修士論文

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

指導教官　　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

生雲 公啓

平成15年2月7日

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

生雲 公啓

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. 中でも実時間の数値シュミレーションの可視化技術は, 大規模な 3 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が目標である. 大規模データの可視化として具体的には 4096^3 のボリュームデータを, 2048^2 のスクリーンに 30fps で出力することを目標とする. このような目標では, シミュレーション系から可視化系へのボリュームデータの転送時間がボトルネックとなる.

我々は環境の実現方法として, 可視化環境への研究資源の投資量に応じて複数の方針を策定しているが, 本稿ではその内, 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.

我々はシミュレーション系から可視化系へ, ネットワークを用いることなくボリュームデータの高速転送行うため, PC クラスタでシミュレーション等を実行し, PC 毎に装備する専用ハードウェアに直接データ転送する方式をとった.

また, 可視化方法として, *ReVolver*/C40 のようにボリュームデータを 3 重化する方式と 3 重化を行わない方式について比較, 検討を行った.

本稿では提案するハードウェア (VisA) について述べるとともに, ボリュームデータが各 PC クラスタにサイクリックに配置されている場合のボリュームレンダリング手法や, Early Ray Termination の効果について述べる.

その後, VisA の主要部分であるピクセル値計算部分の設計を行い, 評価ボードに実装を行った. 実装においては XILINX 社の Virtex-II シリーズの FPGA を用い, メモリモジュールは PC2100 規格 DDR-266 を使用した. Virtex-II に装備されている高速乗算器や, ブロック SelectRAM を積極的に用い, ピクセル値計算部分の最適化を図った. その結果 133MHz での動作を実現させた.

最後に, 我々が今回採用しなかった可視化方式と, 擬似透視投影法についての考察を行った.

Developement of a Graphic Card "VisA" for Real-Time Visualization of Time-Varying Volume Data

Masahiro IKUMO

Abstract

Recently, large-scale numerical simulation is coming to be done in the various fields, caused by the rapid improvement of the computer performance. Real-time visualization technology of numerical simulation is a field that demand advanced technology at a medical field where large scale three-demensional date should be necessary.

Our goal is to built a parallel volume rendering environment to visualize such a large scale date set. Our concrete goal is to visualize 4096^3 data set and output 30 frames per second in the screen of 2048^2 size. In such a goal, it becomes a bottleneck to transfer the large scale volume date from the simulation system to the visualization system.

Now, we are thinking about more than one policy, corresponding to the amount of investment of the research resources to the visualization environment. In this thesis, I describe the way to raelize such environment by using a special hardware for the visualization.

In our system, volume data set is transferd at high speed without network from simulation system to visualization system, bacause simulation is done at the PC cluster and volume data produced by the simulation is transfered to the special hardware which every PC is equipped with.

In this thesis, we describe special hardware "VisA", and volume rendering technique and the effect of Early Ray Termnation in the case that volume data is divided into circulation and assigned to each node.

We designed pixel calculation units which are the main part of VisA, and implement them on FPGA of the Virtex-II series of the XILINX, Inc. using PC2100 DDR-SDRAM266 as a memory module. Finaly, we optimized pixcel calculation units and raised movement frequency to 133MHz.

Finaly, we consider both visualization form which wasn't adopted this time, and perspective projection realized by using parallel projection.

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

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. 中でも実時間の数値シュミレーションの可視化技術は, 大規模な 3 次元データを必要とする医療などの分野 [5] において, 高度な技術が要求されている分野である. これまでも, PC クラスタ等の並列計算機環境を利用した, シュミレーションとその実時間可視化のためのシステムが開発されてきたが, 次世代のシミュレーション技術として, 従来の実験の代替手段となりうる「仮想実験型/仮想体験型のシミュレーション環境」の構築が望まれている. ここでは, 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ている.

我々は, 実時間インタラクティブシミュレーション環境の実現にむけて, 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[1)]を実時間処理する環境について研究を行っている.

我々が目指す実時間システムとは, 毎秒 30 フレームの画像を必ず描画しなくてはならないといった制約を満たされないとシステム自体が止まってしまう所謂ハードリアルタイムシステムではない. 我々は, 制約が満たされなくても可視化スピードが遅くなるだけで, システム自体は止まらないソフトリアルタイムシステムを目指している.

我々は, 並列ボリュームレンダリング環境の実現方法として 3 つの側面を考えている. 処理対象に応じて処理内容を柔軟に変更でき, 処理の高機能化も容易であるソフトウェアによる実現. 比較的安価に手に入る, 汎用グラフィックカードによる実現, 及び超高速を目指した専用ハードウェアでの実現である. 本稿は FPGA(Field Programmable Gate Array) を使った可視化専用ハードウェアの開発とその可視化方法について述べたものである.

本稿では, まず 2 章において研究の背景として我々が検討している実時間インタラクティブシミュレーション環境の構想と, そこにおけるシミュレーション結果の可視化を支援する並列ボリュームレンダリング環境について述べる.

[1)] 4096^3 の 8bit ボリュームデータを SHD 規格相当のスクリーン (2048^2) に秒間 30 枚のフレームレートで出力する可視化システムが当面の目標である.

3 章では可視化専用グラフィックスカード VisA(Visualiztion Accelerator) を利用したボリュームレンダリング環境について，その可視化方法や特徴などを述べ，VisA の備えるべき機能について検討する．

4 章では FPGA で具体的に VisA を開発するための VisA 内のモジュール構成やデータの流れについて述べる．5 章では考察を行い，6 章でまとめる．

第2章　背景

本章ではまず，我々が既に開発してきた *ReVolver*/C40[1] の特徴について説明する．次に現在研究している実時間インタラクティブシミュレーション環境の構成方式，及びそのための可視化環境について述べる．最後に実時間可視化を実現するための並列ボリュームレンダリングシステム環境について3つの視点から述べる．

2.1　Revolver/C40 の概要

本研究の背景として，我々は既にボリュームレンダリング専用並列計算機の開発を行っており，その成果として *ReVolver*/C40 と呼ぶプロトタイプハードウェアを開発しその評価を行ってきた．しかし，次節以降で述べる実時間インタラクティブシミュレーションの可視化のようなシミュレーション系から可視化系へのデータ転送に対する要求が厳しいシステムには適さない．

しかし実時間可視化を実現する *ReVolver*/C40 のアーキテクチャは，新システムの構築においても有用なものであると考える．以下ではボリュームレンダリングとは何か，及び *ReVolver*/C40 の特徴について述べる．

2.1.1　ボリュームレンダリング

ボリュームレンダリングは、ボリューム空間からサーフェイスを算出してレンダリングを行うサーフェイスレンダリングとは異なり、ボリュームの色や透明度を直接スクリーンに射影する手法である．ボリュームレンダリングにおいて、可視化の対象となる空間をボリューム空間と呼ぶ。ボリューム空間には、科学技術計算の結果や自然現象などのデータセットが半透明のボリュームとして色や透明度などに変換されて置かれる。

ReVolver/C40 では、ボリューム空間は立方体であり、ある頂点を含む3辺がワールド座標系における各座標軸 (x,y,z 軸) の正の方向にあるように設定される。このボリューム空間を単位立方体に分割し、離散的にデータを与えたものをボクセルと呼び、各ボクセルには色や透明度の値 (ボクセル値) が与えられる。また、スクリーンはピクセルと呼ばれる単位正方形の集合として捉えられる。

2.1.2　*ReVolver*/C40 の特徴

ReVolver/C40 は，医療画像生成だけでなく，科学技術計算の解析や流体の可視化をも行える専用並列計算機である．この目的を達成するためには，半透明

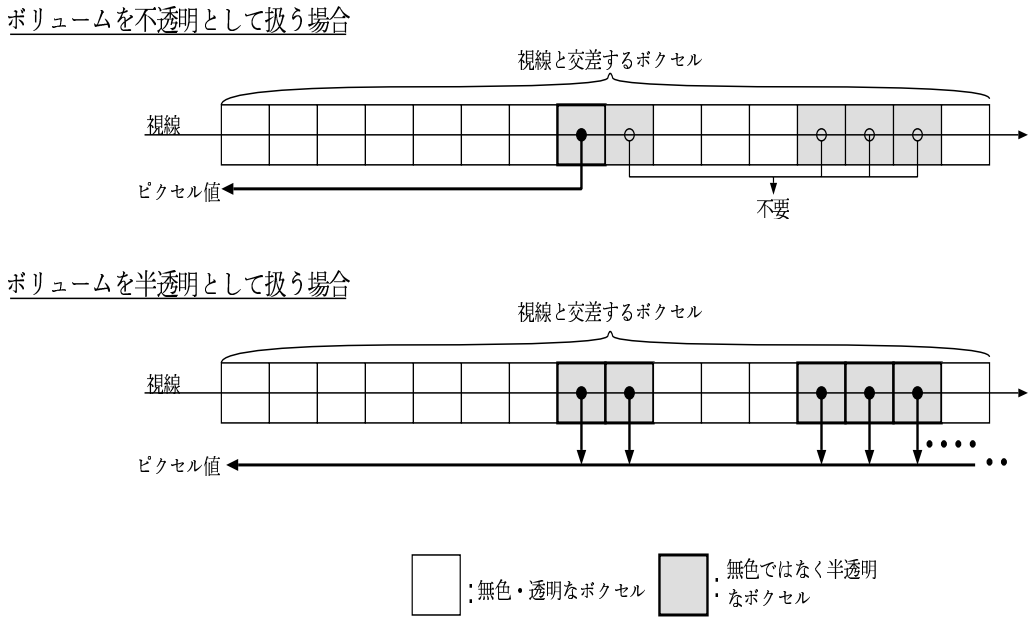


図1: 参照するボクセルの違い

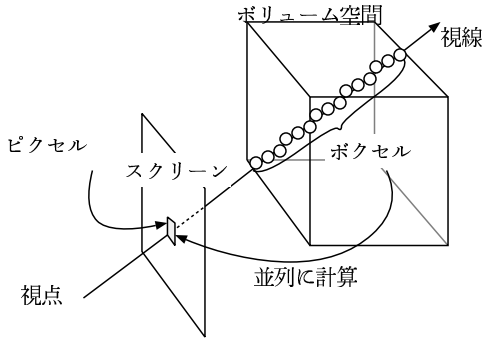


図2: レイキャスティング法

ボリュームの表示，遠近法による画像生成，高速描画という 3 つの要件を満足する必要がある．ボリュームを半透明なものとして扱うには，図1のようにピクセル値を計算する際に視線と交差する無色透明でない全ボクセルを順にアクセスする必要がある．

また遠近法を採用する場合，任意方向の視線が発生するため，メモリアクセスが規則的ではなく，メモリバンクへのアクセス競合の可能性が高くなり，並列処理への障害となる．ここでは，それらの問題を解決し，高速描画を行うために *ReVolver*/C40 が持っている特徴を述べる．

レイキャスティング法の採用　視点からスクリーン上のあるピクセルの中心へ

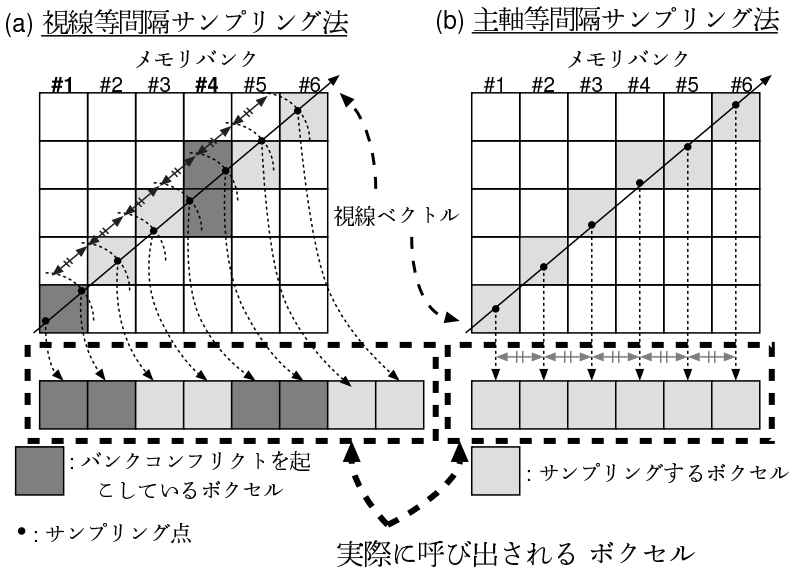


図 3: サンプリング方法の違い

と向かうベクトルのことを視線ベクトルと呼び，視点を一端としピクセルの中心を通る半直線を視線と呼ぶ．図 2 に示すようにレイキャスティング法ではスクリーン上のピクセルごとに，視線がどのボクセルを通過してきたのかを追跡して，ピクセルの色や明るさを決定し，スクリーンに投影する．レイトレーシング法と同じ考え方であるが，反射や屈折のことは考えない．

サンプリング方法の単純化　従来のサンプリング方法では，視線の方向に対して等間隔にボクセルをサンプリングしてきた．この方法を視線等間隔サンプリング法と呼ぶ．

ReVolver/C40 では，視線ベクトルの成分の絶対値のうち，最大値をも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方法を採用している．この方法を主軸等間隔サンプリング法と呼ぶ (図 3(b) 参照)．この方法により，次に述べるバンクコンフリクトが生じない．

ただこの方法では，視線が軸と平行でない時にサンプリング間隔が従来の視線等間隔サンプリング法 (図 3(a) 参照) と比べて大きくなるという性質がある．このサンプリング間隔の補正はピクセル値の計算時に行うことで対処する．

バンクコンフリクトのないメモリ構成　主軸等間隔サンプリング法では，ある

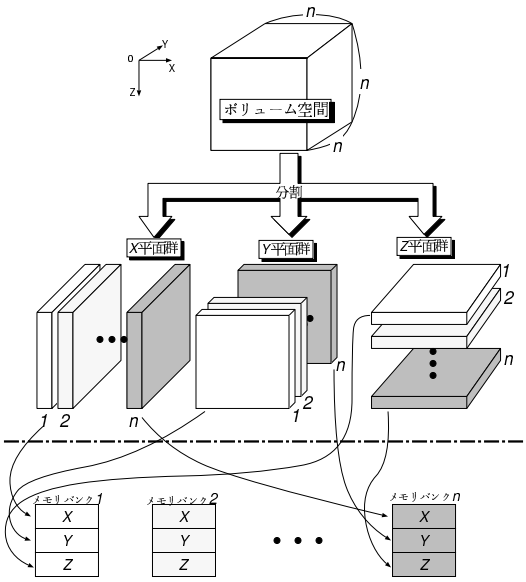


図 4: ボリュームデータの格納方法

座標軸に垂直な slice にボリュームデータを分割してメモリバンクに格納すれば，その座標軸を主軸として持つ視線に関してはバンクコンフリクトフリーとなる．しかし，任意方向の視線の主軸が，この座標軸を主軸として持つとは限らない．そこで，x,y,z 軸のいずれにも垂直な slice が存在するように分割すれば任意方向の視線に対してバンクコンフリクトフリーとすることができる．

ReVolver/C40 の 3 次元メモリ構造では，図 4 のように各平面群に対して，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，ボリュームデータを三重化し，格納されることになる．

この構成により，各平面群の平面は全て別のメモリバンクに格納されることになり，主軸等間隔サンプリングを行うことにより，視線が任意方向であっても，バンクコンフリクトが生じないことを保証する．

ピクセル値計算処理での並列処理　一つの視線に対するボクセル並列のピクセル値計算処理は一次元のパイプライン構成をとることによってその処理を並列化することが可能である．それぞれのプロセッサは視点に近いボクセルから順にピクセル値を計算し，自分の処理が終わると次のプロセッサに

計算結果を渡す．プロトタイプではプロセッサは128台，最終構成のフルシステムでは512台使用する．

透視投影　*ReVolver*/C40では投影法として並行投影だけでなく，投資投影もサポートしている．

- 並行投影
  視点を無限遠におくことで，視線を一方行に固定した投影法．この方法は非常にアルゴリズムが簡単であり，計算量も少ないという利点があるが同じ大きさの物体が異なる距離にあっても同じ大きさに見えてしまうという欠点がある．透視投影のように物体が歪んで見えたりはしない．
- 透視投影
  視点から視線を放射線状に出す投影法で，距離が異なると同じ大きさの物体がを異なる大きさで正しく見ることができる．しかし，視線の中心から外れると，物体の形が歪む欠点もある．

透視投影で描画したほうが人間が見るためには自然な画像になる．しかし，並行投影する場合はの視線ベクトルが全て同じ方向であるのに対して，透視投影では全ての視線ベクトルが違うベクトルであるため視線生成のための計算量が多くなってしまい，描画速度の面から考えると不利である．

2.2　実時間インタラクティブシュミレーション環境

本節ではまず，シミュレーションと可視化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にあたり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極めて稀である．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 (インタラクティブ性)，それに伴って可視化システム側ではそのような時々刻々と変わる状況を実時間で可視化しなくてはならない．

また，インタラクティブ性のないシミュレーションにおいても初期パラメー

タの設定ミスなどのような少し計算を進めればすぐ分かるミスも，これまでの可視化システムでは計算終了まで待たなければ分からなかった．

実際，これまでのシステムは，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いた．そのため，可視化を行う部分と可視化対象となるデータを生成する部分とがはっきり分けられており，計算サーバなどのデータ生成系で行った計算の最終結果をネットワークなどで，グラフィックスワークステーションやパソコンなどの可視化専用システムに転送して可視化するものが一般的であった．また，データ生成系と可視化系が別のシステムであったため，シミュレーション側のデータ形式と可視化側のデータ形式の互換性や，システムの操作性などの様々な問題があった．

以上のような点から考えると，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は，観測や計算をその時々でリアルタイムに可視化する可視化システムが必要不可欠であるといえる．

2.2.2 リアルタイム可視化システム

本項では，リアルタイム可視化システムの利用形態を述べ，それぞれに必要となる仕様を述べる．

リアルタイム可視化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 (以後「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なボリュームデータの実時間の可視化 (以後「オンライン可視化」と呼ぶ) がある．以下では，オンライン可視化とオフライン可視化についてもう少し詳しく述べる．

- 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，可視化の対象が時不変ボリュームデータである場合の利用形態である．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は頻繁でなく，両者が比較的粗に連携している場合である．オフライン可視化では，対象を何度も可視化して解析することが考えられるので，可視化系はできるだけ高精細かつ高速な画像生成とインタフェースの実時間応答性，操作性が求められる．また，あらゆる方向からの視点への対応など，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
  新しく必要となりつつある，シミュレーションの途中経過の可視化など，可

視化対象が時変ボリュームデータの場合の利用形態である．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，できるだけ高速な可視化対象データの受渡しが求められるため，データ生成系と可視化系との間の接続には高い性能が求められる．

また，オフライン可視化のように同じデータを何度も可視化して，解析することもあるため，それに対応できる高速な描画も必要である．また，視点を移動させながら途中経過の観測などはあまり考えられないので，可視化の際のパラメータに対する要求は低い．

以上より，これら 2 つの利用形態に対応するためには実時間可視化が可能な高速な画像生成に加えて，データ生成系から可視化系への高速なデータの受け渡しが重要であるといえる．

2.2.3 実時間インタラクティブシミュレーション環境の構成方式

これまでの議論より可視化系とデータ生成系の連携が重要であることが分かった．システム構成としては以下の方法が考えられる．

方式 1 まず考えられるのが，これまでの可視化システムと同じように，計算サーバにデータ生成系を置きそれとは別に可視化専用のグラフィクスワークステーションのようなものを設けそこに可視化系を置く方式である．

このような方式では，可視化系を専用に設けるため，オフライン可視化で求められる可視化系の性能を満たすことは容易であるが，オンライン可視化で求められるデータ生成系と可視化系との間の接続に対する性能を満たすことが難しくなる．

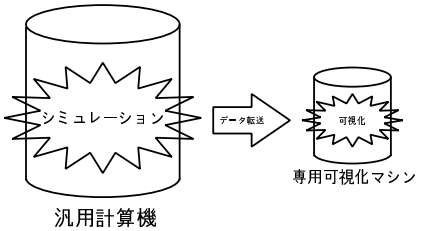


図 5: 方針 1

方式 2 オフライン可視化で求められるデータ生成系と可視化系との間の接続に対する性能を満たすためには，データ生成系と可視化系を別々に設けるので

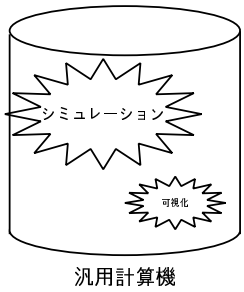


図 6: 方針 2

はなく，例えば同じ汎用の大型計算機内に設けるといった方式も考えられる．

この方式では，両者の接続に対する性能は容易に満たすことができるが，可視化に特化したシステムではないため，可視化系の性能を満たすことが問題となる．

2.2.4 我々の構想

システムの構成方針としては，2.2.3 で述べたのように可視化系とデータ生成系を分ける方法 (方針 1) と一体とする方法 (方針 2) がある．我々は，4096^3 のデータを 2048^2 のスクリーンに $30fps$ で表示することを目標としている．ボリュームレンダリングでは，1 つのボリュームデータ当りの演算量は比較的少なく，描画速度を支配するのはボリュームデータへのアクセスバンド幅と考えて良い．上記の目標を達成するために必要となるメモリバンド幅は $480GB/sec$ となる．ボリュームデータの転送に関して考えた場合，方式 2 のようにデータ生成系と可視化系を一体とするアプローチをとるべきである．

そこで我々は PC クラスタをシミュレーション母体とし，そこで生成されるボリュームデータを PC クラスタ内で可視化することでメモリバンド幅の要求を満たし，上記の目標を達成することを考える．

2.3 並列ボリュームレンダリング環境

本節では並列ボリュームレンダリング処理について述べ、並列ボリュームレンダリング処理を実現する方法として 3 つの視点から考える。

2.3.1 並列ボームレンダリング処理

大規模シミュレーション結果及びその過程の可視化においては, 並列計算機の各ノードで生成された膨大なデータを, 一旦収集した後に可視化するといった従来の可視化手順では実時間性の確保が困難である. したがって, 実時間可視

化を実現するためには, 生成されたデータに対して, そのノードが可視化に必要な処理を行い, 最低限の情報のみを全ノードから収集／処理してユーザに呈示する仕組みが必須である. 以下では, 3 次元データの可視化手段の 1 つであるボリュームレンダリング処理を対象として可視化システムの実現方法について検討する.

ボリュームレンダリングでは，シミュレーションにより得られた 3 次元空間上の数値データを, 色 C と透明度 t に対応づけることで, 3 次元空間内部のデータの分布状況を可視化する. 具体的には, まず視点からスクリーンの各ピクセルに対して視線を出し、視線上のボクセルの値を視点に近い順に $v_o, v_1, v_2, \cdots$ とするとピクセル値は次の式（畳み込み演算）で計算される.

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

$$T_k = \prod_{i=0}^{k} t(v_i) \tag{2}$$

ここで $c(v_i), t(v_i)$ はそれぞれボクセル値 v_i を，色，透明度に変換した値であることを示している．さらに，式 (1)，(2) は以下のような漸化式に変換できる。

$$C_k = C_{k-1} + (1 - t(v_i)) \cdot c(v_k) \cdot T_{k-1} \tag{3}$$

$$T_k = t(v_k) \cdot T_{k-1} \tag{4}$$

この畳み込み演算には、演算区間をいくつかの部分区間に分割し, それぞれの区間に対する計算結果に対して, 再度畳み込み演算を行うことで結果を得ることが可能であるという性質がある．そこで, 1) シミュレーションサーバの各ノードが, 自身が生成したデータに対する部分 3 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, 2) そこで得られた画像とボクセル毎の透明度を, 視点からの距離の順番に従って合成する, という手法で並列処理が可能である.

2.3.2 3 つのアプローチ

このようなボリュームレンダリング処理の実装に際しては, 可視化処理に対してどれだけの投資が可能かに応じて, いくつかの方法が考えられる。まず、可視化用のハードウェア・アクセラレータを用いるか、用いずにソフトウェアのみで行うかという選択がある。また、ハードウェア支援を行う場合においても、テ

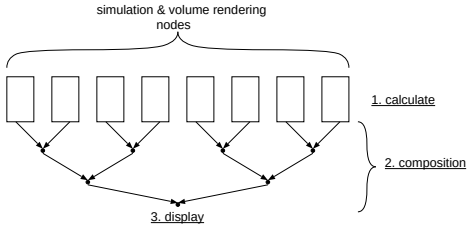


図 7: 並列ボリュームレンダリング処理の概略

クスチャベースのボリュームレンダリング (Texture Based Volume Rendering) を行う汎用グラフィックスカードを用いるか, あるいは直接ボリュームレンダリング (Direct Volume Rendering) を行うボリュームレンダリング専用のハードウェアを実装するかの選択が可能である. 我々は, このいずれのケース対しても対応可能な可視化支援環境の構築を目指している.

以下、ソフトウェアを用いる場合と、汎用グラフィックスカードを用いる場合について簡単に述べ、その後、本稿の主題である専用ハードウェアを用いた可視化システムについて関連研究を挙げながら述べる。

2.3.3 ソフトウェアによる可視化

ソフトウェアによる方法では, シミュレーションに用いたプロセッサを可視化処理にも利用するため, 可視化のための新たな投資が必要ないという利点があるが, 可視化処理もシミュレーションも同一の CPU で行われるため、可視化処理を行っている間は, シミュレーションが止ってしまうという欠点がある.

各ノードで生成したサブボリュームに対する中間画像データ (色、及び透明度) ができると, これをもとに最終合成画像を生成する. 多くの場合, この合成処理は各ノードが保持するサブボリューム間の隣接関係に基づく 2^N 進木構造の合成ネットワークを構成してパイプライン的に処理を行う. ネットワーク自体はソフトウェアによって構成するため、柔軟な構成をとることができるという利点がある。

各ノードでの可視化処理において, 視点, スクリーン, サブボリュームの相互の位置関係を考慮したキャッシュ・ブロッキング手法を用いた最適化, ストリーミング SIMD 命令等を活用した最適化, キャッシュバイパスやプリフェッチ等のメモリ最適化, さらには, Shear-Warp[3], 早期視線終端 (ERT) 等のアルゴリズムレベルの最適化を行うことで専用のハードウェアにせまる性能を出すことも不可能ではない.

ソフトウェアによる手法では, レンダリング処理時間が大きくなる傾向があるが, その反面, 非均質構造格子への対応等の柔軟な処理が可能である. ただし, この場合は可視化パラメータに依存した負荷不均衡を解消するための動的負荷分散処理等が必要になる [6].

2.3.4 汎用 3 次元グラフィックスカードを用いた可視化

ソフトウェアによる並列ボリュームレンダリングにおいて, 計算サーバの各ノードのプロセッサが担当していたサブボリュームに対するレンダリング処理を, 汎用の 3 次元コンピュータグラフィックスカード (3DCG カード) に行わせ, 3DCG カードのハードウェアアクセラレーション機能を活用して高速化を図る方法である.

ここでは, ボリューム空間を座標軸に垂直なスライスの重ね合わせで表現し, それらを半透明テクスチャとしてポリゴンにマッピングしたものを α ブレンディングすることによってボリュームレンダリング処理を行う [7].

3DCG カードがレンダリング後, 2.3.3 章と同様に個々のレンダリング結果をネットワークを介して合成する. 一般に 3DCG カード自信は通信機能を持たないため, 合成のためのノード間通信においては, 計算サーバの CPU を介した処理が必要となる.

現在, 汎用3DCG カードのグラフィックスメモリサイズは高々128MB 程度であり, サブボリュームサイズよりもはるかに小さいということが考えられる. また, テクスチャサイズの上限がボリュームスライスのサイズより小さいこともあり得る. このような場合, ノード内のレンダリング処理においてサブボリュームをサブサブボリュームに分割し, それに対してレンダリングした後に結果を合成して, サブボリュームに対するレンダリング結果を得るといった処理が必要となる.

現時点ではまだ十分な考察が行えていないが, Pentium4 の 2.2GHz、主記憶 512MB(PC800), 3DCG カード GeForce4 Ti 4600 (Graphics Memory: 128MB) を用いて, $128 \times 128 \times 69$ のボリュームデータに対して, 512^2 の画像を生成させたところ, データ転送時間を除くボリュームレンダリング処理に 30.01msec を要することが分かっている. また, データ転送に関しては, テクスチャメモリへの書き込み[1)]に要する時間は 1MB あたり 1.84 msec と AGP4X の最大転送性能

[1)] OpenGL の glTex3dImage() を利用

に近い値を示しているのに対し，フレームバッファからの読み出し[1)]や書き込み[2)]に要する時間は, それぞれ, 1MB あたり 6.16msec ならびに 82.02 msec と低速であることが分かっている [11].

2.3.5 専用ハードウェアを用いた可視化

4000^3 程度の大規模な 3 次元データの可視化について考えると、従来の AVS 等に代表される汎用計算機を用いた可視化では、使用する計算機のメモリ性能の制限からリアルタイム応答性が低い、もしくは非常に大規模な汎用計算機が必要になるなど、設置面積だけを考えてもコストパフォーマンスに問題がある。また我々のシステムの対象ユーザとしては研究室のような単位を考えており、このようなユーザが、汎用の大型計算機を占有することは難しい。

我々は 2.2.4 節で述べたように、PC クラスタをシミュレーション母体として、PCI バスにボリュームレンダリング専用のハードウァアを付加した構成をとる。

このような PC クラスタに専用ハードウェアを付加する方式を採用している実装例として、VGCluster があげられる。VGCLuster ではボリュームレンダリングの並列化の手法として、ReVolver/C40 のようなパイプラインでボクセル並列処理を行うものとは異なり、一つの視線をボリューム並列処理する方式を採用している。

VGCluster では、PCI カード型のボリュームレンダリング専用ハードウェアである VolumePro を搭載した PC で PC クラスタを構成し、PC クラスタ部でシミュレーションを実行し、可視化を VolumePro を用いて行う。VolumePro は 512^3 のデータをリアルタイムでボリュームレンダリングを実行できるハードウェアで、VGCluster ではさらに大規模なデータの可視化を、各スクリーンに対する処理を 512^3 ごとのボクセルに分割し、各 VolumePro に割り当て、最終結果を 1 つのスクリーンに重ね合わせて表示することで実現している。VGCluster ではこの重ね合わせのための専用ハードウェアが必要になっている。

また、メモリの大容量化の恩恵を受け、ワークステーション・パーソナルコンピュータのアドオンボードとして 256^3 程度のボリュームデータの可視化するシステムは既に開発がなされているが、リアルタイム表示をする際には視点や視角などに制限がある。更に、4000^3 程度の大規模なボリュームデータを扱うにはメモリ容量、メモリバンド幅共に不十分である。

[1)] OpenGL の glReadPixels() を利用

[2)] OpenGL の glDrawPixels() を利用

そこで我々は、ReVolver/C40 で採用した 3 次元メモリ構造とキャッシュシステムに併用により、4000^3 程度のボリュームデータに対するリアルタイム可視化を実行することを目標とする。また我々のシステムでは ReVolver/C40 と同様に、医療画像生成だけでなく科学技術計算の解析や流体の可視化なども行えるシステムにするため、半透明ボリュームの表示、遠近法による画像生成といった要件も満足させる。

本システムにおけるピクセル値計算は、*ReVolver*/C40 と同様に 1 つのピクセルの輝度の計算を複数のプロセッサで処理するボクセル並列処理という方式を採用している。この方式はレイキャスティングアルゴリズム (2.1.2 節参照) を視線方向に並列化したものである。

この方法では、スクリーン上の各ピクセルごとに発生する視線に沿って、視線と交差するボクセルを視点に近い順にサンプリングしていく。これを式 (3) を用いて視線上のボクセルがなくなるまで繰り返し、ピクセル値を求める。サンプリングするボクセル数が N の場合、ピクセル値は式 (3) の C_N と同一の値となる。つまり C_N を求めることによってピクセル値を求めること出来る。

式 (3) の C_k, A_{k-1} は $C_{k-1}, A_{k-1}, c(v_k), \alpha(v_k)$ から求めることができ、$c(v_k), \alpha(v_k)$ は v_k を変換した値であるから、結局 C_K, A_K は C_{K-1}, A_{K-1}, v_k から求めることができる。

第3章　専用ハードウェアを用いた可視化システム

本章では2.3.5節で述べた専用ハードウェアを用いた可視化システム環境の特徴についていくつか述べ，効率のよい実装方針について考えていく．

3.1　ピクセル値計算

本システムでは1次元に接続したi(i = 0,1,⋯,m-1)番目のノードがそれぞれm分割したサブボリュームの1つをローカルメモリに持ち，C_N の漸化式計算の中の，C_x から C_y の部分の計算を行う．ここで $x = \frac{iN}{m}$, $y = \frac{(i+1)N}{m} - 1$ である．この構成でパイプライン動作をさせることで，ピクセル値計算の高速化を図っている．

3.1.1　視線方向が負の場合

視線の主軸成分が正(以下,「視線方向が正」と呼ぶ)の時と，視線の主軸成分が負(以下,「視線方向が負」と呼ぶ)の時とでは，処理を行う方向が逆になる．よって例えば視線方向が正のピクセル値の計算を行っている最中に視線方向が負のピクセル値計算を行おうとすると，処理の衝突が起こり，性能が低下する．そのため，常に処理の方向が正となるように回路を実現させる必要がある．

まず式(3)は次の式(5)のように変形することができる．

$$D_{k-1} = D_k \cdot t(v_{k-1}) + (1 - t(v_{k-1})) \cdot c(v_{k-1}) \tag{5}$$

式(3)によって求められるピクセル値C_Nは，式(5)のD_0と同一の値となる．式(5)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，結局D_k, v_{k-1}から求めることができる．

$C_N = D_0$ の証明　$C_N = D_0$ となることは直観的に分かりにくいため，ここで簡単に証明する．ここでは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．そこで，$T_{-1} = 1, C_{-1} = 0, D_{N+1} = 0$ とすると式(5)より

$$\begin{aligned}
D_0 &= D_1 \cdot t(v_0) + (1 - t(v_0)) \cdot c(v_0) \\
&= D_2 \cdot t(v_1) \cdot t(v_0) + (1 - t(v_1)) \cdot c(v_1) \cdot t(v_0) + (1 - t(v_0)) \cdot c(v_0) \\
&\quad \cdots \\
&= D_{N+1} \cdot t_{N-1} \cdot t_{N-2} \cdots t_0 + (1 - t(v_N)) \cdot t(v_{N-1}) \cdot t(v_{N-2}) \cdot t(v_{N-3}) \cdots t(v_0)
\end{aligned}$$

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

となる．また式(3)より，

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

これより$T_{-1}=1, C_{-1}=0, D_{N+1}=0$を代入すると，$C_N=D_0$であることが証明できる．

結局，視線方向が負の場合にi番目のプロセッサに式(5)のD_{N-i}の計算を行わせると，ボリューム・データがN^3ボクセルとすると，0,1,…，N番のプロセッサの順で処理を行うことができる．

以上より，視線方向が正の時は式(3)を計算し，視線方向が負の時は式(5)を計算することで，常に処理の方向を単方向とすることができる．しかしこの方法でパイプライン方式で計算を進めていくためには，式(3)を計算する回路と式(5)を計算する回路両方を用意してかなくてはならない．実際*ReVolver*/C40ではDSPのプログラムで視線方向に従って命令を選択するようになっている．

3.1.2 逆向きピクセル値計算回路の削減

本システムでは各ノード内に複数のピクセル値計算ユニット(PCU)が装備されており，処理の流れを単方向にすることを考えた場合，ノード間の処理の流れを単方向にすればよく，ノード内での計算の流れはノード単位での処理と統一する必要はない．つまり，視線方向が負の場合は，ノード内での計算は視線方向が正の場合と同様に，つまり同じ回路で計算を行い，ノード間に伝播するピクセル値の途中結果とそのノードで計算されたピクセル値とを計算する時のみ，式(5)の回路で計算するようにすればよいことになる．その様子を図8に

示す.

図 8 の上下の 2 つの画像はどちらも，視線方向が負の場合のデータ及び通信の流れをそれぞれ直線，点線で示している．各ノードは左隣りのノードから得た視線情報を用いて自ノード内で処理を行ない，生成した結果を右隣りのノードに転送している．図 8 の上図は自ノード内の処理を全て式 (5) を用いて行った場合を示している．図 8 の下図は先ほど述べた，ノード内で主に式 (3) を用いる方法を表している．後者では計算結果を転送する際に式 (5) の式を用いて補正を行う必要があるが，その処理は図 8 にもあるようにノード内で一回のみである.

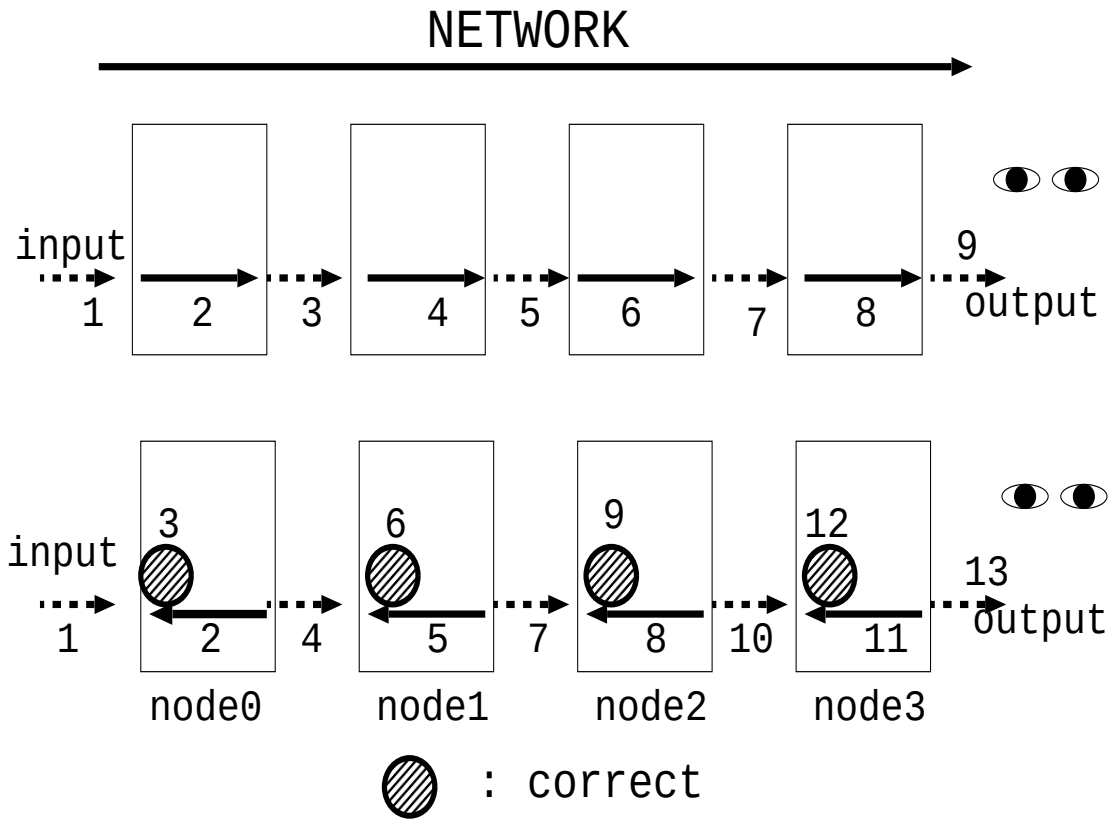


図 8: ピクセル値計算の順序

本来ならば視線方向が正負に応じてそれぞれ式 (3)，(5) に対応する別々の回路を用意する必要があった．しかし，各ノードに補正のための式 (5) の回路を一段分だけ用意することで同じ処理をすることが可能となった.. 結果として両方の回路を用意した時と比べると，ボリューム空間のサイズが N^3, ノード数が

m, とすると回路面積は $\frac{N+m}{2N}$ となる．$N \gg m$ だとすると，回路面積は約 $\frac{1}{2}$ にすることができる．

ReVolver/C40 においても同様の議論が成り立つ．しかし *ReVolver*/C40 では N^3 のボリューム空間を N 枚のスライスに分割して並列処理を行ってる．さらに 1 スライスごとに 1 プロセッサを割り当てるという構成になっているため，N = m と考えられる．そのためここで述べた手法を用いても効果がないといえる．

一方我々は N = 4096, m = 128 を考えており，その比率からこの手法は十分有効であるといえる．

3.2 Early Ray Termination の効果

シミュレーション系における負荷分散形態の代表的なものとして、ブロック分割とサイクリック分割がある。一般にサイクリック分割を行うと、ブロック分割に比べて負荷が均質なものになる。しかしデータの可視化を考えた場合、サイクリック分割では通信回数が増加し、描画速度が遅くなる。

従来の *ReVolver*/C40 での可視化では、プロセッサが一次元配列構造で構成されており、最後尾のプロセッサから先頭のプロセッサへの帰還パスが存在しなかった。そのためサイクリックにデータが配置されることは考慮にいれる必要がなかった。しかしシミュレーションとの連携を重視した可視化システムでは、シミュレーション側で一般的に行われるサイクリックなデータ分割に対応した構成にすべきである。

VisA ではノード間ネットワークを環構造にして、サイクリック分割に対応している。また我々は、Early Ray Termination(ERT) 手法 [9] を用いて通信回数の増加に伴う描画速度が低下する問題を少しでも緩和したいと考えている。そこで本節ではサイクリックな負荷分散形態における ERT の有効性について検証する。ERT の効果はデータ依存性が高いため、検証においては具体的にいくつかのボリュームデータを例に挙げ、定量的に行う。

3.2.1 ERT とは

ボクセル (3 次元立方格子) で構成されたボリューム空間でのボリュームレンダリングは，図2 のようにスクリーンの 1 ピクセル毎に生成された視線上にあるボクセルの色をそのボクセルの透明度に従って，視点のほうから積算していきそのピクセル値 (R,G,B,T) とする．ERT とはこのピクセル値計算の際に，透明度があらかじめ設定した基準値を下回ったら，それ以降のボクセルからの影

響はごく小さいため，そこで計算を打ちきってしまうという手法である

3.2.2 対象とする並列ボリュームレンダリングアルゴリズム

本検証では，N 枚の N^2 サイズのスライスからなる N^3 のボリュームデータを P 台のプロセッサに $\frac{1}{P}$ づつ分解し、各プロセッサは，担当する $M(=\frac{N}{P})$ 枚のスライスに対して，レイキャスティングアルゴリズム (Front-to Back) に基づき，ピクセル値計算のための P 段のパイプラインを搭載し並列処理を行うことを考える．ピクセル P(i,j) に対するピクセル値計算の流れを図 9 に示す.

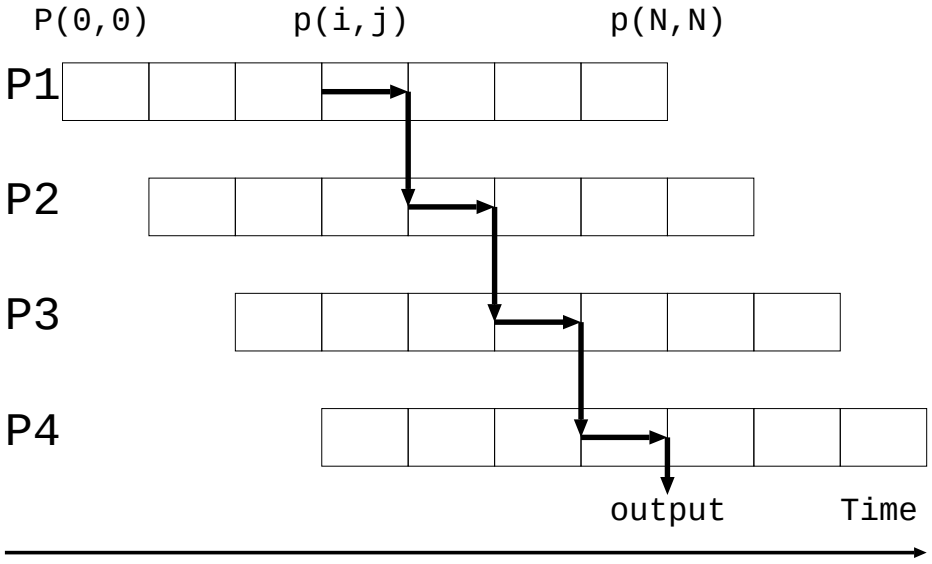


図 9: ピクセル P(i,j) に対するピクセル値計算の流れ

3.2.3 サイクリック分割における ERT

今, P = N の場合に 1 フレーム分の処理に要する時間を通信時間 C と計算時間 T の和として表現すると, ある $P(< N)$ に対して幅 $\frac{M}{m}$ のブロックサイクリック分割時 (以下，m 回サイクリック[1)] と呼ぶ。) に描画に要する時間 $T_{M,m}$ は,

$$T_{M,m} = (C + \frac{M}{m} \times T) \times m \tag{6}$$

となり, ブロック分解 (m=1 に相当) に比べて$C \times (m-1)$だけ処理時間が増加する.

[1)] 図 10 の BLOCK,CYCLIC,BLOCK & CYCLIC はそれぞれ m = 1, m = 4, m = 2 である

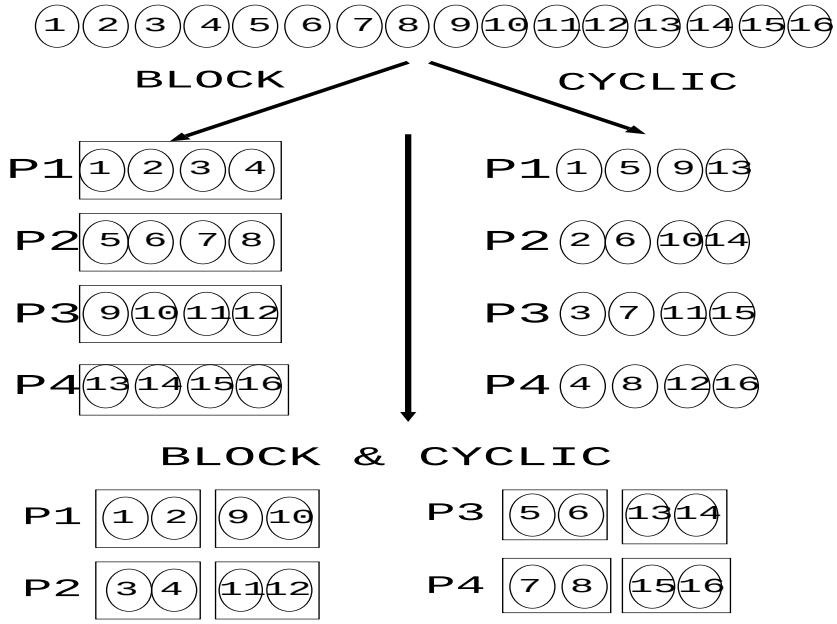


図 10: データのブロック分割とサイクリック分割

次に ERT を適用する。E_i%の視線が $\frac{i}{m}$ までのサンプリングで Terminate すると，描画時間 $T_{M,m,Ei}$ は，$E_0 = 0$ とすると，

$$T_{M,m,Ei} = (C + \frac{M}{m} \times T) \sum_{i=1}^{m} i \times \frac{Ei - E_{i-1}}{100} \tag{7}$$

となる.

ここでもし E の値と $\frac{T}{C}$ の値がある程度大きくなれば，プロック分割の場合に比べて高速化が図れると考えられる.

3.2.4 検証

本節では前節で述べた, サイクリック負荷分割形態における ERT の効果を, 実際にいくつかのボリュームデータ (256^3) に適用して検証を行った.

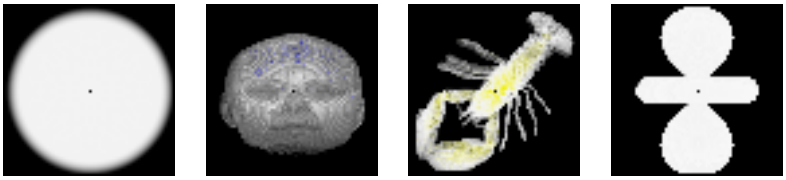
各ボリュームデータに対して，透明度が 0.05 以下になったらピクセル値計算を打ち切るという方針で，それぞれ E_i を計算した．サイクル数 m=8 の時の結果を表 1 に示す.

使用したデータは次の 4 種類である.

- sphere

  中心が (128,128,128) で半径が 128 の球．透明度は中心が 0.8 で最も不透明で徐々に透明になっていき球の表面では 1.0 になっている[1)].

[1)] 透明度の値は 0.0(不透明) から 1.0(透明) までの値をとる



(a)sphere (b)Head (c)lobster (d)hydrogen

図 11: 使用したボリュームデータの概観

- Head
  人の頭部のボリュームータである。対象はボリューム空間全体に広がっている。透明度は全て 0.5 である.
- lobster
  物体を構成している部分の透明度は 0.8 から 1.0 の間で変化する.
- hydrogen
  物体を構成している部分の透明度は 0.9 から 1.0 の間で変化する.

全ての画像とも物体の外側の部分 (黒い空間) の透明度は 1.0 になっている.

表 1: m = 8 の時の E_i の値 (%) $E_8 = 100$

	E_1	E_2	E_3	E_4	E_5	E_6	E_7
sphere	0.00	0.96	24.99	41.34	50.22	53.40	53.50
Head	0.00	1.90	31.13	41.53	45.45	46.53	46.53
lobster	0.00	0.00	0.00	1.00	5.17	5.17	5.17
hydrogen	3.15	9.45	26.15	40.31	40.84	41.02	41.02

次に *ReVolver*/C40 での実測データ C=164msec,T=207msec[2] を適用して，プロセッサ数 P を変化させ[1)]計算した結果を表 2 に示す．lobster の画像で効果が少なかったのは、lobster 自体が薄く、入射した視線がすぐに体内を通過してしまうため、不透明になるピクセルが非常に少なかったためだと考えられる。他の 3 つは ERT の効果が現れて、実行時間が短縮している.

M の値をある程度大きくし，ERT を用いることで，データをサイクリックに分割することで生じる通信時間の増加による描画速度の低下を解消できることが分かった．またレンダリングの対象となるボリューム空間に，ある程度広範

[1)] 画像の大きさは全て 256^3 のため $M = \frac{256}{P}$ である

表 2: m=1 の時に対する実行時間比 (%))(C:T=164:207)

	P = 8 (M = 32)				P = 16 (M = 16)			
m	1	2	4	8	1	2	4	8
sphere	100	81	81	84	100	83	86	95
Head	100	81	83	85	100	82	88	97
lobster	100	101	105	114	100	104	112	130
hydrogen	100	81	82	87	100	83	88	99
ERT 未使用	100	102	107	116	100	104	114	133

囲にボリュームデータが存在すれば，サイクリックに分割しない場合よりも数%から 20%近くの速度向上が見られることが分かった.

3.2.5 考察

シミュレーションと連携させた可視化システムにおいては，その膨大なデータの転送時間が問題となるため，うまく負荷分散して並列に処理しなくてはならない．負荷分散の一形態であるサイクリック分割では，通信時間が増加して描画速度が低下することが懸念されるが，ERT を用いることで，画像をほとんど劣化させることなくより高速な描画が可能であることが確認できた.

ERT を用いて，サイクル数 m の値を大きくすることで通信量の増加を抑制することはできたが，今回使用したボリュームデータのように，半数近くの視線が Terminate することなく最後まで計算されるため, サイクル数を多くしても劇的に高速になるということはない．今回のボリュームデータでは，サイクル数は 2 程度が妥当であると思える.

なお, 今回は *ReVolver*/C40 での実測値をベースに議論を行ったが，PC クラスタ等での並列ボリュームレンダリングで同様のサイクリック分割を行うと, 一般に $C >> T$ であるため, M, E を相当大きくとって初めて有効性が確認できる．また，そのような環境においては，視線を複数同時に処理し，通信回数を減らすことが考えられる．その場合，どのように視線を組み合わせて処理をするかが重要な問題になってくる．多くの視線を同時に処理すると，通信回数は大きく減少するが，その分 E の値が低下してしまう．E の値が低下してしまうと今回の lobster の様に，ERT の効果が望めなくなってしまう.

3.2.6　本システムへの適用

データをサイクリックに分割した際のERTの効果については前節までに示した．本システムにおいてもシミュレーションから生成されるボリュームデータが，各ノードにサイクリックに分割されている場合は，この手法を用いて処理の高速化を図る．ところでEarly Ray Terimination の手法は，3.2.2 項で述べたように視点からスクリーンへと向かう視線に沿って透明度の計算を行い，計算途中のピクセルの透明度がある基準を下回った時点で計算を打ちきるという手法なので，直感的には前節で示したような視線方向が負の場合には適用することができないと考えられる．しかしデータがサイクリックに分割してある場合は，計算の仕方を工夫することで，視線方向が負の場合でもEarly Ray Termination 手法を適用することができる．以下，その方法について述べる．

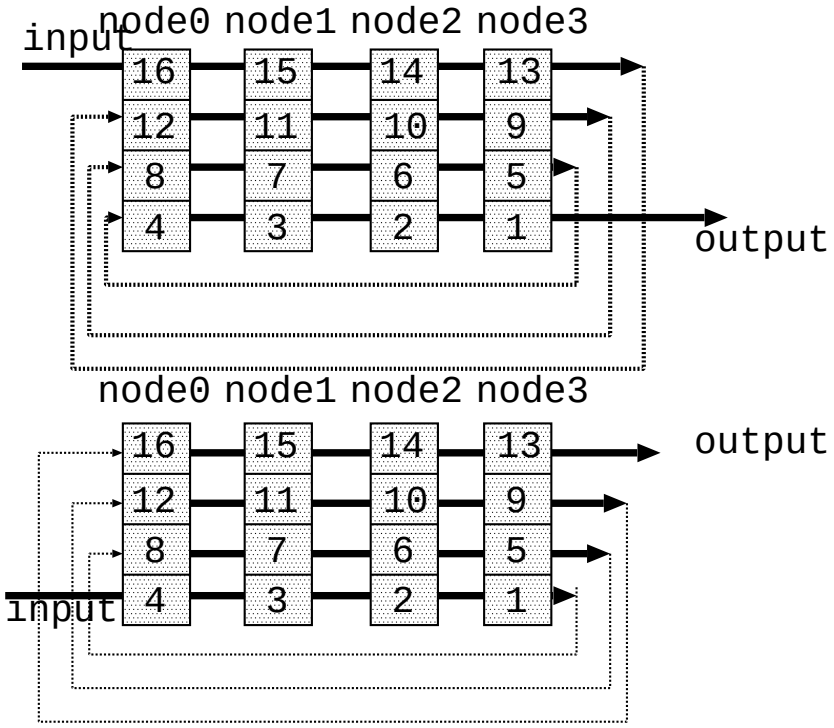


図12: 視線方向が負の場合のERT

ノード数が4，サイクル数が4の場合を例に挙げて説明する．計算すべきサブボリュームは視点に近いほうから1，2，…，16となっている(図12)．ネットワークは単方向でnode1 → node2 → node3 → node4 → node1 → …と環構造になっ

ている．従来の方法ではnode0の16から計算を始めて，node1の15，node2の14，node3の13，node1の12…と計算を進める．しかし，ERTを適用するためには視点に近いほうからの累積透明度，あるいは不透明度が必要である．しかしこの方法では計算順序が逆なので，ERTを適用できない．

そこで計算の順序をnode0の4，node1の3，node2の2，node3の1，node0の8…とし，node3での処理が終了する度に視点のほうからの累積透明度が分かるため，ERTが適用できる．

3.3 可視化方法

本節では時変ボリュームデータの可視化方法について述べる．まずボリュームメモリを3重化する場合の問題点を挙げ，次に3重化せずに可視化する方法について述べる．

3.3.1 3重化の欠点

可視化には2.2節で述べたような2種類の利用形態が考えられる．*ReVolver*/C40で採用したボリュームデータを3重化して主軸等間隔サンプリング方式を用いて可視化するという方式は，視点の位置によらず実時間可視化が可能である．そのため1つのボリュームデータを視点を変え，何度も繰り返し観察，解析を行うオフライン可視化に適した方式であるといえる．

しかし，シミュレーションの途中経過の観察などに利用されるオンライン可視化では，ボリュームデータが一定間隔Tで更新される．ここで一定間隔Tとはまさにシミュレーション側における，ボリュームデータの生成間隔であるといってよい．データ生成系の処理を中断させないためにはN^3バイトのボリュームデータ更新処理と更新されたボリュームデータに基づいた描画をT秒未満で実現させなければならず，レイテンシに対する要求が強い．そのため新しいボリュームデータは次のデータがくるまでに可視化できなくてはならず，間隔Tでデータ更新及び，可視化ができなければ可視化システムに致命的な問題を引きおこすことも十分考えられる．

ボリュームデータを3重化して可視化するという方法では，生成されたN^3程度のボリュームデータを3重化して全てのノードにもたせるために，多くの時間T_{update}がかかる．そのため，Tが比較的短いと3重化のための時間が大きなネックとなり得る．

TとT_{update}の時間はそれぞれボリュームデータのサイズの影響をうける．T

はさらにシミュレーションの複雑さにも影響をうける．T が比較的大きく，3 重化した後描画するのに十分な間隔であれば従来の方式で描画が可能であるが，T が小さく T_update の時間が大きく，制約を満たすことができない場合も考えられる．

次項ではボリュームデータを 3 重化することなくボリュームレンダリングを行う方式 [16] について説明する．

3.3.2　ボリュームデータの 3 重化を行わない方式

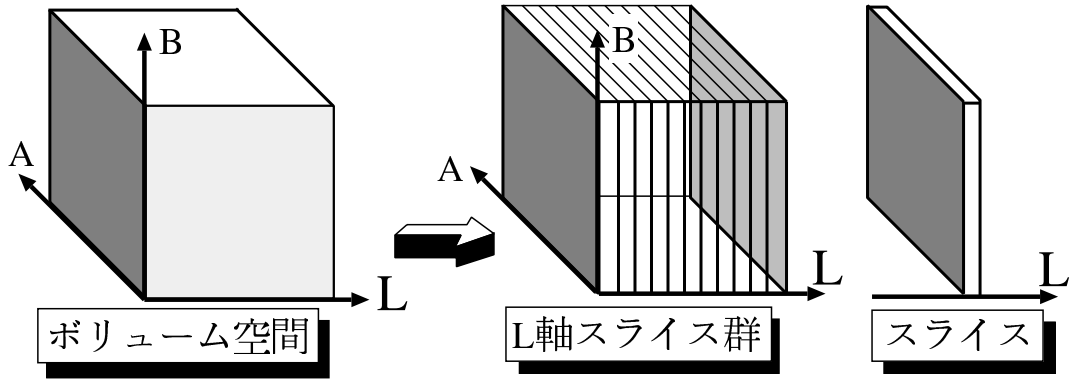


図 13: ボリューム空間と L 軸スライス群

本項ではボリュームデータを 3 重化せずに可視化する方式について述べる．まずは以降の説明で必要となる諸用語の定義を行う．

ボリューム空間を図 13 のように，L 軸の整数座標点 1，2,，…，n-1 に沿って，L 軸に垂直な n 個のスライスに分割し，それらをスライス群とする．

ここで，各スライスに対して，L 軸に対して垂直な面を「スライスの断面」,L 軸に並行な面を「スライスの側面」と呼ぶ．スライス群に対しては，両側にある L 軸に垂直な平面は「スライス群の底面」，あと 4 つの表面は「スライス群の側面」と呼ぶ．また，スクリーン 1 行分を「スキャンライン」と呼び，スキャンライン上の視線の集りを「視線群」と呼ぶ．

ReVolver/C40 のようにボリュームデータが 3 重化してあるシステムでは，視線はスライス群の底面からのみ入射することに帰着できた．そのため，各視線計算は全て先頭のプロセッサから始まっていた．

しかし 3 重化をしない場合はスライス群の側面に入射することも考えられる．その場合，視線計算は視線が入射したスライス (ノード) から始めればよいこと

になる．そのためレンダリングの最初に全てのノードは自分のノードに入射する視線情報を知る必要がある．

各ノードのピクセル値計算ユニット (PCU) は，送られてきた視線情報をもとに視線値計算を行い，自ノードが担当するスライスでの処理が終了すると，その計算結果を隣接ノード転送し，次の視線計算を行う．次に，特徴的な 2 つの視線入射パターンについて述べる．視線がスライス群の底面に垂直に入射した場合は，視線計算は 3 重化した場合と同様に，先頭ノードから逐次的に最後尾のノードへ伝播されながら行われ，最後尾のノードからピクセル値が出力される．また，視線がスライス群の側面に垂直に入射した場合は各視線は入射したノードで最後まで計算され，入射したノードから出力されることになる．この場合，ピクセル値は様々なノードから出力されるため，これらの値を集めるネットワークが別に必要になる．

同様の方式を採用している論文 [16] ではまず各視線がボリューム空間のどこに入射するかを計算する．そして入射点を含むスライスを担当するプロセッサに視線情報が送られる．その後各プロセッサが同期して視線計算を進めていく．これらの処理を 1 スキャンラインごとにパイプライン処理を行い，実時間可視化を行っている．

論文 [16] では視野角や，視点の位置を制限することで，全てのプロセッサが同期して並列処理を行い，ハードリアルタイム可視化を実現している．そのため，制限が守られないとシステム自体が動作しなくなる構成になっている．我々ははそのような制限を排除し，ソフトリアルタイムシステムを目指す．我々が述べるソフトリアルタイムシステムでは，視野角が大きくなった場合でも可視化処理のスピードが遅くなるだけで，可視化ができなくなることはない．

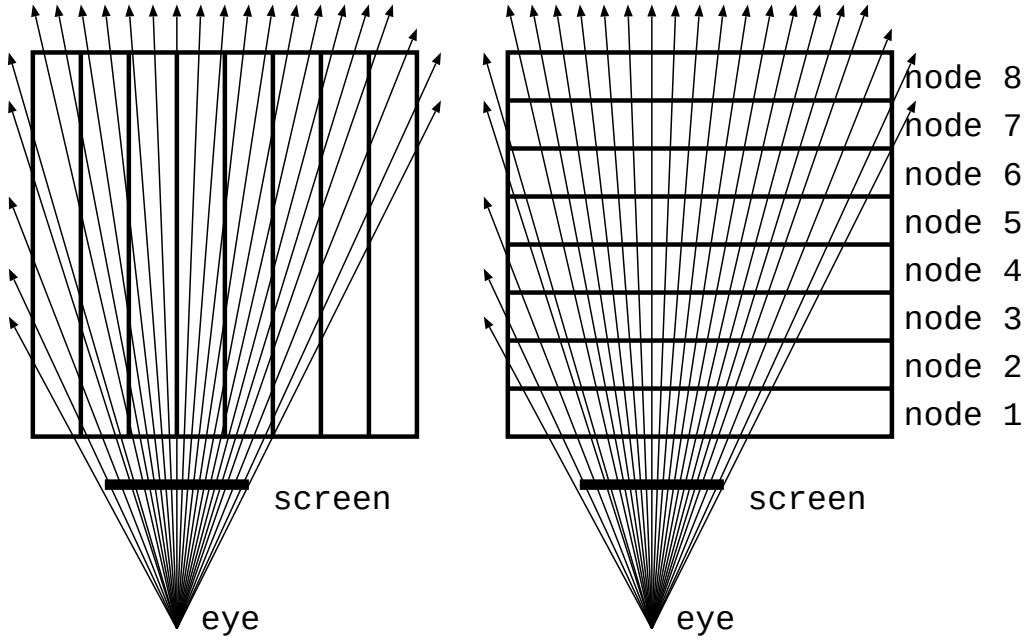
3.3.3 両方式の比較

本稿では *ReVolver*/C40 のようにボリュームデータの 3 重化を行う方式と論文 [16] のように行わない方式について比較する．両方式における特徴を以下に示す．

- 3 重化する場合

  まず，3 重化するためメモリ容量が 3 倍必要になる．またシミュレーションから得られるボリュームデータの更新処理に時間がかかる．一度データ更新が終ると，視点がどこにあっても実時間可視化が可能である．

- 3 重化しない場合



(a) スライスの側面側からの入射　　(b) スライスの底面側からの入射

or

3 重化した場合

図 14: 拡大した時の各ノードの視線密度

3 重化しないためデータ更新は速やかに行われる．しかし図 14 のように，ボリュームデータの特定の箇所の解像度を上げて，拡大して観察しようとした場合を考えると，3 重化していない場合は特定のノードに計算負荷が集中してしまう．図 14(a) を見れば，明らかに右端のノードに比べて中央のノードの負荷が高いことが分かる．特に並行投影した場合は透視投影の時のように視線が広がらないため，よりいっそう特定ノードに処理が集中する．

また，全てのノードからピクセル値が出力されるため，それらを集めて表示するためのネットワークも必要となる．

我々は大規模データの可視化を前提にしており，シミュレーションがそのような大規模な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考える．そのためオンライン可視化において，3 重化をしてもデータの更新時間がそれほど問題にならないと考える．

また，次章で述べるプリフェッチ機構においても 3 重化したほうがプリフェッチしやすく効率のよいプリフェッチが行え，画像の生成速度が速くなると考えられる．その結果，視点を何度も変えてボリュームデータを解析するオフライン可視化のような場合でも，1 つのデータから多くの画像を描画すれば，一枚あたりのレイテンシは相対的に小さくなると考える．

さらに開発にあたっても，各ノードで生成された出力を回収するネットワークを付加しなくてよいこともあり，次章以降はボリュームデータを 3 重化することを前提として議論する．

3.4 システム構成

本項ではこれまでの方針検討を踏まえた形で，提案するシステムの構成を説明する．提案する可視化システム [4] では，N × N × L のサブボリューム単位で並列化し，レイキャスティングアルゴリズムを用いて，1 ピクセル分のピクセル値計算をサブボリューム単位でパイプライン処理することで目標とする描画速度を得る．可視化システムの構成としては 128 ノード構成の PC クラスタに，ボリュームレンダリング向けアクセラレータ (VisA:**Vis**ualization **A**ccelerator)[8] を装備し，VisA 間を双方向高速リンクで接続する．計算結果は VisA 間リンクと同一規格のケーブルによりフレームバッファに送られ，ディスプレイへ表示される．生成された 2 次元画像に対して，さらに高度な後処理が必要な場合は，ホスト PC に送り処理を行う．

ReVolver/C40 では，視線生成，ピクセル値計算，シェーディングの 3 ステージでそれぞれ専用のハードウェアで構成したが，VisA ではピクセル値計算のみを重点的に専用ハードウェア化し，その他のステージで必要であった処理は各 PC の CPU や汎用グラフィックスカードを用いて実行する．以下，VisA の主要構成要素について簡単に説明を行う．

- PC クラスタ部128 ノードの PC クラスタ部は，視線生成処理を行うとともに，シミュレーションの母体となる. ここで生成されたボリュームデータは PCI バスを経由して，それぞれ各ノードに装備された VisA に送られる．

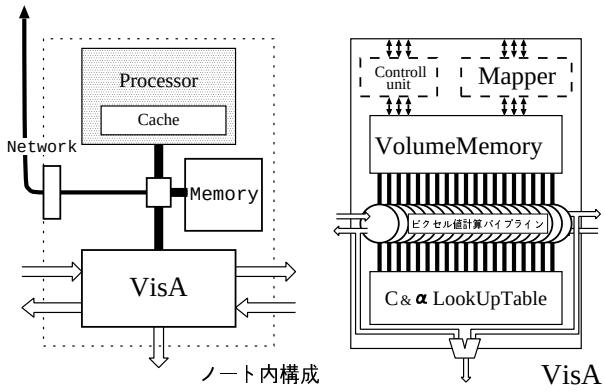


図15: ノードと VisA

- VisA 間リンク
  このリンクはレンダリングパイプラインにおいて計算途中の色情報，透明度情報，Z 値を送るためのリンクであり，フレームバッファの入力側スループットと同様 1008MB/s の転送速度が必要である．各ノードに隣接ノード間の双方向リンクと，フレームバッファへの出力ポートを設けることで，ReVolver/C40 では対応が困難であった種々のレンダリングアルゴリズムに対応する．具体的には DVI 用 LVDS インタフェースを用いてネットワークを構成する．
- ボリュームメモリ
  ボリュームデータを格納するための容量 2GB のメモリで $4000^2 \times 32$ のサブボリュームを 4 セットまで格納可能とする．ノード内のピクセル値計算ユニットに対して 4GB/s のバンド幅を確保するため，PC600 規格相当の Rambus chanel 4 チャネル (合計 4.8GB/s) で構成する．
- ピクセル値計算パイプライン
  32 個の PCU をパイプラインに接続して構成する．表示に必要な RGB は 8bit であるが，誤差伝播の影響を軽減するため 16bit 固定小数点として色と透明度の演算を行う．パイプライン周波数は画面サイズとフレームレートから 128MHz となる．この部分の詳細は後で述べる．

- Look Up Table (C& α LUT)
  RGB 各 8bit の色情報と，8bit の透過率を保持する 256 エントリのルックアップテーブルである．各ピクセル値計算ユニットが 1 ボクセルの演算を

行う度に 1 回参照されるため，スループット的にはボリュームメモリの 4 倍の性能が要求されるが，小容量のメモリであるため，マルチポート RAM を複数用いて実装可能と考えられる．

- 制御ユニットおよび Mapper
  制御ユニットは視線情報を始めとする描画に関する情報を CPU から受け取り，VisA 全体の制御を行う．Mapper は CPU 側からのボリュームデータを受け取り，ボリュームメモリに格納する．シミュレーション結果からのボリュームデータへのマッピング処理が定型的かつ簡易なものであれば，CPU 側でのマッピング処理を省略し，Mapper に直接マッピング処理を行わせることで高速化を図ることができる．この目的のために Mapper には FPGA 的な機能を持たせる．
- プリフェッチ機構
  ボリュームメモリはパイプライン構成している全てのピクセル値計算ユニットから同時にアクセスされるため，所望のパイプライン周波数を実現するために，LUT とボリュームメモリの間にプリフェッチバッファを装備し，ボリュームメモリへのアクセス遅延に伴うパイプラインストールを最小化する．

第4章　VisAの内部設計

VisAを用いた可視化システムの最終目標としては4000^3程度の大規模ボリュームデータを実時間可視化することである．VisAは専用ASICで作ることを前提としている．我々は現在そのプロトタイプとしてVisA Proの製作を行う．本章ではまずVisA Proの構成を述べる．その後，実装するプリフェッチ機構について説明し，実際に作成した回路についての説明を行う．最後に生成した回路の動作周波数について報告する．

4.1　VisA Proの構成

VisA ProはVisAの機能を$\frac{1}{2}$に縮小したもので，基本的な動作はVisAと同様である．内部では16段構成のPCUがパイプライン処理を行う．また高速なPCIバスを利用することで，シミュレーション系からVisAへのデータ転送を高速に行うことができ，データの更新を速やかに行うことができる．図16にVisA Proの構成を示す．VisA ProはFPGA部分としてVirtex-IIシルーズのXC2V6000

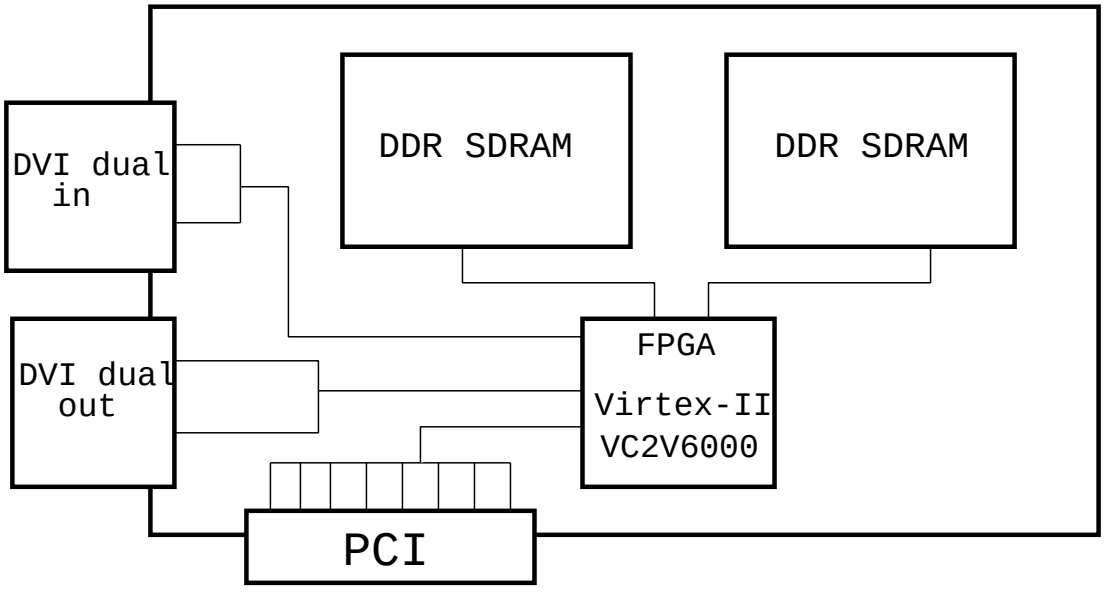


図16: VisA Proの構成

デバイスを採用することを考えている．また，Look up Table及び，プリフェッチバッファはFPGA上のオンチップメモリを使って実装する．

我々は現在VisAProの1/2程度のボードとして東京エレクトロンデバイス株式

会社 (TEL) の TD-BD-DDR266 という Viertex-II/DDR-SDRAM 評価ボードがあるので，このボードを使ってプリフェッチ機構と，PCU の設計を行った．この評価ボードは Virtex-II シリーズの XC2V1000 デバイスと DDR SDRAM266 が一基装備されている．XC2V1000 デバイスは実際に VisA Pro で使用する XC2V6000 の半分弱の性能である．そのため XC2V1000 デバイス上に PCU8 段構成の回路を実現できれば，機能を 2 倍に拡張して XC2V6000 上に実装できると考える．

4.2 ボリュームメモリアクセス制御機構

本節ではボリュームメモリへのアクセス機構について述べ，プリフェッチの必要性とプリフェッチ方針について述べる．

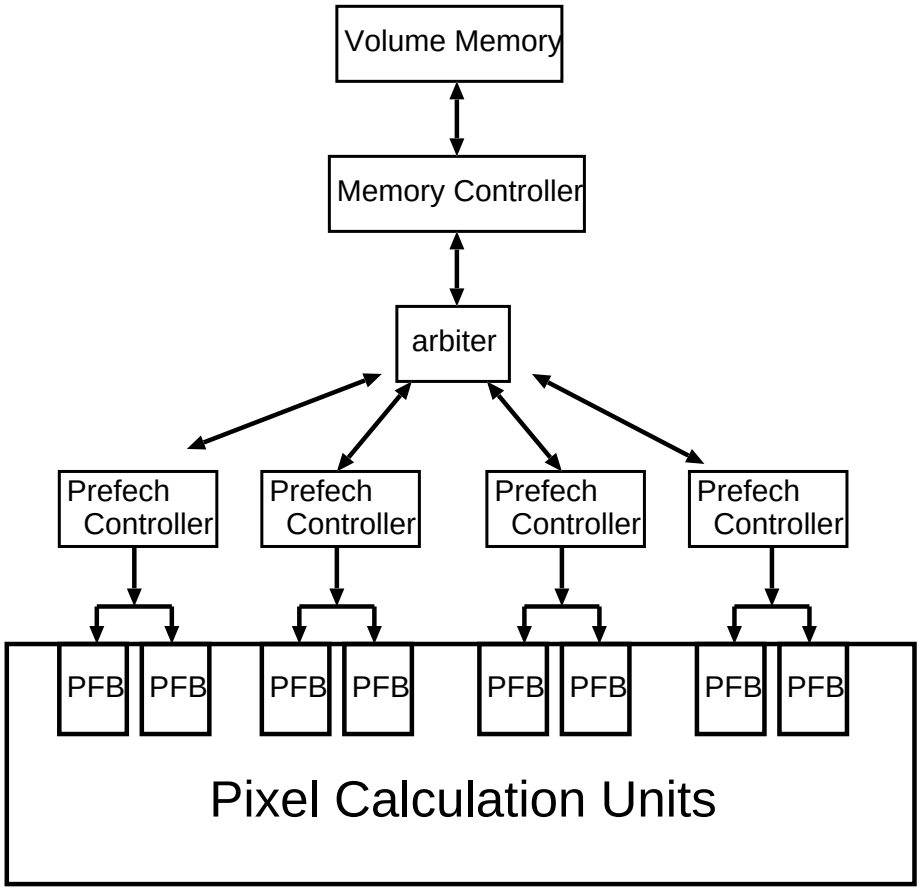


図17: ボリュームメモリアクセス機構

4.2.1 DDR SDRAM

搭載されているメモリはMicron社製のDDR SDRAMで4Mバンク ×8bit×4バンクという構成になっている．連続読み出しや書き込みを行う基準となるバースト長は2，4，8の3種類があり，1度のメモリアクセスでそれぞれ最大16Byte，32Byte，64Byte読み出し可能である．また，読み出し命令を発行してからデータがで始まるまでに要するCASレイテンシは，バースト長によらず，2または2.5サイクルである．またバーストリードの際にデータをすべて読み出すためにはバースト長/2サイクルかかる．

4.2.2 プリフェッチの目的・要求仕様

本システムでは，ボリュームデータは分割されサブボリュームとして各ノードに配置されていることを前提にしている．各ノードには複数のPCUが存在しており，各PCUを同時に動作させるためにはそのサブボリュームデータを各PCUが同時に読み出せるようにしなくてはならない．しかし，VisAの構成より一つのボリュームメモリに複数のPCUからの同時アクセスが必要となる．ボリュームメモリのどの番地をアクセスするかどうかは視線情報から決定される．そのためプリフェッチバッファをPCU分用意し，プリフェッチコントローラが視線情報を解析して，必要になるデータをあらかじボリュームメモリからプリフェッチバッファにフェッチしておく．そしてPCUからのデータのアクセスは全てプリフェッチバッファに対してされることとする．

4.2.3 ブロック単位のプリフェッチ

VisAProでは，DDR SDRAMのスループットを最大限発揮しないと，目標とする動作周波数(128MHz)を実現することができない．そのためDDR SDRAMのバースト転送を活用して読み出しを行う必要がある．そこでサブボリューム空間をブロック単位で構成し，プリフェッチする際にはバーストリードを利用してブロック単位でプリフェッチする．以下ではこのブロックのことを「単位ブロック」と呼ぶ．

プリフェッチ機構を設計するために決めるべきことは次の3つである．

- DDR SDRAMのバースト長
- 単位ブロックの形状
- プリフェッチコントローラの数

次項では上記の3つについて考察を行う．

4.2.4 考察

- DDR SDRAM のバースト長
  バーストリードするためにはバースト長/2 のサイクルで視線計算を行い，メモリに対してリード要求を出さなくてはならない．バースト長が 2 だと毎サイクル視線計算を完了させなくてはならない．逆にバースト長が長いと，プリフェッチできる時間間隔が広くなり，プリフェッチしたデータの利用率が下がってしまう．また，パイプライン止まった場合にできるだけ早く動作が再開できるようにするにはバースト長を短くしたほうがよい．
- プリフェッチコントローラの数
  プリフェッチバッファは PCU 毎に設ける必要がある．しかし 1 つのプリフェッチコントローラが複数のプリフェッチバッファを制御してもかまわない．この場合，ボリュームメモリから読み出す単位ブロックを複数の小ブロック (以後この小ブロックのことを「部分単位ブロック」と呼ぶ (図18 参照)) に分割して，各プリフェッチバッファには部分単位ブロックごとにデータを格納する．複数のプリフェッチコントローラはボリュームメモリにアクセスを行う際にお互いが競合するが，この競合はアービタで解消する．1

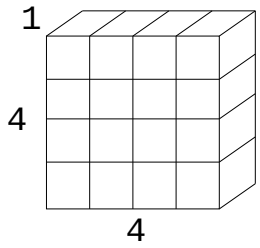


図 18: $4 \times 4 \times 1$ の部分半単位ブロック

つのプリフェッチコントローラが複数のプリフェッチバッファを担当することで，プリフェッチコントローラの数が減り，プリフェッチコントローラどうしの競合を緩和することができる．そのため，プリフェッチコントローラがボリュームメモリに要求を出した後，実際にボリュームメモリからデータが転送されるまでの時間を削減することができる．
- 単位ブロックの形状
  バースト長が 4(32 個分)) の場合では単位ブロックの形状 (直方体，又は立方体) としては (1×1×32),(1×2×16),(1×4×8), (2×2×8),(2×4×4) の組み合

わせが考えられる．

VisA では *ReVolver*/C40 と同様に 2.1.2 節で述べたようなボクセル並列処理でピクセル値計算を行う．ボクセル並列では各 PCU は自分が担当するスライスに対してにのみアクセスするため，部分単位ブロックの主軸方向の幅は必要ない．そのため部分単位ブロックの形状としてはは $X \times Y \times 1$ という形状が適当であると考えられる．

我々は，バースト長を短くし，さらに視線計算時間に余裕をもたせるためにバースト長は「4」とし，ボリュームメモリからは 1 回のアクセスで 2×4×4 の直方体形状の単位ブロックを読み出すことにする．さらにその単位ブロックを $4 \times 4 \times 1$ の 2 つの半単位ブロックとに分割し，それぞれ 2 つの隣接するプリフェッチバッファに格納する．

4.2.5 アービタ

プリフェッチバッファからボリュームメモリへのアクセスを制御するため，アービタを利用する．透視投影では，視線の広がりが大きいほどプリフェッチしたデータの利用効率が悪くなる．その結果プリフェッチコントローラは頻繁にデータを要求するようになる．そこで，各プリフェッチコントローラに優先度を設け，VisA 内のレンダリング処理で後方のプリフェッチバッファを制御するプリフェッチコントローラほど優先度を高くする．また，優先度が低いプリフェッチコントローラの最悪待ち時間を保証するために，スタベーションフリーである必要がある．

4.2.6 プリフェッチの動作

プリフェッチはプリフェッチコントローラが制御する．プリフェッチコントローラは，PCU で必要となるボクセル値のアドレスを，視線情報を使って先読みする．そして必要となるデータが既にプリフェッチバッファに存在しているかどうかを判定する．存在しなければ，アービタにボリュームメモリへのアクセス権を要求する．アクセス権が与えられたらボリュームメモリから所望のボクセルを含む単位ブロックを読みだし，それを 2 つの部分単位ブロックとして制御している 2 つのプリフェッチバッファに格納する．

4.2.7 視線の広がりにによる利用率の低下

PCU は毎サイクルデータを必要とする．PCU の数が P 台であれば，1 サイクルあたり合計で P バイトのデータを必要とする．そのため PCU が 8 個の場合は 1 サイクルあたり 8 バイトのデータが必要である．バーストリードをした

際の DDR SDRAM のバンド幅は，1 サイクルあたり 16 バイトである．そのため，効率よくプリフェッチしデータの利用率が高ければ，メモリバンド幅は足りている．

しかし，視投影において視線が広がると視線間隔が大きくなる．その結果プリフェッチ効率が悪くなるため，プリフェッチバッファに必要なデータがない確率が高くなる．その結果，プリフェッチ待ちになり，パイプラインが正常に動作しなくなるという問題点がある．

例えばプリフェッチ効率が 1/4 に落ちた場合，16 個のボクセルデータをプリフェッチしてきても 4 個しか参照されない．そのためプリフェッチバッファは 4 サイクルごとに次の単位ブロックを要求する必要がある．他のプリフェッチコントローラにおいても同様の減少が起こる．その結果，全てのプリフェッチコントローラがアービタに要求を出し続ける状態になる．このような状態では，一度ボリュームメモリからデータを読み出した後，次に読み出し可能となるのは 8 サイクル以上後になってしまう．この現象が続くと，より一層悪循環になってしまう．そこで本項では，視線間隔が広がるとそれにつれてボリューム空間の外に出て，ピクセル値計算をしなくてもよい視線が増えることを利用してこの問題を解決することを考える．

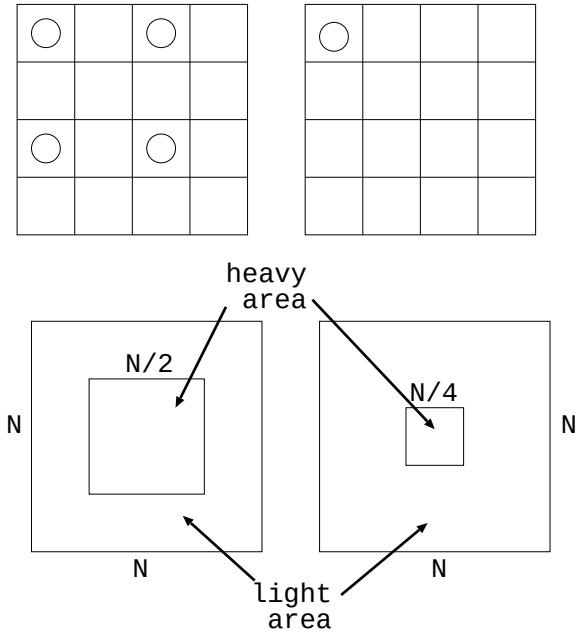


図 19: light 視線領域と heavy 視線領域

視線間隔が大きくなるにつれて，ボリューム空間の外側に出た視線は各PCUを通過するが，ボリュームデータ参照をする必要がないため，次々にパイプラインを通過していく．このような視線を，ボリュームメモリに対する負荷が軽いことから「light 視線」とよぶ．それに対してスクリーンの中央付近を通過する視線で，最後までボリューム空間内に存在しつづけ，しかもプリフェッチしたデータの利用率が 1/n になる視線が存在する．このような視線を，ボリュームメモリに対する負荷が大きいことから「$\frac{1}{n}$heavy 視線」と呼ぶ．$\frac{1}{n}$heavy 視線の数は n の値が大きくなるにつれて減っていき $\frac{1}{n}heavy$ 視線は全体の $\frac{1}{n}$ 程度になると考えられる．つまり残りの $(1 - \frac{1}{n})$ 本は light 視線である．

heavy 視線がデータ待ちでとまっている間に light 視線の情報が入ってきたら light 視線用の FIFO に入り，heavy 視線のせいでパイプラインがとまり，出力がでない場合は light 視線用の FIFO から出力をだし，視線計算がなかなか進まない heavy 視線を追いぬかしていく構成にする．もちろん heavy 視線の出力がでる場合はそちらを優先して出力させる．このようにすることで一番最初に heavy 視線エリアに入った時はしばらく出力がでてこない状態が続くが，それ以降は heavy 視線の遅れを light 視線が隠蔽する形で次々と処理が進むことになる．

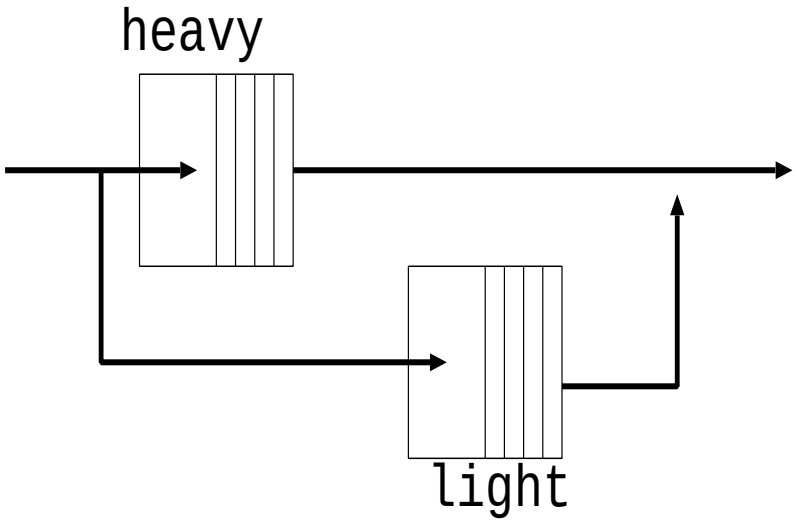


図 20: light 視線のバイパス

具体的な動作を図 21 を用いて説明する．図 21 では 1 から 10 までの 10 本の視線について考える．1 から 5 までの視線は heavy 視線で，6 から 10 までの視線は light 視線である．heavy 視線は入力された時点でプリフェッチバッファに

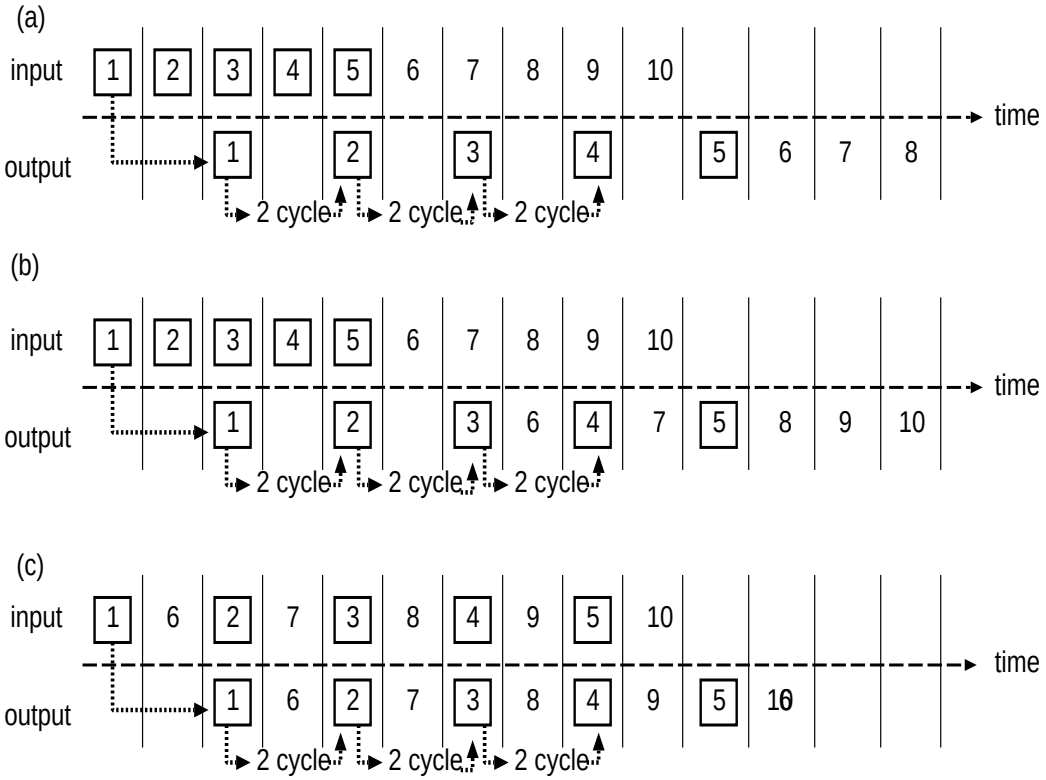


図21: 入出力のタイミング

データがないため，ボリュームメモリにアクセスしてデータをフェッチするのに2サイクルかかると仮定する．まず，図21(a)のようにlight視線のバイパスなしで，heavy視線が連続して入力されその後，light視線が連続して入力される場合を考える．

視線1の情報はプリフェッチバッファになく，入力されてから2サイクル後にデータが揃う．プリフェッチコントローラは視線1で必要なデータをフェッチし終わると次の視線2を見る．視線2もまたプリフェッチバッファにないのでまた2サイクルかけてデータをフェッチし，その後視線2のデータが揃う．このようにheavy視線が連続して到着すると，データのフェッチに時間がかかるため後続のlight視線もまたされることになる．

次に，視線の投入順序は(a)の場合と同様であるが，light視線のバイパスを用いた場合を図21(b)に示す．この場合，light視線は計算しなく，データをフェッチする必要がないため，heavy視線が出力されていない合間に出力してしまうことができる．さらに視線の投入方法を工夫し，図21(c)のようにlight視線とheavy視線が交互に到着するとさらにスムーズに視線を流すことができると考

えられる．

このようにして，light 視線のバイパス機構を用い，さらに視線の投入順序を工夫することで視線の広がりに対する問題点を解決できる．

4.3 ピクセル値計算部分の設計

本節ではピクセル値計算部分の設計について述べる．ピクセル値計算とは 2.3 節の式 (3)，(4) で示された計算のことを指しており，この計算を行う回路を実装する．本節ではまず，計算に利用する Virtex-II の高速乗算器について述べ，その後 VisA 内部の主要なモジュールについて説明する．

4.3.1 BRAM と高速乗算器

ブロック RAM(BRAM) とは 18Kb のブロック単位で VirtexII デバイスに多数搭載されているデュアルポート RAM で，高速メモリとして使用することができる．このメモリは 18Kb の記憶領域と，完全に独立した 2 つのアクセスポート (A および B) から構成されている．

データの書き込みはいずれか 1 つのポート，データの読み出しは同じポートまたは 1 つのポートから行うことができる．各ポートは，それぞれのクロック，クロックイネーブル，ライトイネーブルに同期する．また読み込みも同期を取って行われるので，クロックエッジが必要である．

次に，高速乗算器とは VirtexII 多数装備されている 18bit×18bit の 2 の補数エンベデッド整数乗算器のことを指す．各乗算器は BRAM と配線リソースを共有しているため，効率のよい計算が可能である．しかも，同じ配線を使用しているのにもかかわらず，BRAM のデータ幅が 18bit 以下であれば，BRAM とは完全に独立に動作し，BRAM に干渉することがない．言い換えると，BRAM メモリのデータ幅が 18bit を超える場合は，その BRAM と配線リソースを共有している高速乗算器は使用することができないことになる．

また，この乗算器は高速性を重視して最適化されており最大 210MHz での動作が可能である．実装においては，この乗算器をできる限り活用し高速性を高めていくこととする．

しかし高速乗算器は数に制限があり，ピクセル値計算部分の回路の設計にあたっては，できるだけ乗算器の数を減少させるように工夫しなくてはならない．次項でその考察を行う．

4.3.2 計算式についての考察

式(3)では，加算1回，減算1回，乗算2回，式(4)では乗算1回の計算が必要となる．C_k と T_k 一段分の計算をパイプライン方式で赤,緑,青(以下，それぞれR,G,Bと呼ぶ)3色分の計算を同時に行うためには乗算器は計7個必要となる．また各演算を1サイクルで実行すると4サイクルかかる．

式(3)は透明度tと累積透明度Tを用いた式であった．透明度tと不透明度αは$\alpha = 1 - t$という関係がある．同様に累積透明度Tと累積不透明度Aは A = 1 - T という関係がある．そのため，式(3)の変形として次の3つの場合が考えられる．

- 不透明度αと累積不透明度Aを使用
- 不透明度αと累積透明度Tを使用
- 透明度tと累積不透明度Aを使用

以下では式(3)を除く3つの場合について必要な乗算器の数，及び1ボクセルの計算にかかるサイクル数求める．

- 不透明度αと累積不透明度Aを使用
  不透明度α，及び累積不透明度Aを用いると式(3)，(4)は式(8)，(9)のように変形される．

$$C_k = C_{k-1} + \alpha(v_i) \cdot c(v_k) \cdot (1 - A_{k-1}) \tag{8}$$

$$A_k = A_{k-1} + \alpha(v_k)(1 - A_{k-1}) \tag{9}$$

  この場合，$\alpha(v_k)(1 - A_{k-1})$が各式で共通であることを利用すると，R,G,B,不透明度の計算を行うために，乗算器は4つでよいことになる．また，サイクル数は4サイクルと変化がない．
- 透明度tと累積不透明度Aを使用
  透明度tと累積不透明度Aを用いると式(3)，(4)は式(10)，(11)のようになる．

$$C_k = C_{k-1} + (1 - t) \cdot c(v_k) \cdot (1 - A_{k-1}) \tag{10}$$

$$A_k = 1 - t(v_k) \cdot (1 - A_{k-1}) \tag{11}$$

  この場合は，$(1 - A_{k-1})$が各式で共通であることを利用しても，R,G,B,不透明度の計算を行うために，乗算器は7つ必要になる．また，サイクル数も4サイクルと変化がない．

- 不透明度 α と累積透明度 T を使用
  不透明度 α 及び，累積透明度 T を用いと，式 (3)，(4) は式 (12)，(13) のようになる．

$$C_k = C_{k-1} + \alpha(v_i) \cdot c(v_k) \cdot T_{k-1} \tag{12}$$

$$T_k = T_{k-1} - \alpha(v_k) \cdot T_{k-1} \tag{13}$$

  式 (12)，式 (13) では，$\alpha(v_k) \cdot T_{k-1}$ の計算が共通していることを利用すると，R,G,B, 必要な演算数は，乗算 4 回となった．また，1 段の計算も 3 サイクルで行うことが可能である．

以上の考察より我々は乗算器の数，及び計算に要するサイクル数が最も少ない，不透明度 α 及び，累積透明度 T_k を用いた計算方式を採用する．

4.4 実装

4.4.1 PCU

前項の考察より，式 (12) 及び，(13) の計算を行う回路を作成する．以後この一段分の計算を行うモジュールを PCU と呼ぶ．

PCU 内部はパイプライン方式となっているが，式 (12) の 1 項目と 2 項目の加算は次の PCU 内で計算させることで，PCU 内のパイプライン段数を 2 にすることができ可視化処理のレイテンシを少なくした．一本の視線計算の様子を図 22 に示す．

次に PCU の入力ポート，及び出力ポートについて説明する．

- 入力
  - alpha
    計算対象となるボクセルの不透明度 $\alpha(v_i)$.
  - Tk_1
    計算前までの累積透明度 T_{k-1}.
  - cr,cg,cb
    計算対象となるボクセルのカラー値. それぞれ R,G,B に対応している．
  - sr_pre, sg_pre, sb_pre
    前段でのカラー値の計算結果．それぞれR,G,B に対応している．式 (12) の $\alpha(v_i) \cdot c(v_k) \cdot T_{k-1}$ の計算結果に相当する．
  - Ckr_1, Ckg_1, Ckb_1

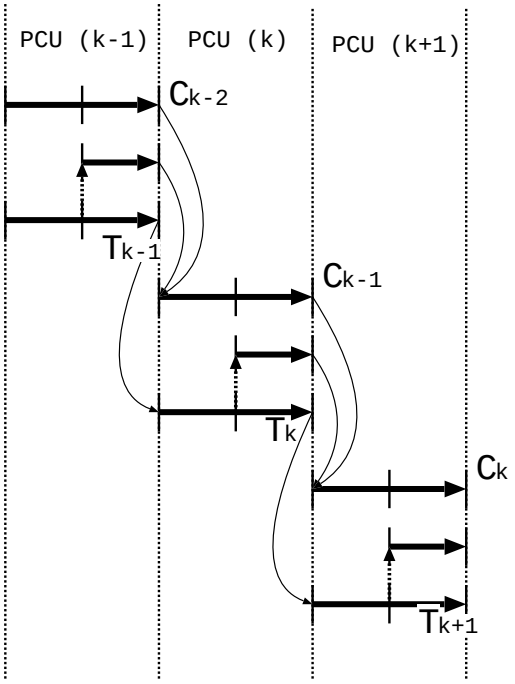


図 22: PCU 内の視線計算の様子

前々段までの累積カラー値．それぞれ R,G,B に対応している．この値に前段での計算結果である，sr_pre, sg_pre, sb_pre を加算する．

- clk

  システムクロック．

- 出力
  - Tk

    この段までの累積透明度 T_k の値．
  - sr, sg, sb

    この段でのカラー値の計算結果．それぞれ R,G,B に対応している．式 (12) の $\alpha(v_i) \cdot c(v_k) \cdot T_{k-1}$ の計算結果に相当する．この値と累積カラー値との加算は次の段でなされる．
  - Ckr, Ckg, Ckb

    前段までの累積カラー値．それぞれ R,G,B に対応している．C_{k-1} に相当する．

PCS の構成を図 23 に示す．カラー計算は 3 色とも同様であるため，図には 2 色分は省略した．$C_{k-1} + S_{pre}$ の計算の後，$carry$ がもし 1 であれば，カラー値はオーバーフローしているため，C_k <= "1111111111111111" とする回路も付随

している.

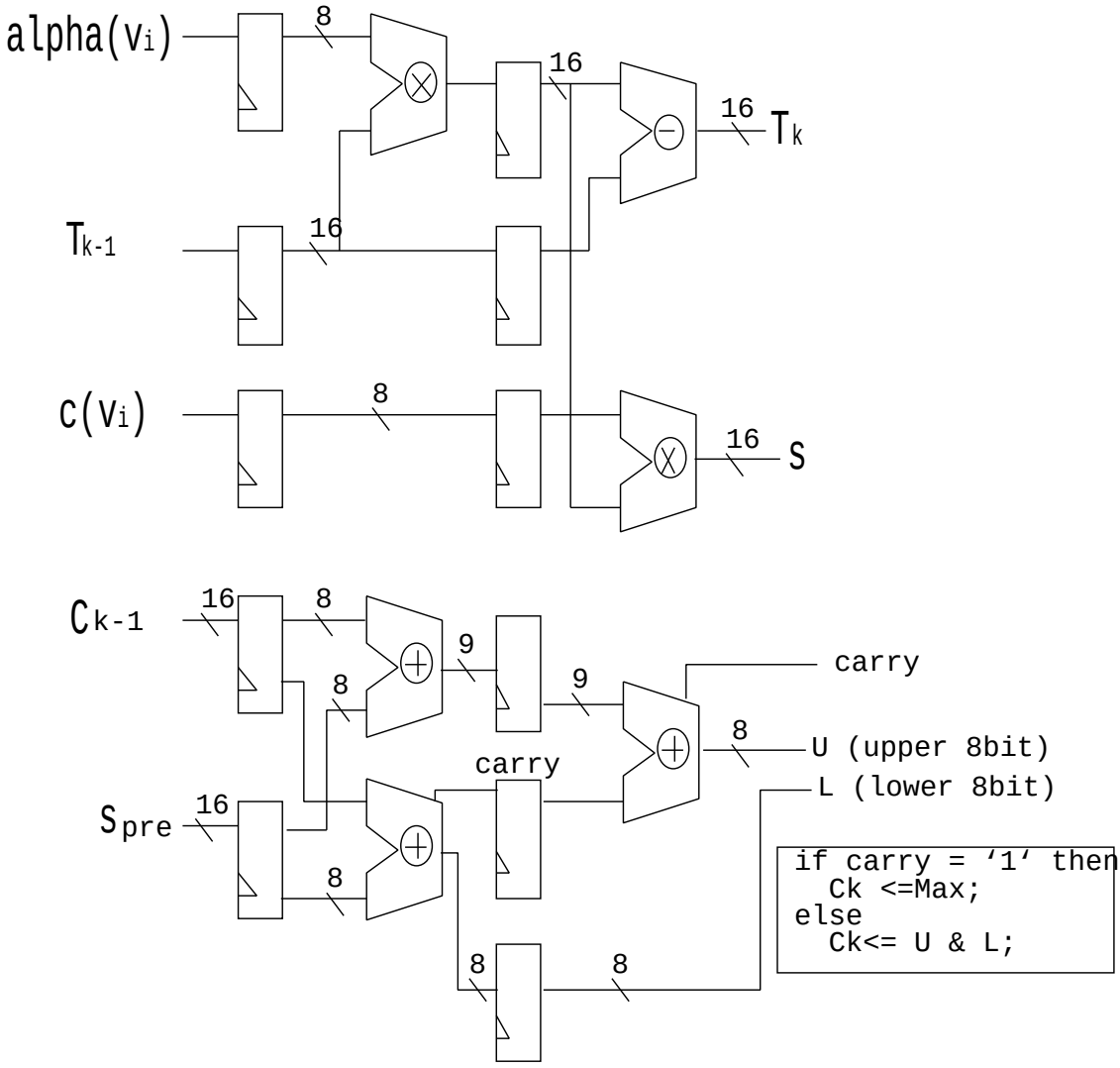


図 23: PCS 構成

4.4.2 Look up Table

Look up Table はボリュームデータを R,G,B,α の値に変換するためのテーブルで，PCU から毎サイクルアクセスがある．そのため高速オンチップのメモリ (BRAM) を使って構成する.

Look up Table は 8bit(256 エントリ) のボクセル値を R,G,B,α 各 8bit の 32bit データに変換するので，その容量は $256 \times 32 = 8Kb$ になる．これは SelectRAM

ブロックを1ブロックを用いてBRAMで実現することができる．データ幅が32bitであるので，4.3.1項で述べた性質より，Look up Tableに使用したBRAMに配線を共有している高速乗算器は使用することができない．また，Look up TableへのwriteWrite要求は，可視化パラメータ設定時のみなので，以後はread要求のみとなるためシングルポートメモリとした．

4.4.3 プリフェッチバッファ

- 要求仕様

  プリフェッチバッファもLook up Tableと同様に毎サイクルread要求があるため高速なメモリで構成する必要がある．

  1枚の画像生成において，一度参照されなくなった半単位ブロックは数スキャンライン分のデータが格納できる程度大きなプリフェチバッファを用意しない限り，次の参照の前にプリフェッチバッファから追い出されていると考えられる．我々は400^3程度のボリュームデータの可視化を考えており，そのような大規模なプリフェッチバッファを用意しておくことは現実的でない．そのためプリフェッチバッファの容量は特別大きくする必要はない．またプリフェッチバッファに所望のデータが含まれている半単位ブロックが存在するかどうかの判定を行う時は，容量が小さいほど高速アクセスが可能となる．これらの理由よりプリフェッチバッファの容量は半単位ブロックを4つ格納できる程度とした．

  次に，プリフェチバッファを構成するBRAMのデータ幅について考える．プリフェッチバッファはボリュームメモリ側からのデータの書き込みとPCU側からのデータの読み込みがしばしば行われるため，両アクセスを同時に行うことができるように，BRAMのデュアルポート機能を利用して実現した．2つのポートはそれぞれ読み込み専用，及び書き込み専用ポートとした．プリフェッチバッファからのデータの読み出しは1バイトずつ行われるので，読み込み側のデータ幅は8bitとした．以下では書き込みポートのデータ幅について考える．

  ボリュームメモリはダブルデータレートであるため，1サイクルあたり16バイトのデータを読み出しを行う．ボリュームメモリから取り出したデータは2つのプリフェッチバッファに分配するので，各プリフェッチバッファには1サイクルあたり8バイトのデータが届くことになる．

- 問題点

届いた64bitのデータをそのままプリフェッチバッファに格納するためにはプリフェッチ書き込み側のデータを64bit幅にしなくてはならない．しかし，Virtex-IIデバイスのBRAMは4.3.1項で述べたように18bitを超えるデータ幅ではそのBRAMと配線を共有しているを高速乗算器を使用することができない．また，BRAMはデータ幅が36bit以下まではSelectRAMブロック1つで構成することが可能だが，それ以上ではデータ幅が36bit増加するごとに使用するブロック数が1つ増加する．つまり64bitのデータ幅のBRAMを構成するためにはSelectRAMブロックが2ブロック必要であることが分かる．

結局，PCUを8段連結構成にする回路において，プリフェッチバッファのデータ幅を64bitにするために必要なSelectRAMブロックの総数は，Look up Table用に8個，プリフェッチバッファ用に16個で24個となる．XC2Vデバイスには高速乗算器が40個装備されているが，上記の議論よりBRAMと配線を共有している乗算器は全て使うことができない．そのため使用できる高速乗算器は16個となり，これは本来必要な32個に到底及ばない個数である．

- 解決策

プリフェッチからの読み出しは先ほども述べたように毎サイクル必ず行われる．しかし，書き込みはボリュームメモリからデータを取ってくる際に起こるが，これはプリフェッチバッファ(正確にはプリフェッチコントローラ)が複数あるため，毎サイクル行われることはない．一度書き込みがおこると8バイトのデータが2サイクルにわたって連続して送られてくるが，その後はボリュームメモリへのアクセス待ち状態になっていると考えられる．4つのプリフェッチコントローラがそれぞれ2サイクルかけてデータを読み込むため，このアクセス待ち時間は各プリフェッチコントローラが順番にボリュームメモリにアクセスすると考えると平均して8サイクル分程度あると考えられる．そこで，読み出したデータはその間隔内でプリフェッチバッファに格納すればよいことになり，プリフェッチバッファのデータ幅を64bit以下にしても大きな支障はないと思われる．

まず，データ幅を32bitにすることを考える．32bitにするとボリュームメモリから得られた16バイトのデータをプリフェッチバッファに格納するのに4サイクルかかるため，上で述べた8サイクル内に収まっている．また

64bit から 32bit にすることで，データ幅が 36bit 以下になり，各プリフェッチバッファとして用いる BRAM は SelectRAM ブロック 1 個で構成することができる．しかし依然として 18bit よりは大きいため，BRAM と配線共有のを高速乗算器使用することができない．その結果 BRAM の個数は 16 個と削減できたが，使用可能な高速乗算器は 24 個であり，まだ不足している．次にデータ幅を 16bit にすることを考える．16bit にすると，データを格納するのに 32bit の時に要した 4 サイクルの 2 倍の 8 サイクルかかる．この値もぎりぎり許容範囲と言ってよい．さらに，データ幅が 16bit なので，プリフェッチバッファは SelectRAM ブロック 1 個で構成することができる．またデータ幅が 18bit 以下であるので，この BRAM と配線共有している高速乗算器も使用可能である．そのため使用可能な高速乗算器は，Look up Table に使用した BRAM と配線共有しているものを除く 32 個となり，ピクセル値計算処理で必要な全ての乗算器は高速乗算器を用いて実装することができた．

4.4.4 ノード

ノードとは主として PCU，Look up Table 及び，プリフェッチバッファで構成されるモジュールのことを指す．内部の構成は図 24 のようになる．

データが既にプリフェッチバッファにプリフェッチされていると，まず *pfb_read_addr* に読み出したいボクセルのアドレスが送られてくる．そのアドレスから 8bit のボクセル値 (*vox*) が読み出され，Look up Table に送られる．Look up Table ではそのボクセル値が 32bit の RGBα 情報に変換され出力される．得られた 32bit 情報は R,G,B,α それぞれ 8bit づつ分割され，PCU に渡される．PCU ではこの値を使ってピクセル値計算を行う．

we, 及び *we2* 信号はそれぞれ Look up Table, 及びプリフェッチバッファのライトイネーブル信号であり．*init* 信号は Look up Table を初期化する際のアドレス入力の切り換えに使用する．*pfb_in* と *pfb_write_addr* はプリフェッチバッファにデータを書き込む際のデータ信号とアドレス信号である．

4.4.5 8 段パイプライン構成

VisA Pro の内部はは前項で述べたノードをノード 1 からノード 8 まで並べた構成になっている．

ところで，式 (12) の第一項と第二項の加算は次のノードで計算するのだが，ノード 8 はもう次のノードが存在しないため別に計算しなくてはいけない．そ

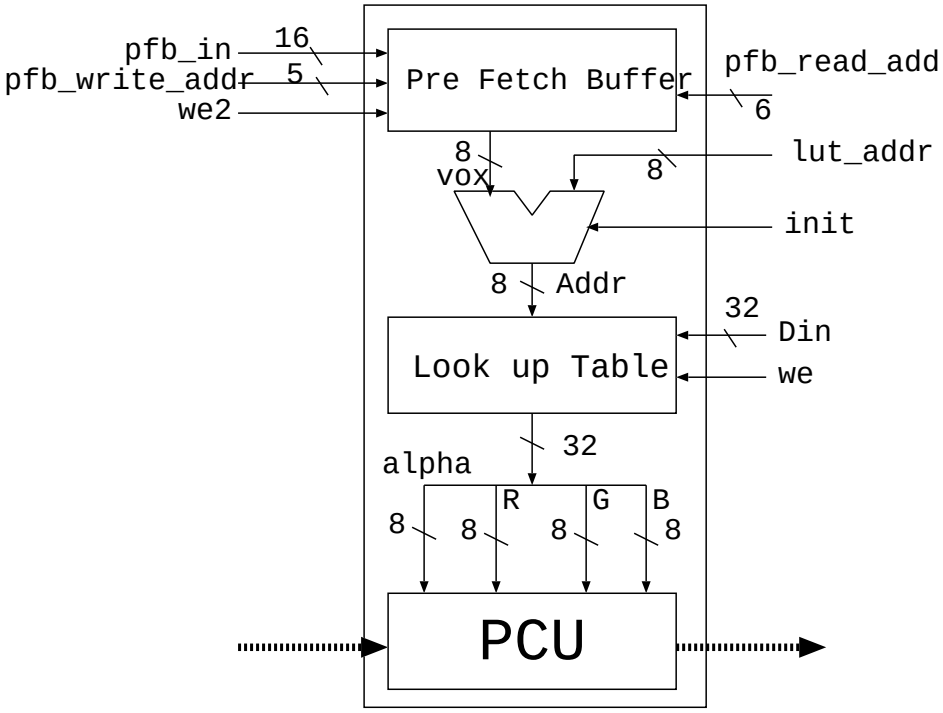
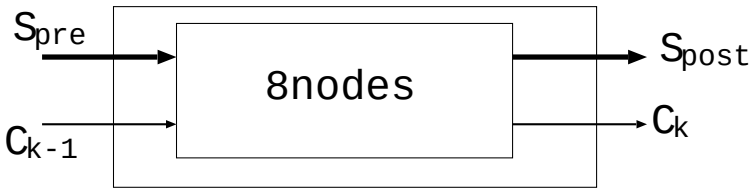
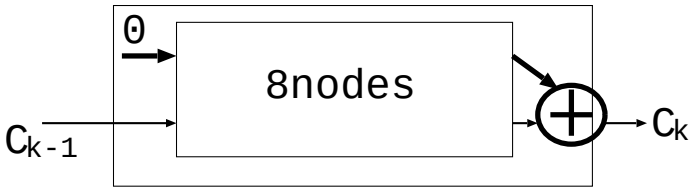


図 24: ノード構成

こで，図 25 に示す 2 つの方法について考える．



(a)



(b)

図 25: 8 段構成

まず図 25(a) の方式はノード 8 のあとはもうノードが存在しないため，ノード 8 での計算結果は次の VisA のノード 1 に渡される．そしてノード 1 では隣接

ノードから送られてきたデータを処理する．(a) の方式では余分に回路を付加する必要がないが，隣節ノードへのデータ転送量が増加してしまう．増加量は R,G,B それぞれ 2 バイトづつで合計 6 バイトになる．

次に (b) の方式では，ノード 8 の後に加算回路を付加し，式 (12) の第一項と第二項の加算をさせてから隣のノードに送る．その結果ノード 1 に入力される C_{k-1} はそのままノード 2 に渡せばよいためノード 0 の S_{pre} は 0 とした．この加算回路は規模も小さく容易に搭載できるため，我々は (b) の方式を採用し，ネットワークの負荷を軽減した．

作成した回路を，Viertex-II シリーズの XC2V1000 デバイス上に実装するとゲート数は 125 万程度となった．1 ノードで 15 万 5 千ゲート程度であり，そのうち PCU 部分は 2 万 4 千ゲートであった．

4.4.6 動作速度の高速化

我々は最終的には SHD 規格である 2048^2 のスク リーンに毎秒 30 フレーム出力することを目標としている．そのために必要な動作周波数は 128MHz である．さらに，PCI バスの動作周波数 33MHz であり，この周波数の整数倍のクロックを使うことが望ましい．また DDR SDRAM266 の性能をフルに発揮するためには 133MHz の動作周波数が必要である．以上の理由より実装において動作周波数の目標値を 133MH とした．

評価環境は以下の通りである．

表 3: 評価環境

CPU	Intel Pentium4
OS	Windows2000
メモリ容量	512MB
CAD	Xilinx FondationF4.1i

自動配置配線だけを行った結果 89MHz で動作可能となった．133MHz ほどの動作速度を実現するためにはゲート遅延だけでなく，配線遅延も十分考慮する必要がある．

Virtex-II シリーズ XC2V1000 デバイスでは selectRAM ブロック，及び高速乗算器は図 26 の黒い部分に示した位置に，10 個ずつ計 40 個配置されている．1

ノードあたり使用するSelectRAMブロックはLook up Tableとプリフェッチバッファで合計2つ．高速乗算器は4つで，4つのうち1つだけはプリフェッチバッファで使用するBRAMと配線共有している乗算器を用いる．結局1ノードあたり5つのSelectlRAMブロック領域を使用する．そこで図27(a)のように，使用するSelectRAMブロックがノードごとにかたまるように配置指定し，自動配置配線を行った．その結果図27(b)に示すように，ブロックSelectRAMのみならず，ノード内の他の回路もノードごとにかたまって配置され，116MHzで動作可能となった．

その後さらにクリティカルパス上の部品の配置をFloorPlannerというツールを用いて少しづつ手動で変更した．FloorPlannerとはマウスでドラッグ&ドロップすることでターゲットとするデバイスへのデザインの配置を制御することができるグラフィカル配置ツールである．FloorPlannerを用いることで最終的に133MHzで動作可能となった．

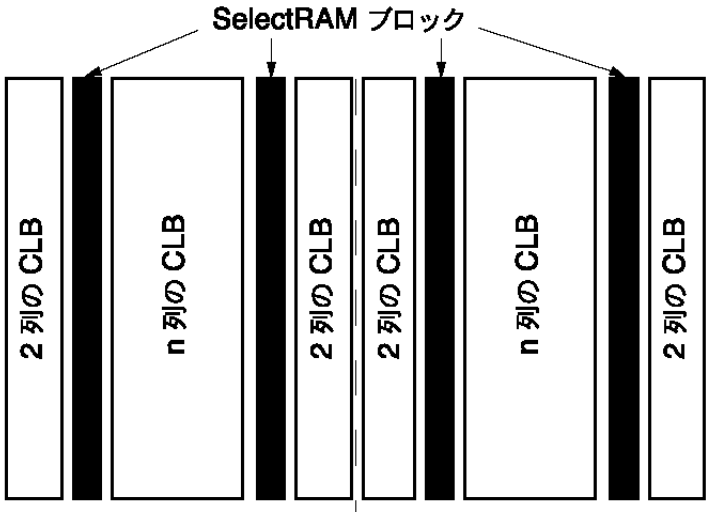
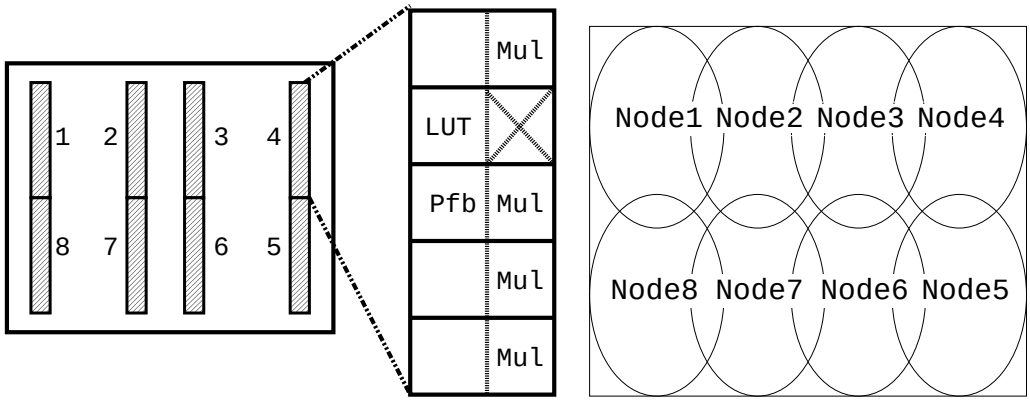


図26: seletRAMブロックの配置

4.4.7 VisA Proへの拡張

4.4.3節で述べたように，プリフェッチバッファのデータ幅を64bitにした場合，1ノードに要するブロックSelectRAMは合計7個であるため，16段構成にするために必要なブロックSelectRAMの個数は112個である．XC2V6000にはブロックSelectRAMが144個装備されているため，プリフェッチバッファのデータ幅を64bitにしても実装可能である．



(a) 乗算器及び，BRAM の配置指定　　(b) 各 PCU ごとの配置状況

図 27: FPGA 上への実装

また，今回の評価ボードへの実装では，高速乗算器の個数不足より 3.1 節で述べた，視線方向が負の場合の補正回路は含めていない．補正回路は 3.1 節の式 (5) の計算 1 段分に相当する．この回路には乗算器が 4 つ必要であると考えられるが，XC2V6000 に装備されている乗算器数より考えて，十分実装可能であることが言える．

第5章　考察

本章では以下の2点について考察する．まずは今回の我々の実装には採用しなかった，全ボリュームデータの3重化をしない方式(3.3.2節)の実装方法について考察する．次に，効率のよいプリフェッチを実現する擬似透視投影法について考察する．

5.1　全ボリュームデータの3重化をしない場合についての考察

本節では3.3.2節で述べた，全ボリュームデータの3重化をしない場合の処理について，VisA内での動作について考察を行う．以下「全ボリュームデータ」は可視化対象となる全ボクセルデータを指すとし,「サブボリュームデータ」は一つのノードか持っているボリュームデータのことを指す.

このVisA内でのサブボリュームのレンダリング処理(以下「サブボリュームレンダリング」と呼ぶ)では，視線がそのノードがもつサブボリュームを通る間のレンダリング処理であるので，当然ながら必要なデータは必ずそのノードに存在することになる．そのため，VisA内のP台のPCUでの並列処理の方法として，ボクセル並列処理に加えて，ピクセル並列処理を行うことも可能である．以下では各並列処理について述べる．

- ボクセル並列処理
  ボクセル並列処理は全ボリュームデータを3重化した場合と同様の並列処理方法で，P台のPCUを$PCU_1, PCU_2, \cdots, PCU_P$の一次元配列構成として考え，視線情報を$PCU_1$に次々に与え，パイプライン方式で計算を進めていく方式である．
- ピクセル並列処理
  ピクセル並列処理はでは，各PCUが視線計算を並列に行う方式である．各PCUは視線がサブボリューム内にある間計算し，該当ノードのサブボリューム内の処理が終ったら新しい視線の計算を始めるという方式である．

ボクセル並列処理を実現する方法として，サブボリュームデータの3重化をすることが考えられる．この方式は全ボリュームデータを3重化する場合と比べて，メモリ消費量は同じであるが，自分のノード内だけでデータの3重化を行うので，データの更新時間がボリュームデータの3重化をしない場合と同様に高速にできる．

全ボリュームデータを3重化する場合は，各PCUの担当は1スライス分であったが，サブボリュームデータの3重化では，各PCUは複数スライスを担当することにもなる．その複数のスライスを「スライス群」と呼ぶ．全ボリュームデータの3重化において，各PCUは担当するスライスが固定されていたのと同様に，サブボリュームの3重化では各PCUが担当するスライス群が固定されるため効率のよいプリフェッチを行うことができると考えられる．

回路の構造としては全ボリューム空間の3重化の時のようにPCU_iでの計算結果は必ずPCU_{i+1}に渡されるのではなく，各PCUはスライス群のスライスの数に応じて自分のなかで繰り返し計算を行う必要があり，そのための判定処理とループバック構造が必要となる．

ところで，各PCUへのデータ入力は前段のPCUからのデータと自分の計算結果からのループバックの2通りになる．ボリュームデータのサイズがN^3であれば，PCU内でのループバック回数は最大N回になる．仮に先行する視線がこのようなN回ループの視線であり，後続の視線は0回ループの視線であった場合，パイプラインはまったく動作しなくなってしまう．しかし，ループ回数の視線から入射するなど，視線の入射を工夫することでそのように事態は回避できるであろうと思われる．

次にピクセル並列について考える．ピクセル並列処理の利点としてはデータがプリフェッチバッファにあれば，ボクセル並列処理のように，PCU間で先行視線のせいで視線が待ち状態になるということはおきないことがあげられる．また，空いているPCUに処理を割り振るというアルゴリズムを使っていれば，視線ごとにPCUでの処理時間がばらついていても他のPCUの処理に影響を与えることがない．

しかし，ピクセル並列処理では一つのPCUが扱うデータがボクセル並列に比べて不規則なので，プリフェッチの効率が悪くなってしまうという欠点がある．

視線の入れかたを考慮しパイプラインをなるべくとめないようにすれば，プリフェッチの効率のよいボクセル並列処理のほうが有利であると考えられる．

5.2 擬似透視投影

VisAでは並行投影，及び遠近法を利用した透視投影の両方の投影法をサポートしている．透視投影で描画したほうが人間が見るためには自然な画像になる．しかし，VisAでは並行投影であればは視線ベクトルが全て同じ方向であるため，

透視投影に比べてプリフェッチがしやすく高速動作が期待できる利点がある．

そこで，VisA では透視投影に近い画像をできるだけ高速に描画するために，並行投影と透視投影を組み合わせた方式 (以後,「擬似透視投影」と呼ぶ) で描画する方式を採用することを考えている．以下擬似透視投影について述べ，VisA での適用効果について考察する．

擬似透視投影では視線は透視投影のように，視点からスクリーンに向けて放射線状に出す．スクリーンはピクセルの集合であるが，あるピクセルとその近

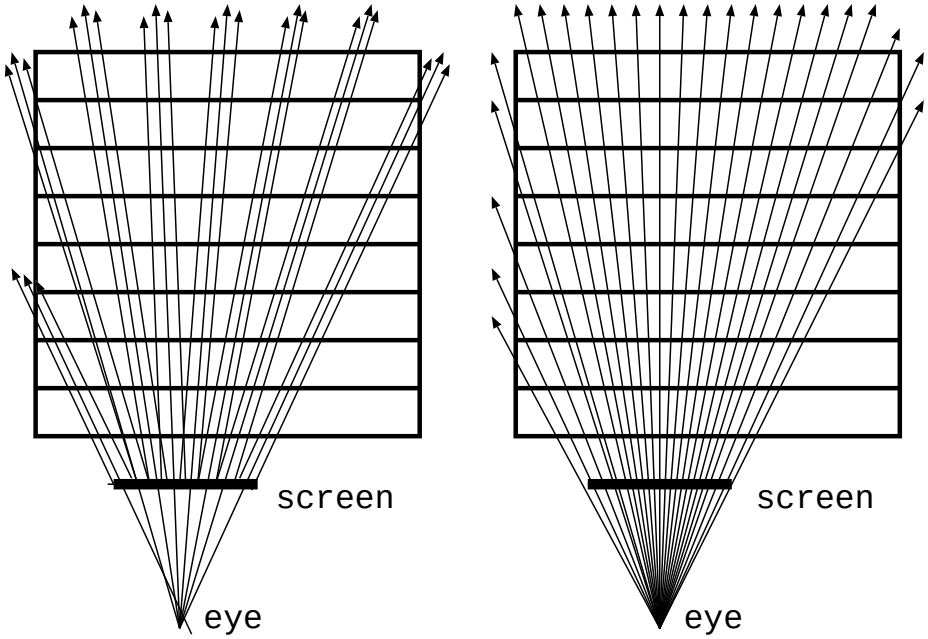


図 28: 擬似透視投影

傍にある複数のピクセルを一つのグループ (以後「サブスクリーン」と呼ぶ) とし，視線ベクトルはこのそれぞれのサブスクリーンに対して一本生成する．そしてサブスクリーン内の各ピクセルの視線ベクトルはその視線に並行であると考えるのが擬似透視投影である．擬似透視投影ではサブスクリーン内は並行投影で高速処理を行っており，スクリーン全体を考えれば透視投影になる．

図 28 の左が擬似透視投影を行った場合の視線の様子で右側がもとの透視投影の様子である．

$S \times S$ のスクリーンを $P \times P$ のサブスクリーンに分割する場合，$1 \leq P \leq S$ となっており，$P = 1$ の時が透視投影で，$P = S$ の時が並行投影に対応してい

る．図 28 左の擬似透視投影では $S:P=3:1$ の場合である．擬似透視投影は $S:P$ の比の値が数十倍がら数百倍と大きければ，透視投影に近い精度のよい絵が出ると考えられる．

VisA では S の値として 1024 や 2048 程度を考えている．また P の値としては $P \times P$ がプリフェッチの際の部分単位ブロックのサイズとなるくらいが妥当であると考えられる．部分単位ブロックのサイズを 4×4 とした場合は，$S:P$ の比の値が 256，あるいは 1024 といった大きな値となるため，精度のよい画像が期待できる．

第6章　まとめ

我々は，シミュレーションと連携した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を目指し，研究を行っている．環境の構築の方法として，可視化処理に対してどれだけの投資が可能かに応じて, いくつかの方法が考えられる．本稿ではその中で，専用ハードウェアVisA による並列ボリュームレンダリング処理について述べた．

VisA ではボリュームデータをスライスに分割し，複数のスライスをサブボリュームとしてPC クラスタの各ノードにもたせるというデータ分割法で並列処理を行う．そのような並列処理環境において有用な，ピクセル値計算方法及び Early Ray Termination について述べた．

また可視化方法としては，ボリュームデータの 3 重化を行わない方法と 3 重化を行う方法を比較し，両者の優劣を検討した．

そして，VisA のコア部分であるピクセル値計算部分を XILINX 社の FPGA である Virtex-II を用いて設計した．ピクセル値計算部分の処理に Virtex-II の SelsevRAM ブロックを使用した高速乗算器を用い，それらを効率よく配置することで, メモリモジュールである DDR SDRAM の基本動作周波数である 133MHz で回路を動作させることができた．今後はこのピクセル値計算部分に加え，本稿で提案したプリフェッチ機構を実際に実装し，VisA のプロトタイプである VisA Pro を完成させる．

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.

数々の有用な御指導，御意見を頂いた，森 眞一郎助教授，五島 正裕助手に心から感謝致します.

本研究の共同研究者である高山 征大氏，丸山 悠樹氏，中田 智史氏，そしてコンピュータ工学講座計算機アーキテクチャ分野の諸氏に感謝致します.

また，VisA カードの製作に当たっては東京エレクトロンデバイス (株) の千野氏, 山田氏, 尾花氏, 小田島氏にご協力頂いた．諸氏に慎んで感謝いたします.

参考文献

[1] 對馬 雄次 他, “ボリューム・レンダリング専用並列計算機 ReVolver のアーキテクチャ”, 情報処理学会論文誌, 第36 巻, 第7 号, pp.1709-1718, 1995.

[2] 吉谷直樹 他, “ボリュームレンダリング専用並列計算機 ReVolver/C40 の性能評価”, 情処研報告, 99-ARC-132, pp.79-84, 1999.

[3] Philippe Lacroute and Marc Levoy, ”Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation,” Proc. of SIGGRAPH'94, pp.451-458, July 1994.

[4] 原瀬 史靖 他, “数値シミュレーション過程の実時間可視化を支援するハードウェア” , 可視化情報シンポジウム,2002.7.

[5] 山本 恭弘 他 “有限要素法を用いた心臓大動脈の触診シュミレーション”, 日本バーチャルリアリティ学会第6 回大会論文集, 2001.

[6] L.Chen, I Fujishiro, and K Nakajima, “Parallel Performance Optimization for Large-Scale Unstructured Data Visualization for the Earth Simulator”, *Proc. of Parallel Graphics and Visualisation*, pp.133-140, 2002.

[7] C. Rezk-Salama, K. Engel, M. Bauer, G. Greiner, T. Ertl, ”Interactive Volume Rendering on Standard PC Graphics Hardware Using Multi-Textures and Multi-Stage Rasterization”, In Proc. Eurographics/SIGGRAPH Workshop on Graphics Hardware, 2000.

[8] 山内, 他, “アクティブボリュームレンダリングに基づくシミュレーションステアリング,” 信学技報 CPSY2001-35, pp.1-8, 2001 年8 月.

[9] “M.Levoy. Efficient Ray tracing of volume data”, ACM Transactions on Graphics, Vol.9,No.3, 245-261.July 1990.

[10] 生雲 公啓 他,“サイクリックにデータを配置した並列ボリュームレンダリング処理における ERT の効果”, 情報技術フォーラム FIT 一般講演論文集 第3 分冊 pp.233-234,2002.

[11] 生雲 公啓 他.“実時間インタラクティブシミュレーションのための並列ボリュームレンダリング環境”, 情報処理学会関西支部 支部大会 講演論文集 pp.121-124,2002.

[12] XILINX “Virtex-II Platform FPGA handbook,” December 2001.

[13] Barthold Lichtenbelt, Randy Crane, Shaz Naqvi,“Introduction To Volume

rendering",Hewlett-Packard Company,1998.

[14] 長谷川 裕恭, "VHDL によるハードウェア設計入門",CQ 出版 (株),1995.

[15] 並木 秀明, 永井 亘道,'VHDL によるディジタル回路入門", 技術評論社 (株),2001.

[16] 金 喜都 他, "ピクセル並列処理によりボリューム・レンダリング向きの超高速専用並列計算機アーキテクチャ, 情報処理学会論文誌,Vol.38, No.9, pp-1668-1680,1997.