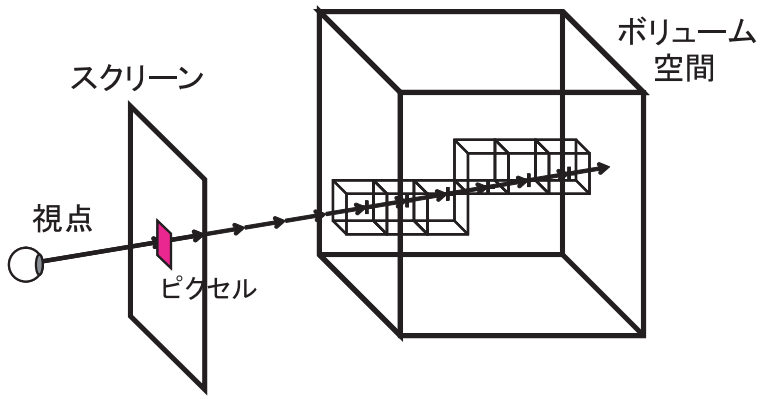


図2.1: ボリュームレンダリングの処理概図

2.1.1 累積計算処理

ボリュームレンダリングで用いる累積計算について説明する．あるピクセルを通る視線上のボクセルの値を視点に近い順に $v_0, v_1, v_2, \cdots$ としたき，ピクセル値は次の式（畳み込み演算）で計算される．

$$C_k = \sum_{i=0}^{k}(1 - t(v_i)) \cdot c(v_i) \cdot \prod_{j=0}^{i-1} t(v_i) \tag{2.1}$$

ここで $c(v_i), t(v_i)$ はそれぞれボクセル値 v_i を，色，透明度に変換して値であることを示している．さらに，式 (2.1) は以下のような漸化式で表される．

$$C_k = C_{k-1} + (1 - t(v_k)) \cdot c(v_k) \cdot T_{k-1} \tag{2.2}$$

$$T_k = t(v_k) \cdot T_{k-1} \tag{2.3}$$

この畳み込み演算による合成は，隣接関係（特に視点からの距離における前後関係）を保持する限りにおいては，部分合成が可能であるという特徴がある．すなわち，演算区間をいくつかの部分区間に分割し，それぞれの区間内での重畳計算を済ませ，その中間結果を最後に畳み込み演算しても得られる結果は同じになる。

この性質により，複数のノードに分散して割当てた部分三次元空間（以下，サブボリュームと呼ぶ）に対して畳み込み演算を行い，各ノードで得られた画像と画素毎の透明度を視点からの距離の順番に従ってパイプライン的に合成する，という手法による並列化が可能である．

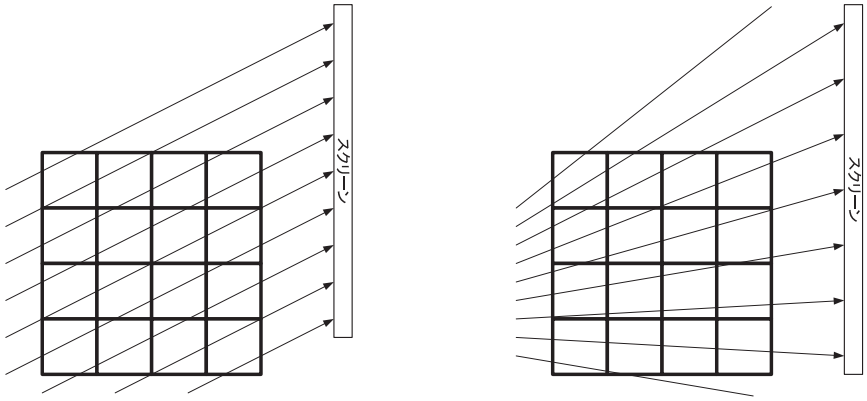
2.1.2 投影方法

視点の位置と視線の関係により，ボリュームレンダリングでは大別して 2 種類の可視化結果の投影方法が存在する（図 2.2）．

並行投影　視点を無限遠におくことで，視線を一方向に固定した投影法．全視線の視線ベクトルが同一となることから，視線ベクトルの計算が一度で済む．また，ボリュームへのアクセスパターンも変化しないため，メモリアクセスパターンが一定周期での繰り返しとなり，事前キャッシュ処理などが実現しやすい．

透視投影　視点から視線を放射線状に出す投影法．ピクセルごとに視線ベクトル計算が必要となり，ボリュームへのアクセスパターンも視線ごとに異なるためメモリアクセスのパターンが複雑となりランダムアクセスに対する性能要求が大きくなる．

透視投影で描画した方が人間が見たときの画像は自然となる．並行投影する場合に比べ透視投影では視線計算やボリュームのアクセスパターンが複雑するが，一方で視線が放射状に進むことでボリューム空間を早期にはずれるものも出てくる．そのため，ボリューム空間の全ボクセルを視線が通過する並行投影に比べると，ボクセルのアクセス量は大幅に減るというトレードオフが存在する．このバランスはボクセルのサンプリング方法に依存する．



(a) 並行投影　　(b) 透視投影

図 2.2: 2 種類の投影方法

2.2　実時間インタラクティブシミュレーション環境

本節では，シミュレーションとボリュームレンダリングの関係について述べ，実時間インタラクティブシミュレーション環境を構築するために必要となるリアルタイム可視化システムについて述べる．

2.2.1　シミュレーションと可視化

通常のシミュレーションでは，初期状態といくつかのパラメータをシミュレーションを開始する前に設定し，そのパラメータをシミュレーションの途中で変更することはまれである．しかし，医療分野で必要とされている触診などのインタラクティブなシミュレーションでは，計算途中に実験者の判断によってパラメータを変更することがしばしば起こり，それに伴い可視化システム側でも時々刻々と変わる状況を実時間で可視化しなければならないという要求が出てくる．

既存のボリュームレンダリングシステムでは，シミュレーションの最終結果

の可視化，CT スキャンの観測結果の可視化など，結果の可視化に主眼が置かれてきたため，上記のようなインタラクティブシミュレーションへの適応が困難である．

実時間インタラクティブシミュレーション環境の構築にあたって，シミュレーション結果や観測結果をその時々でリアルタイムに可視化可能なボリュームレンダリングシステムが求められるようになった．

2.2.2 実時間ボリュームレンダリングシステム

実時間ボリュームレンダリングシステムの利用形態としては，シミュレーションの最終結果のような静的なボリュームデータの可視化（以下「オフライン可視化」と呼ぶ）とシミュレーションの途中経過などの時変ボリュームデータの実時間可視化（以下「オンライン可視化」と呼ぶ）がある．以下に，その詳細を述べる．

オフライン可視化 医療画像の解析やシミュレーションの最終結果の解析等，ボリュームレンダリングの対象として不変のボリュームデータを扱う可視化のことである．データ生成系から可視化系へのボリュームデータの受け渡しは頻繁ではなく，両者が比較的粗に連携している．オフライン可視化では，視点の連続的な変更など，対応対象を何度も可視化して解析することが求められるため，可視化の際のパラメータに対する要求が高い．

オンライン可視化 新しく必要とされ始めた，シミュレーションの途中経過の可視化のような，ボリュームレンダリングの対象が時間変化を伴う場合の可視化のことである．この場合，データ生成系から可視化系への可視化対象データの受け渡しが頻繁に起こる．そのためオンライン可視化では，できるだけ高速なボリュームデータの受け渡しが求められるため，オフライン可視化に比べデータ生成系と可視化系との接続に高い性能が求められる．

上記の 2 種類のボリュームレンダリングに対し，システムの構成方法も大きく分けると 2 種類存在する．一つはオフライン可視化での高性能ボリュームレンダリングへの要求を満たすために，可視化専用のグラフィックスワークステーションをデータ生成系と別に設置する方法である．別のアプローチとしてオンライン可視化に求められるデータ生成系と可視化系の接続性能の向上を図るために，シミュレーション系の大型計算機の中に可視化系を含めてしまう方法がある．

2.3 並列ボリュームレンダリング

ボリュームレンダリングでは可視化する二次元スクリーンのピクセル数分の視線すべてについて，その視線上の色情報Cと透明度Tを重畳し色情報を完成させる．ボリュームデータへのアクセスが自由に行えるならば，各視線毎の並列処理による高速化は容易である．しかし，ボリュームデータやスクリーンサイズの大きなものでは，扱う視線数・視線上のボクセルデータ数がともに多くなるため単体のソフトウェアやハードウェアによる実現は処理量の増大により，実時間性を求める用途には対応できない．

そこで可視化系の計算能力向上のための並列化に関する研究が多く行われている．視線毎の並列処理については，前述の通りボリューム空間へのアクセスがすべてのノードから自由に行えるようにすれば実現できる．また，一つの視線内の処理に関しても，2.1.1 節で述べた計算区間の分割により，重畳し終わったデータについてはCを，終わっていないデータについては中間結果(C, T)を，可視化系の計算ノード間で通信できれば分散処理が可能となる．

前者の並列化実現については，自由なボリューム空間へのアクセスをすべてのノードから行えるようなメモリアクセス機構の実現は全ノードに同一ボリュームを保持させるような仕組みか，または担当するボリューム空間を限定し分割して保持する仕組みが考えられる．ボリュームデータが実時間で変化する場合には，その変化を伝搬できるようなボリューム更新用の通信が発生する．

後者の分散処理については，各計算ノードで自ノードで求めた計算結果と前段のノードから伝搬された中間結果を合成し，次ノードへ伝搬する仕組みが必要となる．ノード内でのボクセルの処理能力や視線の分割数によって通信量の増減はあるが，出来るだけ通信・合成による遅延の少ない通信機構が必要となる．

単体のソフトウェアやハードウェアで実現するときとの大きな違いとして，視点の方向によってノード間の通信方向が大きく変化するという特性がある．そのため，並列ボリュームレンダリングにおいては，ノード間の通信を考慮したレンダリング手法と視点方向による通信速度の低下が少ない通信機構が必要とされる．

既存の実装方法には，ソフトウェアによる実装では PC クラスタを用いたもので，BSC[5] や SLIC[6] といった合成アルゴリズムを工夫したものがあり，専用ハードウェアによる実装としては八分木の通信路を持った VG クラスタ [7] な

どがある．これらはノード間の通信路を十分に確保して視点の方向に起因するレンダリング速度の低下を抑えている．

また，シミュレーション分野とは異なるがボリュームレンダリングの一種であり，高い実時間応答性の実現に特化したゲームのような用途への応用分野では，グラフィック専用プロセッサ（GPU）の並列動作による実装として nVidia 社の SLI や ATI 社の CrosFire が挙げられる．前者は PC 用グラフィックカードを広帯域の専用通信路としてブリッジと呼ばれる専用コネクタで結び，メモリ空間を共有しメモリの広帯域化をはかったもので，後者はグラフィックカード 2 枚にマスター・スレーブの役割を持たせ，DVI ケーブルで結び両者で分担して画像を計算し結果画像をマスター側で合成し表示する仕組みを持っている．ただし，GPU は PC の表示機能に特化しており，枚数のスケーラビリティは無い．また，視点依存性などへの専用回路による対応機能を持たないため，シミュレーション分野にそのまま応用することは困難である．

2.4 VisA（Visualisation Accelerater）によるレンダリングシステム

実時間インタラクティブシミュレーション環境おけるボリュームレンダリングシステムでは，高精細かつ高速な描画速度に加えて高い応答性能が求められる．すなわちシミュレーションなどにより得られるボクセルの更新と，視点の移動による視線の更新がリアルタイムに発生することを想定なければならない．本節では，これらの要求を満たすレンダリングシステムの構築を目指し構築した，実時間インタラクティブシミュレーション環境向けボリュームレンダリングシステム（以下 VisA クラスタと呼ぶ）を紹介する．

2.4.1 システム概要

ReVolver/C40 をベースとして実時間インタラクティブシミュレーション環境との親和性を高めるために，視線生成とシェーディングを PC クラスタが行い，ピクセル値計算のための専用ハードウェア VisA を各ノードに配置した並列ボリュームレンダリングシステムである．PC クラスタを数値シミュレーションの母体として，PCI バスからボリュームデータの更新を行う仕組みで時変ボリュームデータのレンダリングを可能としている（図 2.3）．

ボリュームデータの大規模化によりボトルネックとなるボリュームアクセスを最適化するため，X-Y-Z の三軸の視線方向にあわせボリュームを三重に保持

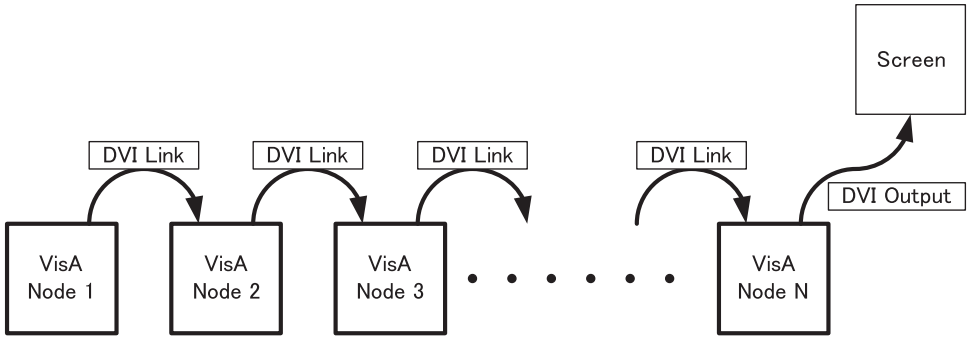


図 2.3: VisA クラスタ

し，主軸等間隔サンプリングを行うという特徴をもつ．

2.4.2 サンプリング手法

ボリュームデータの重畳に用いるサンプリング方法としては，視線[1)]を視線の方向に対し等間隔にボクセルをサンプリングする，視線等間隔サンプリング法（図 2.4(a)）が一般的である．

VisA クラスタでは，視線ベクトル[2)]の成分の絶対値のうち，最大値を持つ座標軸を主軸と定め，この主軸の方向に対して等間隔にボクセルをサンプリングする，主軸等間隔サンプリング法（図 2.4(b)）を用いている．このサンプリング方法と次に述べるボリュームデータの三重化を用いることにより，バンクコンフリクトが生じないメモリアクセスが可能となる．

ただしこの方法では，視線が軸と平行でない時にサンプリング間隔が視線等間隔サンプリング法に対し大きくなるという問題が生じる．そこでピクセル値計算の際に視線ごとにサンプリング間隔が異なることに起因するサンプリング数のずれを補正している．この仕組みにより VisA クラスタは視線ごとのサンプリングが可能となり，PC クラスタに視線計算を任せることで，透視投影によるボリュームレンダリングにも対応可能となる．

2.4.3 ボリュームメモリ構成

並列ボリュームレンダリングにおいては，クラスタのクライアント 1 ノードごとの記憶領域のアクセス範囲が限られているため，ボリュームデータは $N \times N$ の平面スライスや $N \times N \times L$ の直方体スライスに分割してノードに格納される．

VisA クラスタでは ReVolver/C40 のメモリ構成を踏襲したメモリ構成となっ

---

1) 視点を一端としピクセルの中心を通る半直線

2) 視線からスクリーン上のあるピクセルの中心へと向かうベクトル

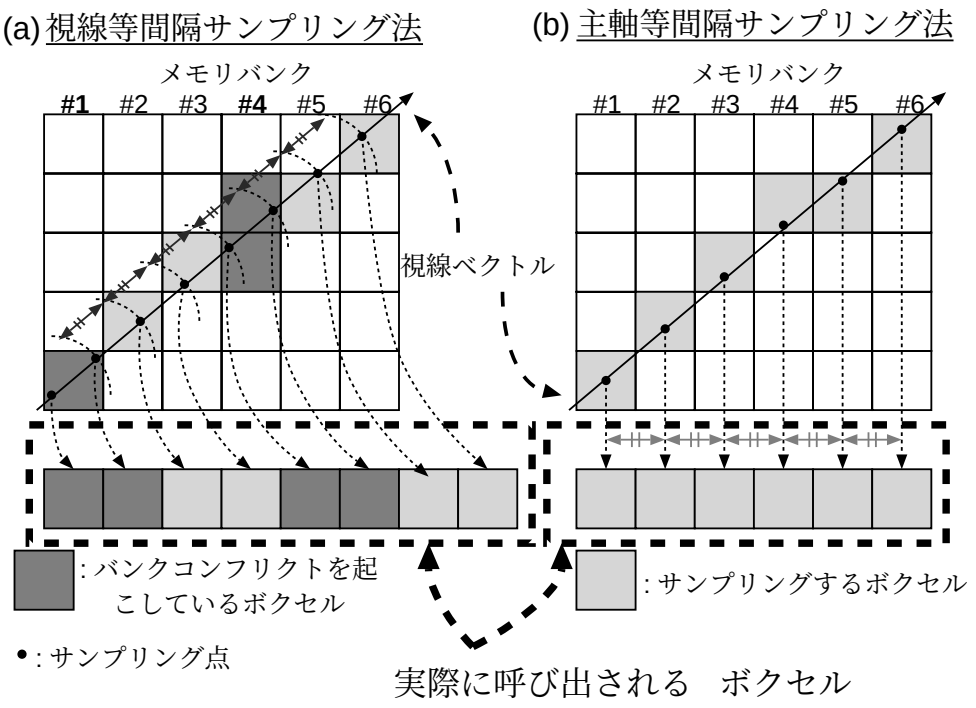


図 2.4: サンプリング手法

ている．図 2.5 のようにボリュー空間を X,Y,Z の 3 つ軸に水平な平面で 3 通りにスライス分割し，各スライス平面群に対して，平面に $1, 2, \cdots, n$ と番号を付け，同じ番号を持つ平面のデータをその番号のメモリバンクの対応する領域に格納する．すなわち，ボリューム内の座標 (p, q, r) のボクセルは，メモリバンク p の X 領域，メモリバンク q の Y 領域，メモリバンク r の Z 領域の 3 箇所に格納されるので，ボリュームデータは三重化して各ノードに分散格納されることになる．これによりレンダリング速度の低下がある程度まで抑えられる．

このメモリ構成と主軸等間隔サンプリング法により，3 つの座標軸のそれぞれに対し，軸に垂直なスライスでボリュームデータが分割された状態でメモリバンクに格納されるので，視線の主軸（視線ベクトルの方向成分の絶対値が最大となる座標軸）方向に垂直なスライスを選ぶことで，バンクコンフリクトフリーなメモリアクセスが実現できる．

2.4.4　ピクセル値計算手法

一つの視線に対するピクセル計算処理は，一次元のパイプライン構成をとることにより，その処理を並列化することが可能である．また，前述のバンクコンフリクトフリーであることを保証したことにより，個々の視線について同一

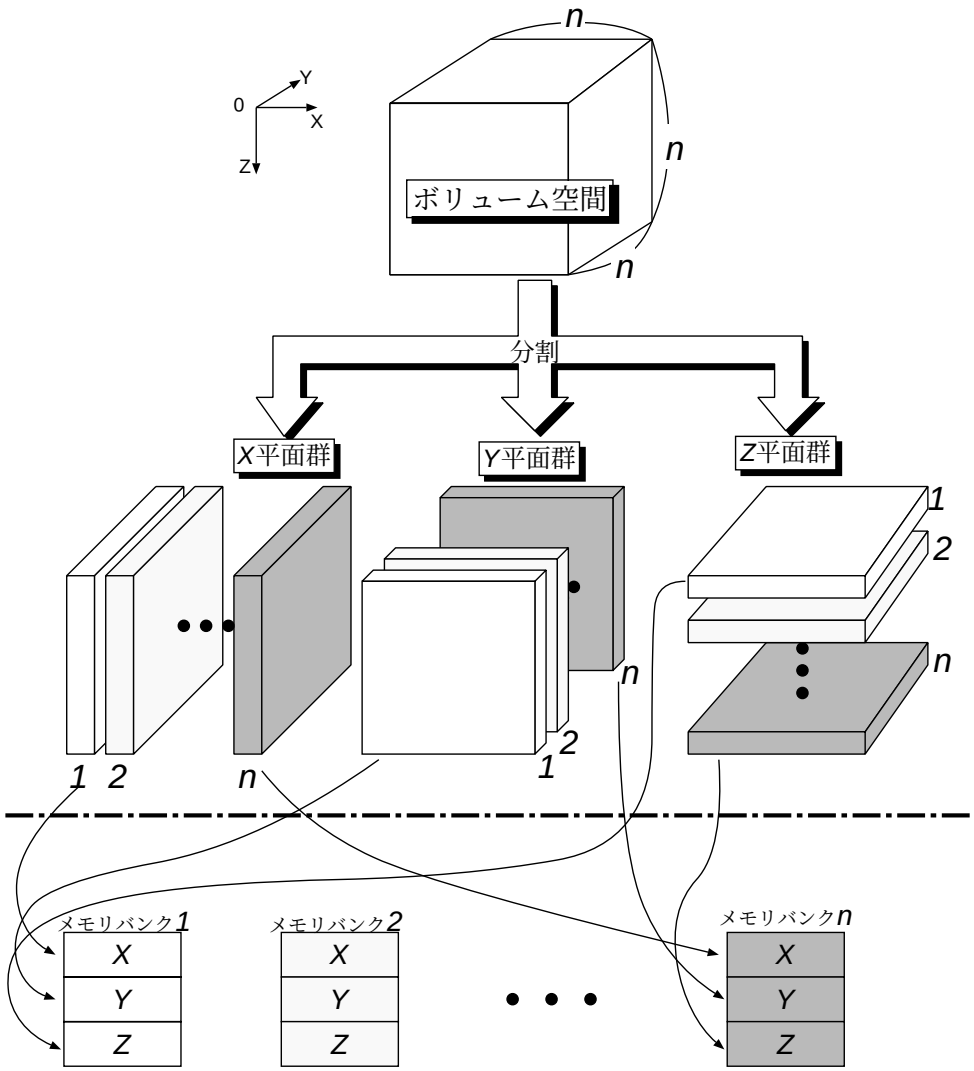


図 2.5: ボリュームデータの格納方法

のボクセルを参照することはないので，視線に関する並列化も可能となる．

各ノードのプロセッサは自分の持つスライスから，処理対象となるピクセルに対応したボクセルをピクセル値を計算し，処理が終了すると次ノードに計算結果と処理対象となる視線の位置情報を渡し，次のピクセルに対応するボクセルの処理に移る，という処理をスクリーンの全ピクセル分繰り返す．

2.4.5 通信路の特徴

2.4.3 項で述べた三重化の仕組みにより，一次元に並んだクラスタ構成ながら三軸方向のリンクを仮想的に持つ処理系となっている．

1 枚分のボリュームレンダリングに必要なボリューム群は X-Y-Z いずれか一

つのスライス群であり，それを視線にあわせ選択利用することで，常に視線の主軸方向で効率の良いノード間パイプライン処理が可能となる．通信路としては，視線の計算パイプラインの間に通信が介在するため，出力のレイテンシを維持するために通信路は通信遅延の少ないもので無ければならない．また，一般的なボリュームレンダリングの性質として，扱うボリュームの量にあわせ通信量が増えるので高速であることも必要である．

2.5 プロトタイプ VisA Pro カード

提案した VisA クラスタのプロトタイプ実装を目的として，VisA での 1 ノードの計算量の半分の処理能力を実現可能な FPGA（Field Programmable Gate Array）搭載汎用 PCI ボードを開発した．VisA クラスタにおける 1 ノード分の処理を図 2.6 のような構成で実現する．

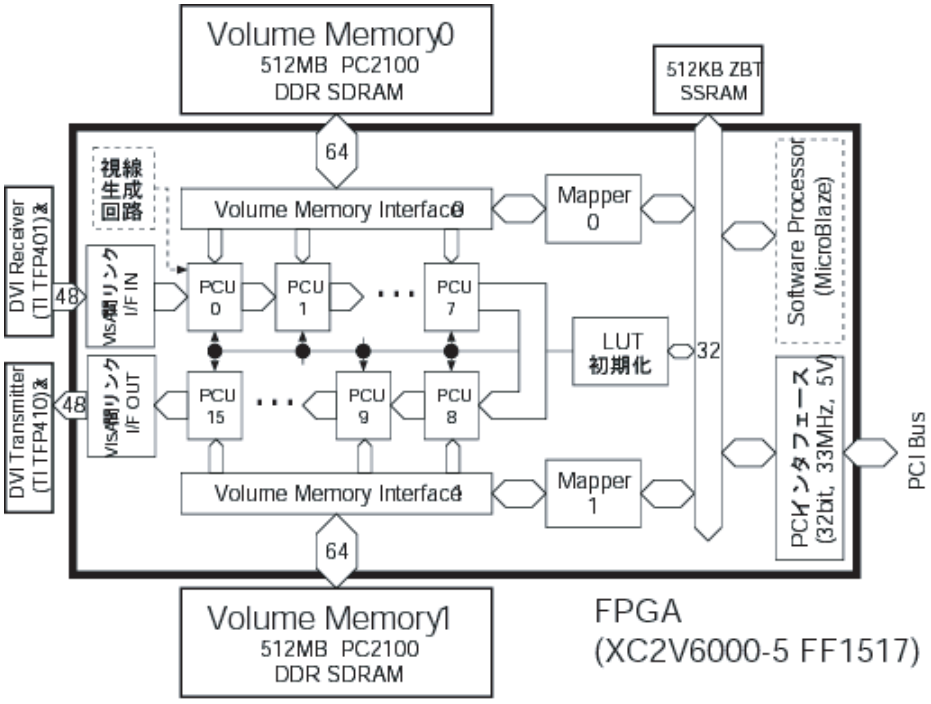


図 2.6: VisA Pro カードブロック図

本稿で新しく提案するレンダリングシステムについてもこのボードを用いたプロトタイプ実装をし今後評価を行っていくので，ここで紹介しておく．

図 2.7: VisA Pro カード (写真提供 東京エレクトロンデバイス (株))

2.5.1 高速大容量メモリ

DDR-SDRAM SO-DIMM を 2 枚搭載し，別系統クロックにて最大 DDR333MHz(PC2700 規格メモリ使用時) にて独立に動作可能である．各メモリスロットには，ノート PC 要に市販されているメモリ容量 1GB までの DDR SO-DIMM モジュールを搭載可能である．従って，128bit 幅の 2GB メモリ，あるいは，64bit 幅の 1GB メモリ 2 チャネル等のメモリ構成が可能である．最大メモリバンド幅は 2 チャネル合計で 5.2GB/s に達する．

VisA クラスタではこのメモリ空間をボリュームデータ格納領域として用いることで，巨大なボリュームデータにも対応可能である．

2.5.2 DVI デュアルリンクの入出力チャネル

高解像度液晶ディスプレイに対応した DVI-D デュアルリンクの入出力チャネルを持つ．これにより，本カードで生成した画像データを直接ディスプレイに出力することや，グラフィックカードのディジタル出力を一旦取り込んで加工したのちに出力すること等が可能である．

DVI インタフェースで使用する LVDS インタフェースチップとしては TI 社の TFP401/TFP410 を各 2 セット搭載し，デュアルリンク構成時には入出力それぞれ 1GB/s の転送速度 (165MHz 動作時) を実現可能であり，この入出力チャネルを並列処理のためのネットワークとして使用することも可能である．

VisA クラスタでも DVI インタフェースは隣接ノード間で，中間値の受け渡しを行う高速リンクとして用いられている．

2.5.3 高性能大容量 FPGA

本ボードは，Xilinx 社の VirtexII シリーズ FPGA XC2V6000-5(600 万ゲート) を搭載する．同 FPGA は，内部に 18bit × 18bit の高速乗算器と 18Kbit のオンチップメモリブロックをそれぞれ 144 個内蔵し，単体ではともに 100MHz 以上で動作可能である．これにより画像処理で必要とされる高い整数演算性能とメモリバンド幅を確保することが出来る．

VisA クラスタでは，この部分にピクセル計算ユニットを FPGA の資源が許す限り多段化して実装し，ボード内で計算処理のピクセルパイプライン化を行っている．

2.5.4 その他の特徴

- ホスト PC とのインターフェースとして PCI64/66 並びに PCI32/33 に対応している．VisA クラスタではシミュレーション系からのボリュームデータの更新用バスとして用いる構想となっている．
- 166MHz 動作時の 512KB(128K×36bit)SSRAM を搭載している．SSRAM には Read/Write アクセス切り替え時の Idle サイクルが不要な ZBT(Zero Bus Turnaround) タイプを採用している．

第3章　ハードウェア向けレイ・キャステング

ボリュームレンダリングの際，視線上に存在するボクセルの値を順に取り出す作業をレイ・キャスティングと言う．この処理は1フレーム分の描画で，ボリュームデータ全体を1度走査せねばならず，いかに早く全体を走査できるによってフレームレートが決定する．

ハードウェアによるボリュームレンダリングの実装を考えたとき，ボリュームデータを格納する場所は大容量で高速という条件からDRAMが利用される．近年DRAMは，一般的なプログラムの参照局所性に基づき，バースト転送を前提として高い帯域性能を実現するという流れで高性能化が進んでいる．しかし，ボリュームレンダリングはその処理の性質上，ボクセルデータの時間的局所性に乏しく，空間的局所性についても視線の方向に依存して低下する．このため，ボリュームレンダリングにおいてメモリに要求される帯域性能を満足させるには，DRAMのバースト転送を念頭に置き，イニシャライズやバンク切り替え，リフレッシュなどの処理によるデータレスポンスの低下を隠蔽できるレイ・キャスティング法を考える必要がある．これは，並列化を行った場合でも各ノード内のサブボリューム単位の処理に関して必要な検討事項である．

この章では，代表的なレイ・キャスティング法でありVisAクラスタでも用いているピクセル順レイ・キャスティング法と，ライン単位で保持されるCPUのキャッシュ特性に着目してこのレイ・キャスティング法の欠点を取り除いた，キューボイド順レイ・キャスティング法[8]を紹介する．

そして，ハードウェアでの実装におけるキューボイド順レイ・キャスティング法の応用について述べる．

3.1　ピクセル順レイ・キャスティング法

このレイ・キャスティング法はボリュームレンダリングにおける最も単純なボリューム空間の走査手順を用いた計算手法である．

手順としては，処理をするピクセルを選び，そのピクセルに対応する視線の視線ベクトルを計算し，以後

1. サンプリング
2. ピクセル値更新
3. 次サンプリング点の決定

という処理を一本の視線についてボリューム空間を外れるまで繰り返す．この一連の処理を投影する二次元スクリーン上の各ピクセルについて順次行い二次元画像を完成させる．

以下，レイ・キャスティングの具体的な手順を簡単化したモデルで説明する．投影方法は透視投影とし，ボリューム空間が N^2 の二次元，ボクセルの計算系への一時保存キャッシュ（CPU による実装であればキャッシュメモリ，ハードウェアによる実装であればプリフェッチバッファ）が 1 回あたり連続するボクセル 4 個分であるモデルで考える．

視線が左から右に向かう図 3.1 (a) の例の場合，視線方向に沿って 4 つのボクセル値が順にサンプリングされる．キャッシュ・ヒット率は最大化される．一方視線が下から上に向かう図 3.1 (b) の例では，1 回のバースト転送で得られるボリュームのキャッシュ方向と視線方向が垂直に交差するため，フェッチした 4 つのデータのうち 1 つのボクセル値しか利用できない．N^2 ボクセルのボリュームデータにおいて，(a) と (b) のような条件を比較した場合，同じフレームレートを実現するには (b) では (a) の N 倍の転送速度が必要となり，メモリのアクセス速度限界によってフレームレートが頭打ちになる．

このようにピクセル順レイ・キャスティング法はアルゴリズムが単純でハードウェア化も容易であるが，大規模ボリュームを扱う上で，視線依存の速度低下が許容出来ない範囲に広がってしまうという問題がある．

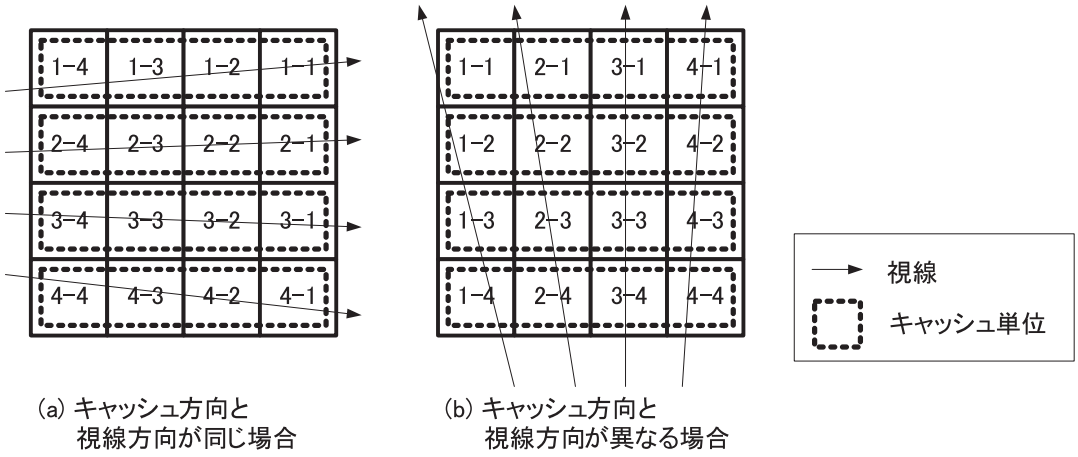


図 3.1: 視線の方向とサンプリング点の処理順序

3.2　キューボイド順レイ・キャスティング法

ボリュームレンダリングでは，重畳計算処理の中断が可能であることに着目し，参照の空間的局所性を向上させる方法として，額田らが考案したキューボイド順のレイ・キャスティング法 [8] がある．

このレイ・キャスティング法では，あらかじめ視線ベクトルの計算を全ての視線について完了しておき，ボクセルをキャッシュ・メモリに蓄えられるだけのデータ量で空間分割し（個々の空間をキューボイドと呼ぶ），視点から遠いキューボイドから順に

1. キューボイドを通る全ての視線に対し重畳計算を行う
2. 計算が終了していない中間値について，その視線のデータの最終出力先にあたるピクセルのバッファに一時保存する
3. 次のキューボイドで登場する視線に対応した中間値を取り出し前データとして渡す

という処理を繰り返す．これによりバースト転送を活かしたメモリ・アクセスで読み出される値を効率よく処理し，視点依存の要求バンク幅の増大を抑えることが可能となる．

3.1 節の場合と同様の簡単化したモデルで手順を説明する（図 3.2）．視点位置から最も遠いキューボイドである点線 1 に属する 4 つのボクセルを 1-1，1-2 の順に処理し，中間値 (C, T) を視線 a の属するピクセルのバッファに一時保存する．さらに 1-3，1-4 の順に処理し，同様に中間値を一時保存する．次に視線から遠いキューボイドは点線 2 に属する 4 つのボクセルで，この 4 つについても図中 2-1，2-2，保存，2-3，2-4，保存という順で処理する．そして再び視線 a のデータ 3-1，3-2 を処理するために保存した中間値を取り出す．3-2 まで処理が終了すると，ピクセルの値が完成するので最終情報 C を保存する．

ここで，各視線情報は主軸等間隔サンプリングにより視線のアクセスするボクセルが独立していることに注意されたい．この性質により，個々のピクセルのバッファにはボクセルの走査順に関係なく中間値が順次更新され最終情報が完成されることが保証される．

額田らの論文 [8] では，ソフトウェア実装による評価を行っており，Itanium2 プロセッサ上での C++プログラムの処理速度のテストによれば，視点位置がピクセル順レイ・キャスティング法で最悪となるパターンで速度が 6 倍向上し，同

じく最良となるパターンでの中間値保存によるオーバーヘッドによる速度低下も 1.2 倍程度で済んだということだ.

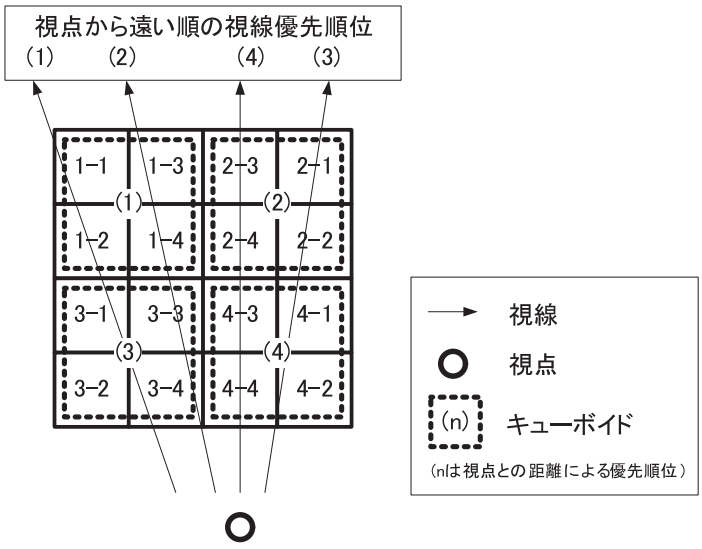


図 3.2: キューボイド順レイ・キャスティング法による処理順序

3.3 キューボイドの原理を応用したハードウェア実装

キューボイド順レイ・キャスティング法の中間値を一時保存する仕組みを利用することで，ボリュームレンダリングをハードウェアにより実現する場合についても，DRAM からのバースト転送に対して読み出しデータの利用効率向上をはかることができる.

キャッシュの代わりにプリフェッチバッファを設け，適当な数のボリュームの集合（ここでは仮にブロックと呼ぶことにする）を 1 つの単位としてバースト転送によりバッファに取り込み，ブロックを通る視線上のボクセルを，キューボイドのレイ走査順序をまねて視点から遠いものから順に処理し，画面出力用フレームバッファのピクセル値に一時保存する形をとる.

CPU のキャッシュを前提としたキューボイド順レイ・キャスティング法の実装に比べ，プリフェッチバッファによるブロック単位のレイ・キャスティングでは，FPGA のリソースの範囲内で自由にバッファサイズを決定でき，入っているデータをより無駄なく使うことが出来る点で優れる．また，レンダリング処理の間に次の読み出しの初期化を行うことにより，DRAM のイニシャライズ時

間の隠蔽と，バースト転送によって取り出されるデータのヒット率向上を同時に見込むことが出来る．これにより，必要バンド幅を抑えられるほか，DRAM のメモリアクセスパターンの複雑化も避けられる．

3.4 まとめ

本章では，ボリュームメモリへのランダムアクセス性を隠蔽し，DRAM にボリュームデータを格納した場合にも性能が見込めるレイ・キャスティング法として，キューボイド順のレイ・キャスティングを応用したハードウェアでの実装について検討を行った．次章では，このレイ・キャスティング法をベースに中間画像合成の仕組みを工夫した新しいボリュームレンダリングシステムの提案を行う．

第4章　高速単方向リンクによる中間画像の並列三次元合成

第2章で示したとおり，ボリュームレンダリングでは色情報Cと透明度Tを重畳し最終的に色情報を画面に表示する．一つの視線上のボクセルデータ群について，部分区間に分割し並列処理する場合，重畳し終わったデータはCを最終画像のピクセル値として出力し，重畳が完了していないデータについては中間結果(C,T)を前後のボクセルデータ群を処理したノードの中間結果(C,T)と重畳できるように次のノードを選択し送信する．各ノードには自分のボリュームの計算結果と前段のノードから伝搬された中間結果を合成する仕組みが必要となる．

我々は，VisAと同様の一次元の高速リンクの特徴を活用した新しい合成手順により，三重化の不要な並列ボリュームレンダリングシステムを提案する．本章ではその通信手法と実装の検討について述べる．

4.1　中間画像合成を用いた並列ボリュームレンダリング

本節ではインタラクティブ性を追求し，高スループットとともに低レイテンシの並列ボリュームレンダリングシステムを実現するための，中間値合成手順の検討について述べる．

4.1.1　中間値合成に関する問題点

ボリュームレンダリングを複数の計算ノードで並列処理する場合，大別すると2つの並列化のアプローチが存在する．一つは，VisAクラスタのように計算ノードを隣接ノード間でつないだ可視化系で，もう一つはVGクラスタのように計算ノードからの中間値を全て一つのノードに集め，集めたノードで中間値の順番にあわせ重畳を行う可視化系である．

どちらの実装においても，並列化することによりノード毎のサブボリュームを処理した中間画像（中間値の集合）の視点からの前後関係に関し，ノードのネットワーク上での隣接関係とボリューム空間上での隣接関係を考慮して，中間画像の合成を行わねばならない．

視点からの前後関係を考慮しなければならない具体例について，ある視線に沿ってた平面上で三次元空間から取り出した，二次元に並ぶボリュームについてを並列化するモデルで説明する（図4.1）．図中に矢印で示した視線A，Bに

ついて，本来の重畳順序はそれぞれノード番号 $(2 \to 1 \to 5)$，$(2 \to 6 \to 5)$ の順であるが，この隣接ノード間の前後関係を保って中間画像を伝搬するためには，ノード 2 と 1，1 と 5，2 と 6，6 と 5 というように隣り合った 4 つのノード間すべてと中間画像の交換が可能な通信路が必要となる．すなわち，視線 A についてであれば，三次元のボリューム空間では隣接 6 方向への通信路を用意するか，ノード 2 から 5 にたどり着くまでノード 1 の中間値と 2 の中間画像を両方の伝送する仕組みが必要となる．

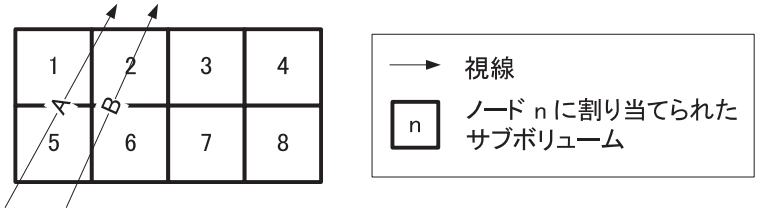


図 4.1: 並列化により発生する中間画像単位のノード間通信

この合成順序の問題をノードのリンク内で解消する方法を探るため，次項以降で検討を行う．

4.1.2 ソートラスト方式の中間値合成モデル

ボリュームレンダリングのサブボリューム分割による並列化を行うときの，中間値の合成順序に関する計算面での特性を明らかにするため，ここでは通信路の容量制約や速度制約が存在しないと仮定したモデルで考える．

サブボリューム間の自由な中間値伝搬が保証されるならば，以下のようなソートラストと呼ばれる単純な実装が考えられる．

サブボリュームの中間値計算を行う計算ノードと中間値の合成専用ノードをもうける（前述の通り，ノード間では制約無く自由に通信可能であるとする）．

計算ノードでの処理

1. 担当するサブボリューム空間の各視線毎に重畳処理を行う．
2. 重畳で得られた中間値については視線の投影スクリーンにおける位置情報 (x, y)，色情報 C，透明度 T に加え，視点からの距離 $D_{pix(N,x,y,z)}$ を合成ノードへ送る．
3. 全ての視線について 2 の計算を繰り返す．

合成ノードでの処理

1. 計算ノードから送られてきた中間値を投影スクリーンにおける位置情報に

あわせ並べ，ノード毎の中間画像として再構成し蓄積する．

2. 全計算ノードからの中間値を格納し終えると中間画像ごとの奥行き情報 $D_{pix(N,x,y,z)}$ の平均を計算し中間画像の奥行き情報 $D_{sub(N)}$ を得る．
3. $D_{subvolume(N)}$ の大きいものから順に画像同士の重畳を行い，最終画像を得る．

この手順のうち 2 と 3 の手順が，前項で述べた重畳順序の整列処理にあたる．

このモデルでは，計算ノードと合成ノードが全て 1:1 のネットワークで接続されていることと同値であり，計算ノード毎の投影スクリーンに対応した中間値の再配置により中間画像を作っているので，計算ノードがもつサブボリュームの隣接関係を全く考慮せずに逐次の合成が可能である．

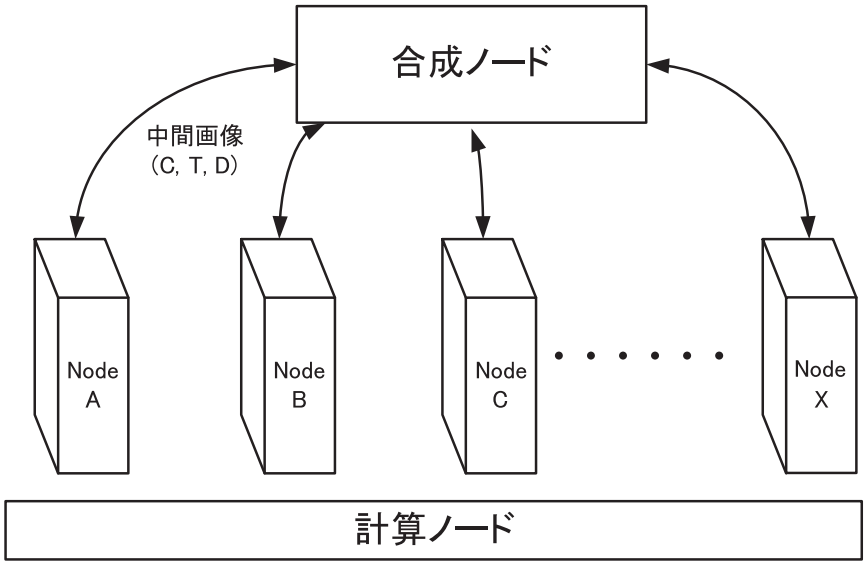


図 4.2: ソートラスト方式の中間値合成モデル

4.1.3 隣接関係を利用した中間値の部分合成

次に，前項のモデルに隣接関係を考慮した拡張を考える．ただし，ここでも通信に関する制約は無いものとする．

前項のモデルでは，合成ノード内で全サブボリュームの中間画像を生成し，視点との距離順に再配置していたが，計算ノード内での各視線の処理時間に比べればサブボリュームの中間値転送にかかる時間は非常に小さいので，合成ノードで行っていた手順 1，2 の処理を計算ノード内で行い中間画像を合成ノードに送るように変更する．

この拡張により，隣接するサブボリュームの中間画像については，視線の主

軸方向で隣接するものに関して，$D_{subvolume(N)}$ を評価する代わりに隣接 2 ノードの主軸方向前後関係の評価を行うだけで部分合成処理をすることが出来る．視点から奥のものから順に合成することは，隣接ノード間では主軸（奥行き方向に正の座標系とする）方向で後ろのものから合成していくこととに置き換えられる．すなわち，計算ノードと合成ノードの間に，主軸方向で隣接するノードに関して部分合成を先に行う仕組みを設けられる．

中間画像の部分合成により得られる画像は，主軸方向に並ぶサブボリューム群の一部を一つのサブボリュームと見なし，そのボリューム内の計算を行い中間画像を得る処理と同値となる．

このモデルでは，前項のモデルに比べ合成ノードの手順 3 で行う並べ替え処理量と重畳回数を大幅に削減することが出来る．

4.1.4　合成ノード不要のモデル

前項のモデルで，隣接する主軸方向の任意の区間について，サブボリュームを連結して，合成ノードの合成回数を削減したが，さらに部分合成をうまく組み合わせた拡張を行う．ただし，ここでも通信に関する制約は無いものとする．

主軸方向で隣接する全てのノードについて部分合成を行うと，最終的に主軸方向に伸びる直方体上のサブボリュームが再構築される．

ここで，再構築されたサブボリュームについて，再度隣接関係に注目すると，最初に注目した主軸と垂直な二次元座標にサブボリュームが並ぶことになり，その二次元座標について視線ベクトルの絶対値が大きい 2 番目の主軸方向を設定することで，上記部分合成の仕組みを再利用しサブボリューム群をさらに連結したボリューム内の計算を行い中間画像を得る処理と同値となる．

同様にもう一度再構築すれば，サブボリューム群は一つのボリューム空間に戻ることになり，中間画像の部分合成によって得られる画像が，全ボリューム空間のボリュームレンダリングを行って得られる最終画像となる．すなわち，X-Y-Z 軸に視線ベクトルの絶対値の大きい順に優先順位をつけ，優先順位に従い隣接している全てのノードの部分合成を行う手順を 3 回繰り返すことにより，合成ノードが不要となり，隣接ノード間の視点に対するの前後関係の評価を行うだけで，逐次的に全ボリューム空間のボリュームレンダリングを行うことが出来る．

手順をまとめると以下のようになる．

計算ノードでの処理

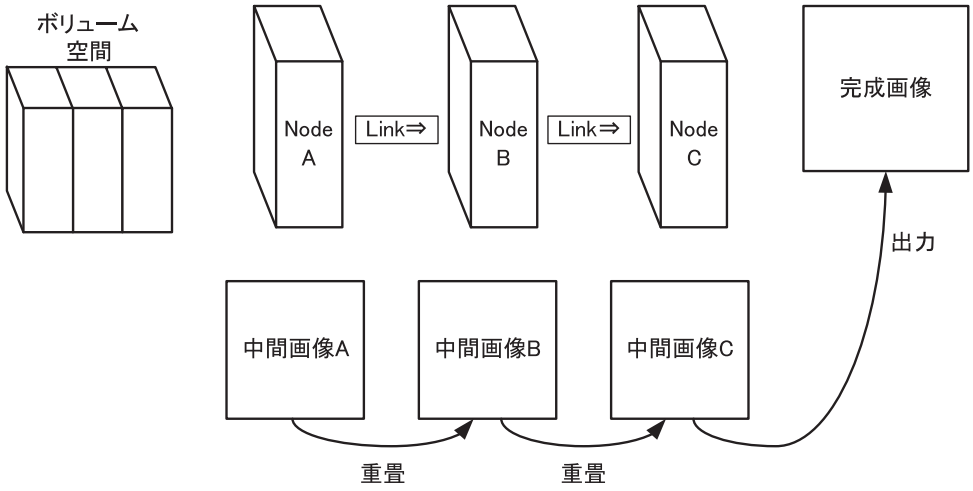


図 4.3: ノードの連続方向にあわせたサブボリューム分割

1. 担当するサブボリューム空間の各視線毎に重畳処理を行う．
2. 重畳で得られた全中間値を投影スクリーンにおける位置情報に並べ中間画像を生成する．
3. 視線ベクトルの絶対値を評価し，大きいものから優先順位をつけ，優先順位に従って各軸方向の部分合成を 3 度繰り返し，最終画像を得る．

ここまで説明した部分合成を用いた合成の手順について例を示す．重畳の第一段階の図 4.4 中 (1)∼(9) の 1 本ずつの矢印の伝搬についてはサブボリューム同士の重畳が行われる．

ここで，各矢印の分を重畳し終わった中間画像についての関係に注目する．全体の内，特定区間を重畳した (1)∼(3)，(4)∼(6)，(7)∼(9) のそれぞれで重畳し終わった中間画像については，隣接する矢印同士は前後関係の正しい隣接した中間画像を保持することになる．これにより図 4.5 のような重畳が可能となる．同様にこれで重畳した図中矢印 (a)∼(c) で重畳し終わった中間画像についても，前後関係の正しい隣接した中間画像を保持することになる．これにより図 4.6 の矢印 (A)，(B) の重畳が可能となり，その結果として得られる画像はボリューム空間の全ボクセルの値を重畳した正しいボリュームレンダリング結果画像となる．

まとめると，図 4.7 のようにボリューム空間をノード数の立方体にサブボリューム分割してノードを単方向のリンクでつなぎ，3 回ツリー状での重畳を行うことで，視線依存のフレームレート低下のない安定したレンダリングが可能となる．

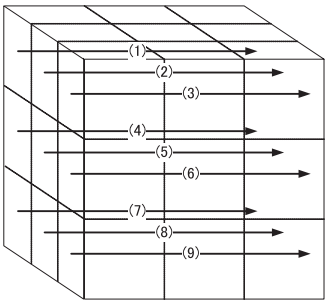


図 4.4: 重畳 第一段階

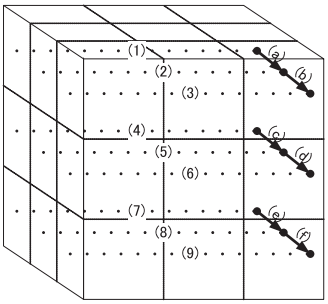


図 4.5: 第二段階

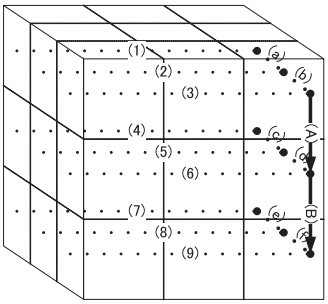


図 4.6: 第三段階

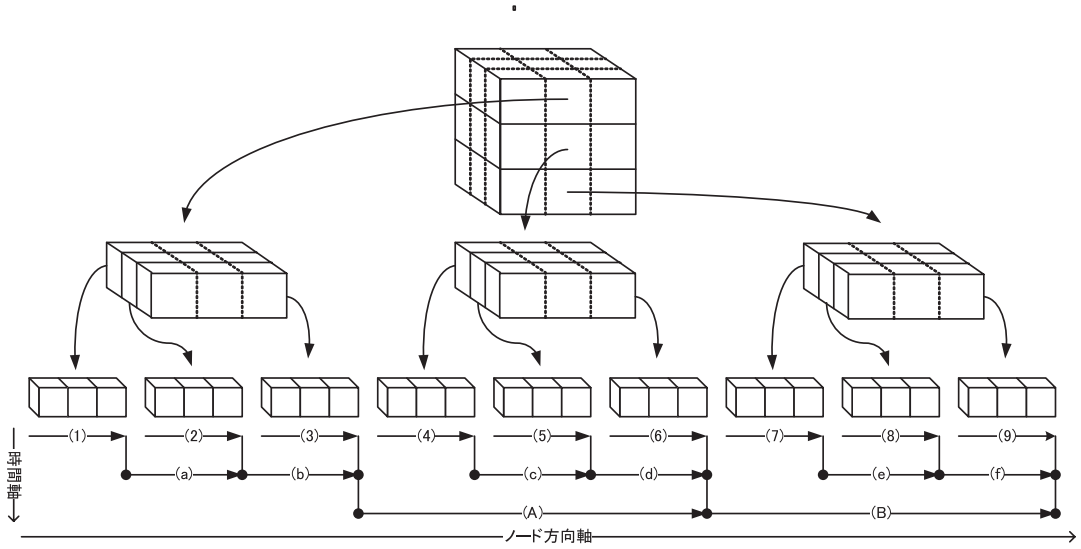


図 4.7: 一筆書きの重畳ツリー

4.1.5　前項モデルに必要な通信路に関する考察

前項までのモデルでは，通信については制約を設けていなかったが，ここで，このモデルの実現に必要な通信路の性質について考察する．

サブボリュームからの全ボリューム空間の再構築について考えると，視線ベクトルの各軸方向成分の絶対値の大小によって，以下の 6 通りの合成順の組み合わせの中から決定され，その順に従って再構築される．

$$(x,y,z),(x,z,y),(y,x,z),(y,z,x),(z,x,y),(z,y,x)$$

また，それぞれのケースについて，視線ベクトルの各軸方向の成分の正負により 8 通りの部分合成パターンが存在する．

これらの全パターンについて部分合成が可能となるためには各ノードは隣接 6 方向への送信パスを持たなければならない．すなわち，各軸方向に全隣接ノード間で格子状の双方向通信路が必要となる．

4.1.6　隣接ノード間の合成計算の工夫

ボクセル重畳の基本式となる漸化式 (2.2) は，2 つのボクセル値の重畳を行う際，視点からの距離の前後関係を考慮して遠いものを右辺の値として与え，重畳しなければならないことを表している．しかし，以下の証明により，前後関係が逆の場合についても，逆の場合の漸化式 (4.1) を用いることで，ノードの前後関係によらず単方向での重畳を行う可能となる．ここでは，それを示す．

式 (2.2) は次の式 (4.1) のように変形することができる．

$$D_{k-1} = D_k \cdot t(v_{k-1}) + (1 - t(v_{k-1})) \cdot c(v_{k-1}) \tag{4.1}$$

式 (2.2) によって求められるピクセル値 C_N は，式 (4.1) の D_0 と同値となる．式 (4.1) の D_{k-1} の値は D_k, c_{k-1}, t_{k-1} から求めることができるので，D_k, v_{k-1} から求めることができる．

$\mathbf{C_N = D_0}$ の証明　計算の便宜上サンプル空間の外側には無色透明 ($t(v_k) = 1, c(v_k) = 0$) のボクセルが存在するとする．

$T_{-1} = 1, C_{-1} = 0, D_{N+1} = 0$ とおくとき，式 (4.1) より

$$\begin{aligned} D_0 &= D_1 \cdot t(v_0) + (1 - t(v_0)) \cdot c(v_0) \\ &= D_2 \cdot t(v_1) \cdot t(v_0) + (1 - t(v_1)) \cdot c(v_1) \cdot t(v_0) + (1 - t(v_0)) \cdot c(v_0) \\ &\quad \cdots \end{aligned}$$

$$
\begin{aligned}
&= D_{N+1} \cdot t_{N-1} \cdot t_{N-2} \cdots t_0 + (1 - t(v_N)) \cdot t(v_{N-1}) \cdot t(v_{N-2}) \cdot t(v_{N-3}) \cdots t(v_0) \\
&\quad + (1 - t(v_{N-2})) \cdot c(v_{N-2}) \cdot t(v_{N-3}) \cdots t(v_0) + \cdots + (1 - t(v_0)) \cdot c(v_0) \\
&= D_{N+1} \cdot t_{N-1} \cdot t_{N-2} \cdots t_0 + (1 - t(v_{N-1})) \cdot c(v_{N-1}) \cdot T_{N-2} \\
&\quad + (1 - t(v_{N-2})) \cdot c(v_{N-2}) \cdot T_{N-3} + \cdots + (1 - t(v_1)) \cdot c(v_1) \cdot T_0 + (1 - t(v_0) \cdot C_0
\end{aligned}
$$

となる．また式 (2.2) より，以下の式も成立する．

$$
\begin{aligned}
C_N &= C_{N-1} + (1 - t(v_N)) \cdot c(v_N) \cdot T_{N-1} \\
&= C_{N-2} + (1 - t(v_{N-1})) \cdot c(v_{N-1}) \cdot T_{N-2} + (1 - t(v_N)) \cdot c(v_N) \cdot T_{N-1} \\
&\quad \cdots \\
&= C_{-1} + (1 - t(v_0)) \cdot c(v_0) \cdot T_{-1} + (1 - t(v_1)) \cdot c(v_1) \cdot T_0 + (1 - t(v_2)) \cdot c(v_2) \cdot T_1 \\
&\quad + \cdots + (1 - t(v_{N-1})) \cdot c(v_{N-1}) \cdot T_{N-2} + (1 - t(v_N)) \cdot c(v_N) \cdot T_{N-1}
\end{aligned}
$$

この 2 式ぶ $T_{-1} = 1, C_{-1} = 0, D_{N+1} = 0$ を代入すると，$C_N = D_0$ の関係が成立する．

4.1.7 まとめ

並列ボリュームレンダリングの一つの実装方法として，隣接関係に注目した中間値の部分合成を行うことにより，X-Y-Z の三軸方向にそれぞれリンク方向が任意の単方向通信路を持つ計算ノード群を用いた実装が可能であることを示した．

4.2 一次元単方向リンクへの三次元リンクの埋め込み

前節で述べた部分合成を用いた並列ボリュームレンダリングを具体的に実装しようとすると，三次元の単方向リンクを実現することは不可能ではないが，ある視点から見た 1 枚のレンダリング結果を得る処理においては，3 回の合成で全リンクのうち半分以下しか利用されない．

具体的なリンクの利用率は N^3 個の計算ノードが並ぶとき

$$
\frac{N \times (N+1) \times (N+1) + N \times (N+1) + N}{N \times (N+1) \times 3N} = \frac{N(N^2 + 3N + 3)}{3N^2(N+1)}
$$

となり，ノード数が増えると 1/3 に収束する

そこで本節では，ノード間リンクを減らすために一次元単方向リンクに提案する部分合成のための三次元単方向リンク埋め込んでも，ボリュームレンダリング処理が可能であることを示す．

4.2.1 単方向一次元リンクによるノードの連結

図 4.8 のような一次元単方向リンクをモデルに考える．このモデルでは，視点からの距離に依らない任意の順序で軸に優先順位をつけ（以下説明のために仮に (x, y, z) の順とする），x 軸方向のリンクまず設け，(x, y) 平面上でその個々のリンクが一直線につながるように，y 軸方向で結び，さらにその直線の末端を z 軸方向で連結している．

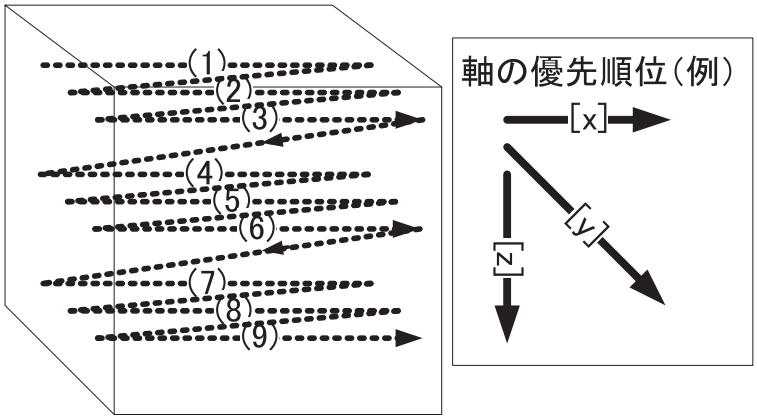


図 4.8: 優先順位を任意につけた計算ノードの一次元リンク

一次元リンクの上では全ノードがリンクしているので，各ノードに入力した中間画像をそのまま出力できる機構を設けてやると，ノード間の中間画像の伝搬が可能である．ただし，単方向の制約から一次元リンク上の上流に位置するノードから下流に位置するノードへ方向への伝搬に限定される．

4.2.2 前節提案手法の合成可否の検証

任意に設定した (x, y, z) 順の一次元リンクのノードの位置による制約のなかで，前節で提案した 3 回の部分合成が実現できるかを検証する．説明のために，以下視線ベクトルの方向による部分合成での軸を優先順位にそって，第 1 軸，第 2 軸，第 3 軸と呼ぶことにする．

第 1 軸の合成を行うとき，3 ノードずつの 9 組の合成について個別にみると，1 組あたりの 3 ノードは必ず一次元リンク上に順番に並んでいるため，問題なく合成可能である（図 4.9）．

第 2 軸の合成を行うときには，第 1 軸方向の合成は完了しているので，第 1 軸の方向に集められた残り 2 つの軸によって出来る平面に，第 1 軸に沿ってで合成した 9 つの中間画像が並んでいる．このとき一次元リンクのうち第 1 軸方

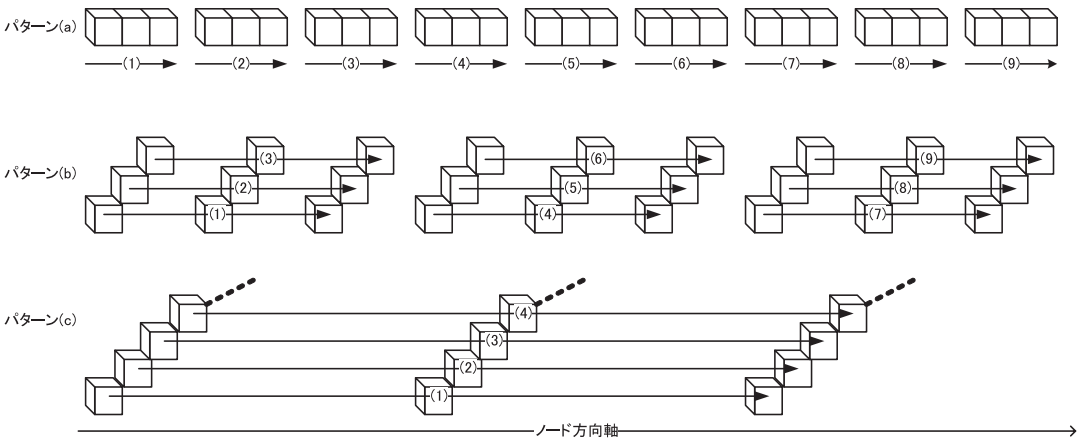


図 4.9: 重畳 第一段階

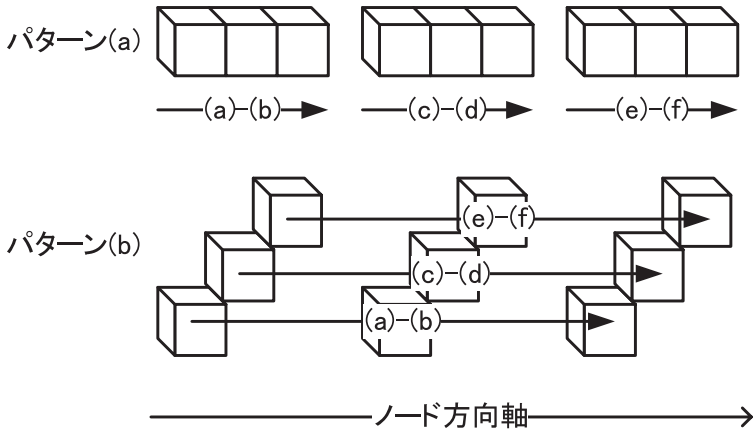


図 4.10: 第二段階

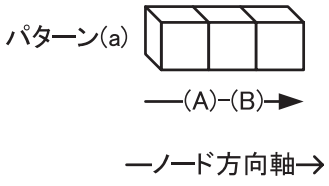


図 4.11: 第三段階

向については伝搬路としての役割のみを持ち，図 4.10 に示したように，中間画像は 3 つのパターンのいずれかの順で，一次元リンク上に順番に並ぶ．よって，第 2 軸についても問題なく合成可能である．

第 3 軸の合成を行うときには，図 4.10 の各のパターンに対して，図 4.11 ように 3 つの中間画像が，必ず順番に並ぶため，どの場合についても一次元リンク上で合成可能となる．

以上により，部分合成における軸の優先順位に依らず，一次元の単方向リンク上に並べた計算ノードについて，仮想的に三次元のリンクが埋め込まれていると考えることで，前節で提示したボリュームレンダリングの計算手順をそのまま実行することが可能であることが示せた．

4.3　一次元単方向リンクによるボリュームレンダリングの検討

本節では，前節で述べた一次元単方向リンクを用いた部分合成による並列ボリュームレンダリングの実装に向けた，具体的な通信路の検討を行う．

4.3.1　提案モデルの実装上の問題点

これまで部分合成によるボリュームレンダリングが可能であるとしてきたモデルでは，通信路について方向以外の制約が無いものと仮定して説明してきた．

前節で述べた一次元単方向リンクを用いるモデルでは，視点に依存した隣接ノードの部分合成ではない部分合成を行う必要がでてくため，ノードを超える通信の多重伝送が無制限に可能であることが前提となる．

しかし，このモデルの実装を考えたとき，無制限の多重伝送を保証出来るような通信路は存在しない．多重伝送が出来ない通信路で構成すると仮定すると，例えば第 1 軸の合成にあたり，図 4.9 のパターン (b) や (c) の例のように，中間画像の隣接ノードを超えた合成が発生した場合，3 ノードずつの 9 組の合成について，同時には 1 組ずつしか合成できないため，時間方向で最大 9 倍の時間がかかってしまう．

この問題はノード数が増加すると深刻になり，N^3 個のノードを持つシステムでは第 1 軸の合成について最悪パターンで N^2 倍の時間がかかることになる．

4.3.2　中間画像のピクセル単位のパイプライン合成の提案

ここで，隣接するサブボリュームの部分合成に用いてきた中間画像の合成に注目する．中間画像は投影スクリーンと同じサイズの画像として伝搬され，各ノードでの合成処理は，全ての画像のピクセルについて，同じ座標のピクセル毎にで中間値 (C,T) を 1 段分重畳計算することになる．座標によって画像の位置が決まっているため，この合成処理をピクセル単位で行うことも可能である．

この性質を利用して，ノード間の中間画像合成をピクセル単位でパイプライン化することが可能になる（図 4.12）．すなわち，中間画像のピクセル単位の粒度で中間値 (C,T) の伝搬を行えるような通信路を用意してやれば，前節で述べた一次元単方向リンクの導入による，視点依存の合成時間増大を大幅に改善することが可能となる（図 4.13）．

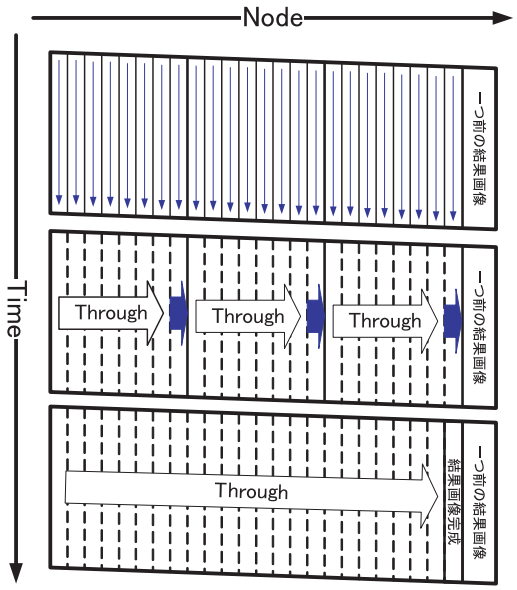


図 4.12: 中間画像合成のピクセル単位パイプライン化

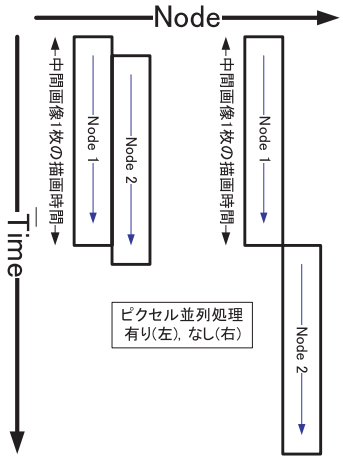


図 4.13: パイプライン化の有無による合成時間短縮イメージ

4.4 まとめ

本章では，一次元単方向リンクによって実現可能な並列ボリュームレンダリングシステムの提案を行った．サブボリュームの隣接関係を考慮した中間画像の部分合成を行う仕組みと，ピクセル粒度で中間値の伝搬が可能な一次元単方向リンクによる通信路を組み合わせることにより，ノードのスケーラビリティに優れ，視点依存によるフレームレート低下の影響が少ないという特徴を持つ．

第5章　DVIインタフェースによる通信

前章で述べたとおり，一次元のノードリンク上の隣接関係とボリューム空間上の隣接関係の重なりを考慮した中間値合成の仕組みを用いることで，一次元の高速リンクで結ばれた計算ノード群でも，視点依存による処理速度の低下を抑えたボリュームレンダリング処理が可能である．また，レンダリングの計算上の特性により視線の正負方向の違いは吸収できるため，一次元のノードリンクは単方向リンクでよいことを示した．

視線方向の正負による制約の緩和により，ノード毎の書き込みタイミング待ち合わせなども重畳伝搬の時に行うだけで済むため，ネットワークは単方向で高速化のみに特化したもので実現できる．我々は，このネットワーク特性に合致した伝搬路の一実装対象として，VisA クラスタでのリンクでも用いていた液晶ディスプレイ表示用デジタル LVDS （Low Voltage Differential Signaling）リンクである DVI インタフェースによる実装を検討した．

本章では，DVI インタフェースを通信路としてみたときの性能を評価し，提案システムの実現に向けた実装の妥当性について検討する．

5.1　通信路としてみたDVIインタフェース

DVI インタフェースは本来液晶ディスプレイのデジタル接続のためのインタフェースである．このインタフェースは液晶のピクセルクロックに同期して 1 ピクセル辺り 24bit(RGB 各 8bit) のデータを 1 フレーム分 × 毎秒 60 枚シリアル伝送する通信路と見なすことが出来る．たとえば UXGA(1600×1200pixs) の表示では，

$$1600 \times 1200 \times 24 \times 60 = 2.76\text{Gbps}$$

という高速通信を行っている．

提案する単方向リンクの実装に最適な安価な伝送路であり，液晶への結果出力が容易となるという副次的メリットも得られる．

5.1.1　ピクセル対応の通信方法の提案

DVI インタフェースの信号は Digital Display Working Group の Digital Visual Interface DVI Revision 1.0 規格で定めらている．VESA の Monitor Timing Specifications に準拠したピクセルクロック・同期サイクルにより，1 ライン分について図 5.1 のように，表示データと液晶ディスプレイの水平・垂直同期や

付加データの送受信用の非表示データとで構成されている．信号帯域を最大限に利用するには入出力部で非表示データ部にも情報を載せる手法が考えられる．たとえばUXGAの表示であれば，ピクセルクロック161MHzで送信できるので，ピクセルに戻すと仮想的には，2160×1245pixsのスクリーン情報が伝送可能であり，

$$(1600 + 560) \times (1200 + 45) \times 24 \times 60 = 3.87\text{Gbps}$$

の帯域を持つ伝送路と見ることができる．

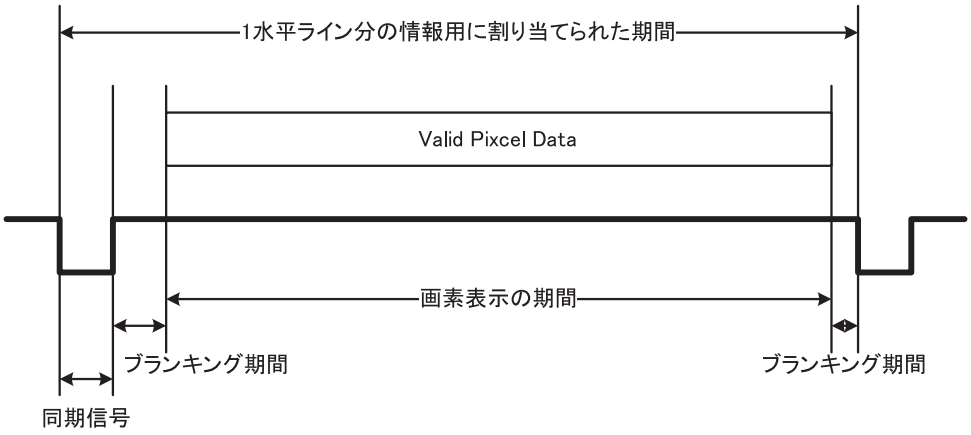


図5.1: DVIインタフェースの信号（1ライン分）

仮想スクリーンのピクセル1つずつに24bitの情報を載せられる．ピクセルの色情報24bitをメモリのセル，フレーム中での位置x-yをアドレスとおき，図5.4のように数フレームを単位周期として単位周期内でバンク番号を割り振り，送信ノード受信ノードともにフレームサイズのフレームバッファを置くと，送信側からは書き込み専用，受信側からは読み出し専用のメモリと見なすことができるようになる．送信側のバッファを省略しデータとアドレスを1パケットとして送信し，受信ノードでアドレスを解釈しデータを受け取れば，データ・アドレス多重バスと見なすこともできる．

また，DVIインタフェースを持つカード上での信号処理について考えると，受信したデータを直接送信回路に流し込むことで，フレームのデータをそのまま次ノードへスルー出力することも可能である（図5.2）．これにより帯域の占有を許可すれば隣り合ったノードに限定しない通信も可能となる．

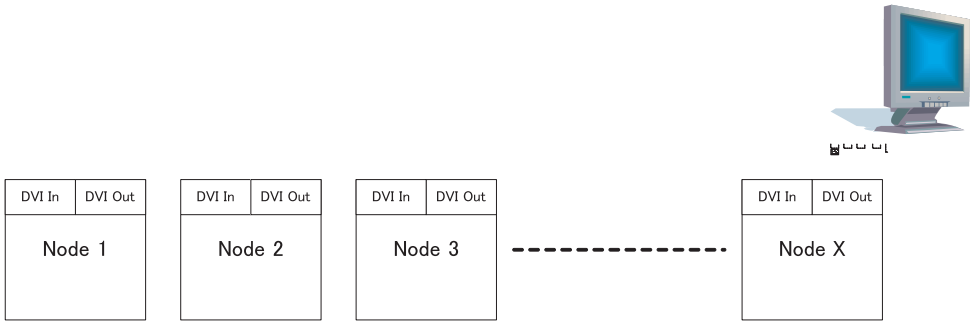


図 5.2: DVI 入出力を用いた画面のスルー出力

5.1.2 必要通信速度と通信時間の余裕

今回の重畳処理に適した通信方法を考えると，ノード間通信は次ノードへの中間画像送信が主体となる．当面の処理目標として，1024^2 のスクリーンに 1024^3 のボリュームデータを用い 15fps の処理を実現することを目標とする．

2 ノード間の伝搬をベースとした中間画像伝搬の仕組みにおいて必要な中間画像格納用のメモリ量は，前後のノードでの重畳とスルー出力の機能が備わっていれば，スクリーンデータ 1 枚分で十分である．1 ピクセルあたり色情報 C を 24bit，透明度 T を 12bit 割り当てるとして，仮想フレーム 1 フレーム（2160×1245pixs）の中に収まる．15fps を目標としたとき，実際の仮想フレームの更新周期は 60fps なので，1 枚のレンダリング結果を表示するのに対し 4 フレーム分の時間と帯域が割り当てられることになる．この時間の余裕を利用することで，提案するピクセル並列でのパイプライン化を行った 3 段階の軸方向単位の部分合成が可能となる（図 5.3）．

5.1.3 通信路としての実装面での改良の検討

前項での試算の通りフレーム 1 枚分の中間値バッファを用意し重畳・スルー出力することで，提案する並列ボリュームレンダリングシステムの通信路は実装可能であるといえる．

しかし，実際には個々のノードが担当するサブボリューム内の視線すべての更新量は小さいのに対し，全ノードでフレーム 1 枚分の中間値バッファを用意することは非常に無駄が多い．ここでは，さらに大きなボリュームへの対応や，より高いフレームレートの実現を目指したときに必要となる帯域の効率利用について考える．

個々のノードが書き込みを行うフレームの領域サイズに合わせフレームバッ

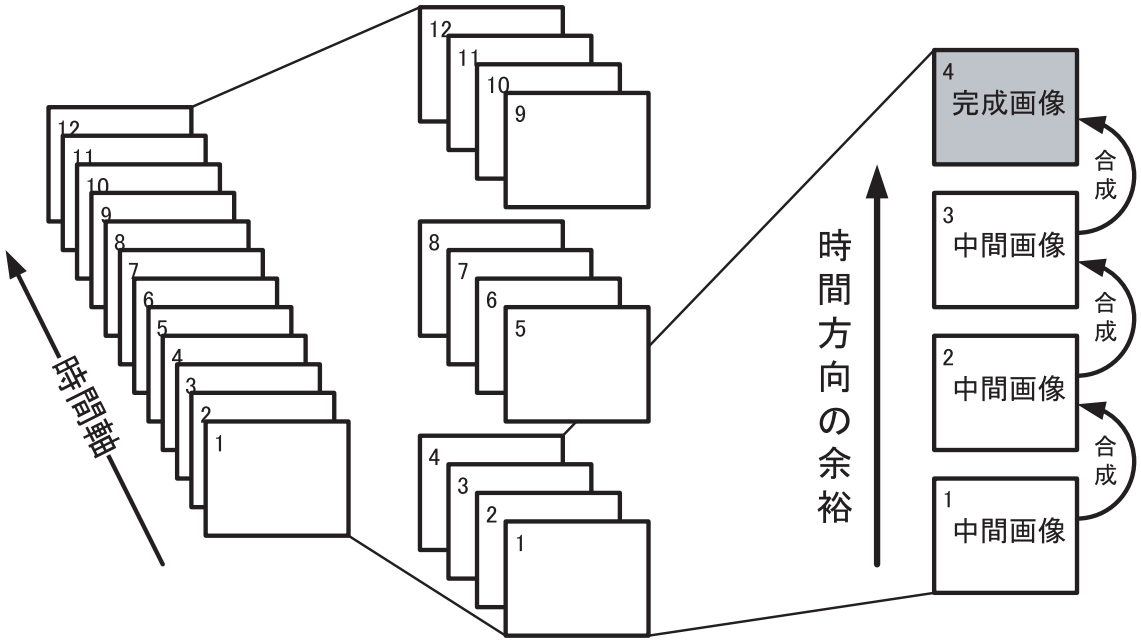


図 5.3: フレームの時間軸方向の余裕を利用した中間値合成

ファのサイズを削減し（視点依存による最大サイズに合わせるとする），自ノードが重畳更新する領域以外のデータについてはスルー出力してしまいえばよい．

一方ノード数が多くなると，スルー出力での伝搬には伝搬路の配線遅延の問題からスルーできる段数に限界がある．そこで，重畳更新する領域以外についてもスクリーン全体を数分割し，数ノードごとにスクリーンの担当領域（以後，タイルと呼ぶ）を設定し，担当タイルについては無条件にフレームバッファに取り込む仕組みを設けることを考える．

具体的な動作としては，担当タイルのデータを受け取った時のみ自ノードのフレームバッファに取り込み，それ以外のタイルについてはスルー出力する．取り込み時にはフレームバッファの内容のうち更新すべき部分を更新し，再び自ノードの担当タイルが来たときに次ノードへ送り出す．この動作は DRAM でいうリフレッシュの原理とにている．

これにより DVI インタフェースによるディスプレーへの入出力機構を単方向にリンクした伝搬路を，24bit 幅の FIFO の役割を持った遅延線メモリ（書き換えはセル単位でパイプライン処理が可能）として扱う事ができる．伝搬路を遅延線メモリと見なしたとき，その大きさはノード毎のフレームの更新周期によって自由に決定できるので，投影スクリーンの大きさと計算ノード数，要求フレー

ムレートにあわせタイルの分割数を増減可能である．

図 5.5 の例で説明すると，スクリーンを A∼D に 4 分割し，左のノードから右のノードへ伝搬していく間のに発生する，4 ノードごとの更新の様子を示している．たとえばノード A 系列であれば，4 タイル毎に A1，A2，A3，· · · と書き変えていくことになる..

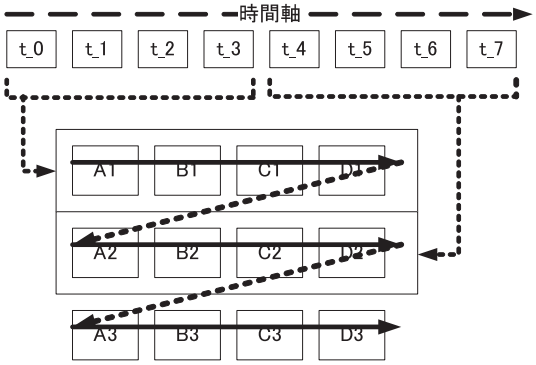


図 5.4: 時分割でのフレームの番号付け

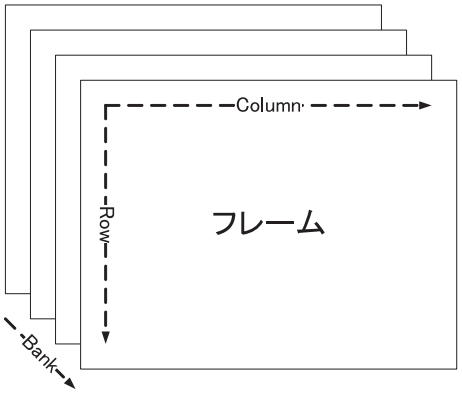


図 5.5: ピクセルをメモリに対応づけた通信方法

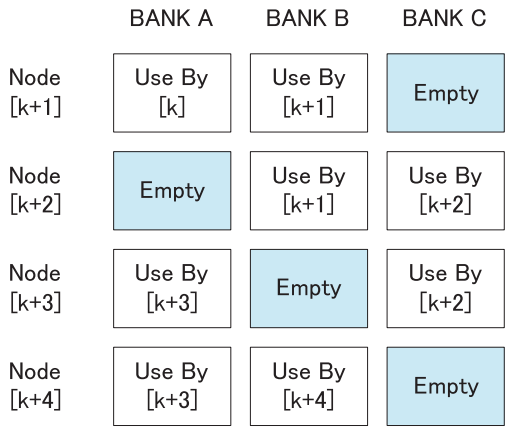


図 5.6: バンクのノード間共有の概要

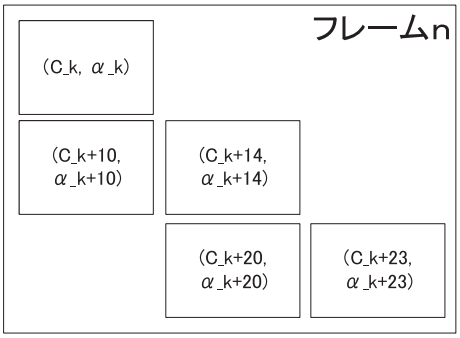


図 5.7: フレームのノード間共有の概要

第6章　まとめ

6.1　超高速単方向リンクを用いた並列ボリュームレンダリングシステム

我々は，双方向通信や多次元のノード間リンクを不要とし，またVisAにおけるボリュームデータの三重化や，VGクラスタにおける合成専用ノードの設置なども必要としない，新しいボリュームレンダリングシステムの提案と実装に向けた検討を行った．

並列化に当たっては，サブボリュームの処理について，ソフトウェアによる実装におけるキャッシュヒット率を最大限に向上させるキューボイド順レイ・キャスティング法の原理をベースに，ボリュームデータ格納領域となるDRAMのアクセス性能との親和性を考慮したハードウェア向けのレイ・キャスティング法について提案した．

また，一つの視線上のボクセル値に関し，奥行き方向の前後関係が保証される限りにおいては，隣接するボクセル同士自由に視線を区間分割して計算することが出来る，というボリュームレンダリングの性質を利用し，サブボリューム単位での中間画像の合成順序に部分合成を多段化して行うという工夫と，合成方向が逆の場合に使える漸化式の採用により，非常に単純な単方向の高速一次元リンクによって，ノード数におけるスケーラビリティの高い並列ボリュームレンダリングシステムが実現できることを示した．

最後にDVIインタフェースによる通信路の性質について評価を行った．インタラクティブ性の実現にも最適な高速な通信路で，提案する一次元単方向リンクによる実装の対象として，十分な通信リンクが構成できることを示した．

6.2　シミュレーションとの協調機構の検討

本稿では，シミュレーションとの協調については言及してこなかったが，VisAクラスタ同様，シミュレーション環境としてPCクラスタを想定している．しかし，VisAクラスタにおけるシミュレーションは,「大規模データの可視化を前提としており，シミュレーションがそのような大規模データを生成するのには多くの時間がかかり，データの更新頻度はそれほど多くない」という，非常に大きな制約を持った想定となっている．三重化を行う上での制約であるためVisAクラスタではこのような想定にならざるを得なかったということだが，シ

ミュレーション対象が医療などのオンライン可視化を必要とする分野であった場合には利用できない.

我々の提案では，ボリュームの更新はボリュームデータがPCクラスタ側で生成されたあと，転置などの必要なくボリュームデータをサブボリュームに分割し各ノードに送り込めばよい．そういう意味でも三重化しないことには大きなメリットがあるといえる.

三重化を廃しVisAクラスタのリンクをより効率化した我々の提案手法では，PCIバスからのボリュームデータ更新でも，VisAクラスタと比べれば，サブボリューム分のPCクラスタ側でのノード間伝搬量も1/3となり，転置など再配置処理も不要なので単純にサブボリュームの立方体データを1回伝搬するだけで済む.

しかし，PCIバスはノード側の処理速度（VisA ProカードではFPGAの動作クロック）に対し，33MHz固定となり大きなボトルネックになる可能性がある．データの圧縮を意図して差分送信という手法も考えられるが，ノード側でデータ解凍処理を各ボクセルについて行っていては応答性能の低下が避けられなくなる.

そこでさらなる高速応答を行うための提案として，我々のシステムの特徴でもあるDVIリンクをデータの入り口として利用する手法を述べる．画面情報としてボリュームデータをPCのフレームデータとして構成し，DVI出力を持つグラフィックカードから出力する．各ノードはフィード時は自ノードのデータを取り込みながら最終ノードまでスルーし続け，中間値バッファに自ノードのサブボリュームデータを取り込みDRAMに書き込む．DRAMへの取り込み量はノード数に依存するがDRAMの書き込み速度（PC2100 DDR-SDRAMメモリで133MHz）に対して，ピクセルクロックは同等（UXGAでは161MHz（但し，画面データに当たる部分の実効クロックは135MHz)）であるが，画面情報1ピクセルのbit幅に対しDRAMの1セルbit幅は大きいのでフィード側からは連続で送り実効2.87Gbpsの速度が実現可能である.

本来の提案システムは超高速リンクについては，単方向で高速という条件を満たせば何を通信路としても良かったが，このフィード方法を用いる場合はDVIインタフェースに限定されることになってしまう．しかし，シミュレーションとの協調環境では非常に有用な手段であると言える.

6.3　負荷分散の検討

この節では，単方向リンクの最終ノードから最初のノードへの帰還パスがあった場合のリング構造を利用した負荷分散の可能性について述べる．

6.3.1　提案システムにおける負荷不均衡の問題

ボリュームレンダリングでは，ボリューム空間のデータの密度は多くの場合中央であったり注視点を中心とする範囲に偏っており，投影方法に関係なく周辺部にはデータが存在しないようなことが多いという性質がある．また，データの密度が均一であっても透視投影の場合視点から広角に広がる視線については，視点から遠ざかるとボリューム空間の外へ視線が外れてしまうため，結果的に周辺部のサブボリュームを通過する視線数は中央部に比べると相対的に少なくなる．

こういった状況に対し，提案システムではボリューム空間をノード数で等分したものを想定しており，実際にはサブボリュームの割り当て場所による負荷の不均衡が起こりうる．

6.3.2　負荷分散のための帰還パスの増設

一般的な並列分散クラスタであれば，ロードバランサにより負荷の監視を行い，負荷の不均衡を是正するようになっているものが多い．しかし，提案システムにはノードの一筆書き方向の伝搬路しか存在しないため，全体の負荷情報を共有するような仕組みの実現は困難である．提案システムでは最終ノードの出力を DVI による液晶ディスプレーへのレンダリング結果表示を行っていたが，結果出力用のパスは別に存在すると仮定し，最終ノードの DVI 出力を最初のノードの DVI 入力に連結し，全体で単方向リングの構造を作った場合に，提案システムの一筆書きによるボリューム空間走査の仕組みを維持したまま負荷分散に対応できるかを考える．

検討に当たり，図 4.8 のモデルを利用する．図中 (1)〜 (9) の矢印について，個々の矢印内でノードリンクの方向に沿って前後にサブボリュームの割り当て量を移動することは特に問題なく可能である（図 6.1）．また，(1)〜 (9) の矢印同士で負荷分散を行う場合についても，1 本の矢印に割り当てるノードの数を調整し，前後のノードで担当サブボリュームを移動させれば，問題なく負荷分散が可能になる．（図 6.2）．

以上の検討により，帰還パスを設けることにより，前節で述べたボリュームを

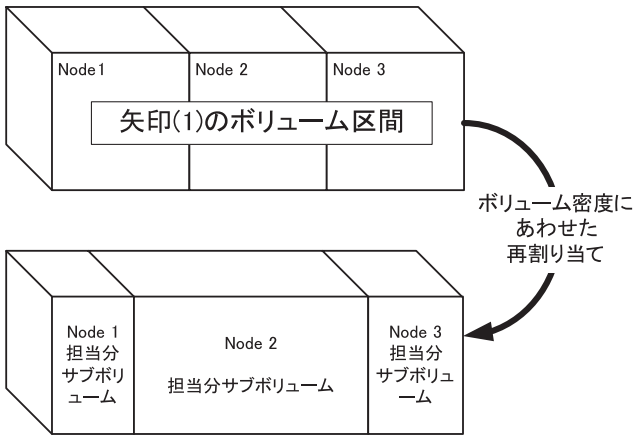


図 6.1: 矢印内のサブボリューム再配置

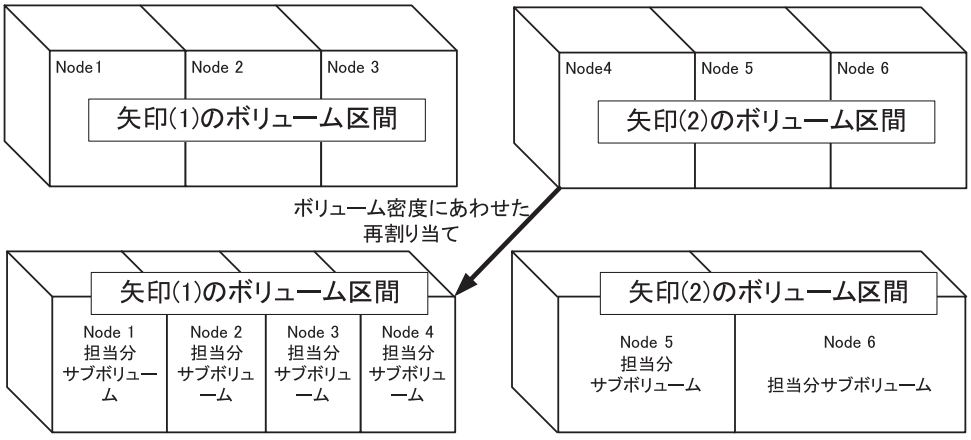


図 6.2: 矢印間のサブボリューム再配置

フィードする仕組みを組み合わせれば，提案システムの一筆書きによるボリューム空間走査の仕組みのメリットを活かしながら，負荷分散の仕組みを取り入れることが十分に可能であると言える．

6.3.3 オーバーサンプリングへの応用

シミュレーションにおける可視化では，スクリーンに対しボリュームデータのサイズが大きい場合には，サンプリング間隔の調整により全体を見渡したいという要求と，特定部位のみを拡大して詳細に表示したいという要求の双方に対応できるシステムにする必要がある．

提案システムでの部分拡大への対応を考える場合，注目する部分が属するサブボリュームを持つ一部のノードに視線を集中させ，サンプリング間隔を詰めて重畳計算を行う手法が考えられる．しかし，このとき周辺ノードは出力の伝搬のみ行うことになり，一部に負荷が集中しフレームレートの低下が避けられなくなる．

注目する部分のボリューム区間が確定している場合，シミュレーション系から注目する部位のボリュームデータを再度全体に再配信するという対応も可能ではある．

しかし，本節で述べた負荷分散の仕組み利用することで，可視化系のみによってボリュームデータを再配置することが可能になる．各ノードの持つボリューム空間用のメモリの中に再割り当て用の空間をあらかじめ設けておれば，全体のボリュームデータを失うことなく一時的に拡大して可視化するような要求にも応えられる．

謝辞

本研究の機会を与えてくださった，富田 眞治教授に深く感謝の意を表します.

また，数々の有用な御指導，御意見を頂いた，森 眞一郎助教授，中島 康彦助教授，嶋田 創助手，三輪 忍助手に心から感謝いたします.

さらに、日頃の研究において多くの御議論，御協力を頂きました富田研究室の皆様に深く感謝いたします。

参考文献

[1] 對馬雄次, 中山明則, 荻野友隆, 金喜都, 森眞一郎, 中島浩, 富田眞治: ボリュームレンダリング専用並列計算機 —*ReVolver*/C40—, 並列処理シンポジウム JSPP'95, pp. 11–18 (1995).

[2] 原瀬史靖, 山内聡, 津邑公暁, 五島正裕, 森眞一郎, 中島康彦, 北村俊明, 富田眞治: *ReVolver*/C40 を用いた時系列ボリュームデータの実時間可視化, 情報処理学会研究報告 2002–ARC–148, pp. 7–12 (2002).

[3] 生雲公啓: 時変ボリュームデータの実時間可視化のための専用グラフィックスカード VisA の開発 (2003).

[4] 森眞一郎, 小田島大介, 生雲公啓, 五島正裕, 中島康彦, 富田眞治: 960MB/s の DVI-D 入出力リンクと DDR-SDRAM を 2 系統を持つ FPGA 登載 PCI カード — 並列可視化処理への応用 —, 第 11 回 FPGA/PLD Design Conference ユーザ・プレゼンテーション, pp. 31–34 (2004).

[5] K.Ma, J.Painter, C.Hansen and M.Krogh: Pallael Volume Rendering Using Binary-Swap Compositing, *IEEE Computer Graphics and Application*, pp. 59–68 (1994).

[6] Stompel, A., Ma, K.-L., B.Lum, E., Ahrens and Patchett, J.: SLIC:Scheduled Linear Image Compositing for Parallel Volume Renderin, *IEEE Symposium on Parallel and Large-Data Visualization and Graphics 2003*, pp. 33–40 (2003).

[7] 村木茂, 緒方正人, 越塚健児, 梶原景範, 劉学振, 永野靖忠, 下川和郎, Ma, K.-L.: VG クラスタ：スケーラブルビジュアルコンピューティングシステム, Visual Computing グラフィクスと CAD 合同シンポジウム 2001, pp. 85–90 (2001).

[8] 額田匡則, 小西将人, 五島正裕, 中島康彦, 富田眞治: 参照の空間局所性を最大化するボリューム・レンダリング・アルゴリズム, 情報処理学会論文誌：コンピューティングシステム, Vol. 44, No. SIG 11(ACS 3), pp. 137–146 (2003).