計算機アーキテクチャ 97−15
（1992. 12. 4）

ボリューム・レンダリング向き並列計算機のアーキテクチャ

明石 英也　北須賀 輝明　薄田 昌広
森 眞一郎　中島 浩　富田 眞治

京都大学工学部

科学技術計算などによって得られる等間隔3次元配列を，ボリューム・レンダリング法によって表示する専用計算機のアーキテクチャを提案する．本計算機は，科学技術計算の可視化において必要不可欠である，遠近法による表示や従来の専用計算機で考慮されていなかった半透明なボリュームのリアルタイム表示を目標としている．

このため，本計算機では，単純なレイ・キャスティング・アルゴリズムと，特殊な構造を持つ3次元メモリを採用している．これにより，任意の方向の視線と交差するボクセルを同時に3次元メモリから取り出し，ピクセル値の計算を1メモリ・アクセス・サイクルで処理する構造を持つ．

本稿では，レイ・キャスティング・アルゴリズムと3次元メモリ，そしてピクセル値の計算方法について述べる．

A Parallel Computer Architecture for Volume Rendering

Hideya Akashi　Teruaki Kitasuka　Masahiro Susukita
Shinichiro Mori　Hiroshi Nakashima　Shinji Tomita

Department of Information Science, Kyoto University

A parallel computer architecture for scientific volume rendering is proposed. The goal of this architecture is to provide real time perspective projection of scientific volume, as well as the ability to treat with translucent volume.

These features are heavily required in the field of scientific visualization and these make this architecture distinguishable from previously proposed ones.

In order to provide these features, the architecture adopts the simple volume ray casting algorithm so that we can construct 3D memory which can simultaneously provide all voxels corresponding to a ray in arbitrary direction. Getting all voxels from 3D memory at each memory cycle, pixel value is calculated in pipelined manner.

The detailed description of the algorithm, the 3D memory and the pixel value calculation are presented in this paper.

1 はじめに

近年のスーパ・コンピュータおよびワークステーションの高速化にともない, 膨大な計算を必要とする流体解析, 構造解析などが比較的高速に行なえるようになってきた. これらの問題は, 通常差分法または有限要素法によって計算される.

ボリューム・レンダリングは, このような科学技術計算によって得られる膨大な量のデータを, 分かり易い形でユーザに提示し, 洞察, 理解の機会を増やすという点で, 重要なテクノロジーである.

また, 医学分野においては, CT スキャナなどの発展によって人体の断面をサンプルリングすることが可能である. このような断面を立体的に表示し医療援助を行なうという点においても, ボリューム・レンダリングは重要な技術である.

そこで, 本稿では科学技術計算などによって得られたデータの表示をリアルタイムで行なうことを目的とする専用並列計算機のアーキテクチャを提案する.

本稿では, 2章で本アーキテクチャの設計方針について述べた後, 3, 4章で本アーキテクチャの構成および高速化の手法について述べる. その後, 5章で, 本アーキテクチャにおいて, 高精度な表示を行なうための, レイ・キャスティング・アルゴリズムについて述べる.

2 設計方針

2.1 ボリューム・データのモデル

本稿において, ボリューム・データは, 科学技術計算などによって得られた 3 次元データ空間を, 等間隔でサンプリングした各格子点ごとにデータの属性値（材質, 温度, 密度など）が与えられているデータ構造である. 各格子点の属性値をボクセル値とよぶ.

ボリューム・データの処理方法に応じ, 以下の 2 種類のモデルを考える.

1) **離散モデル:** 各ボクセル値を中心とした単位立方体をボクセルとよび, ボクセル内の任意の点でのデータの属性値は, ボクセル値と等しいとみなす.
2) **連続モデル:** 8 つの頂点各々が, 独自のボクセル値を持つ単位立方体をボクセルとよび, ボクセル内でのデータの属性値は, 線形性をもつとみなす. このモデルにおいて, ボクセル内のある点のデータの属性値を, フィールド値とよぶ.

本稿では, 4章までは, ボリューム・データを離散モデルとして説明を行なう. そして, 5章では, 連続モデルについての説明を行なう.

2.2 アーキテクチャの要件

本アーキテクチャは, 科学技術計算などによって得られるデータを, 分かりやすい形で高速に表示することを目的とする. そのために, 以下の 3 つの要件を備えなければならない.

1. 半透明なボリューム表示

ボリューム全体を把握するために, ボリュームの内部を半透明な表示によって眺めることができなければならない.

これを可能とするために, ピクセルの色（ピクセル値）P を, 視線と最初に交差するボクセルの色にするのではなく, 視線と交差する全てのボクセルの色から計算する機構を設ける.

2. 遠近法による表示

科学技術計算の中には, 地震波の解析 [8] など, 大きな空間を処理するものがあり, これらのアニメーションを作成する場合には, 遠近法による表示が必要となる.

これを可能とするために, 視点とピクセルを通る視線を高速に生成する機構を設ける. CUBE architecture[5] や, PARCUM II[3] のように, 視線が平行であることを積極的に利用した方法は使えない.

3. リアルタイム表示

ボリューム・データを解析するため, 理想的にはボリュームの回転, 断面の表示などがインタラクティブに行えることが望ましい. 特に, リアルタイム・アニメーションを行なうためには, 視点の移動にともなうボリュームの再表示がリアルタイムで可能でなければならない.

リアルタイム処理を可能とするため, 並列処理によってボリューム・レンダリングを高速化する.

2.3 設計方針

前節で述べた要件を満足するため, 以下の3つを本マシンの設計方針とする.

- レイ・キャスティング法に基づくピクセル値計算.
- ボクセル並列処理によるピクセル値計算の高速化.
- 完全なボクセル並列処理が可能な3次元メモリの実現.

以下では, 各々の設計方針について簡単な考察を行なう.

2.3.1 ピクセル値の計算方法

ピクセル値の計算を行なう方法は, 大きく分けて以下の3通りがある.

ピクセル順
: 各ピクセルを通る視線を, ボリュームに対してキャストすることによりピクセル値を計算する.

ボクセル順
: 各ボクセル毎に, スクリーンのどのピクセルに投影されるか判定し, ピクセル値を計算する [7].

ハイブリッド
: ピクセル順, ボクセル順の処理を混合したもの.

半透明なボリューム・データの表示を行なう場合, 視点からボリューム・データを見た時のボクセルの順序関係が非常に重要な要素となるため, ボクセル順のアルゴリズムは不向きである. したがって, ボリュームに対して視点からレイ・キャストを行なうピクセル順のアルゴリズムを採用する.

SCOPE[9], GODPA[2], PARCUM IIは, ピクセル順のアルゴリズムを採用しているが, 半透明ボクセルの処理を考慮していない. また, CUBE architectureでは半透明ボクセルについても考慮しているが, ピクセル値を高速に演算する機構がないため, 表示を高速に行なうことができない.

これに対し, 本マシンでは, 4.2節で述べるような, 半透明なボリューム表示に必要なピクセル値演算機構を備え, 高速な表示処理を可能とする.

2.3.2 並列処理の方式

次に, レイ・キャスティングを並列処理によって高速化する方法を, 以下の3種類に分類し, 考察した.

ピクセル並列処理
: ピクセルの処理を1PEに割り当て, 各PEは視線方向にボリューム・データを探索しながらピクセル値を計算する.

ボクセル並列処理
: 視線と交差する全ボクセルのボクセル値をメモリ・ユニットから並列に取り出し, 全ボクセル値からピクセル値を高速に計算する [4].

ボリューム並列処理
: ボリューム空間を分割し, 各サブ・ボリュームの処理を1PEに割り当て, 最終的に画像をマージする [2][7][9].

ピクセル並列処理において, 任意の方向に対してボリュームを投影可能とするためには, ボリューム・データをPE間で共有する必要がある. このため, 論理的な共有メモリを実現しなければならず, ハードウェア量が多くなる. また, アクセス競合を完全になくすことはできないため, PE台数が増加すれば実現は非常に困難になる.

ボリューム並列処理においては, 各PEの生成した画像から, 視点とボリュームの距離 (**Z値**) に応じてピクセル値を計算するために, 複雑な処理が必要となる.

このため, 本アーキテクチャでは, ボクセル並列処理によって高速化を図る.

2.3.3 ボクセル並列処理の方式

ボクセル並列処理を行なう上で, 解決すべき問題点として, (1) 視線と交差する全ボクセルのメモリ・ユニットからの並列読み出し, (2) メモリからの並列読み出し速度に応じたピクセル値計算の高速化, がある.

(1) の問題点を解決するには, ボリューム・データを多数のメモリ・バンクに分割して格納し, 任意の視線に交差する全ボクセルを, バンク・コンフリクトなしに同時に読み出す機構が必要となる.

これを実現するため, 我々は, レイ・キャスティング・アルゴリズムと, 3次元メモリ構成法の両面から検討を行なった.

アルゴリズムとしては, 視線のサンプリングを, 当該視線の方向ベクトルの成分の絶対値が最大となる座標軸（以後**主軸**と呼ぶ）上で, 等間隔（ボクセル間の距離）で行なうものを採用する. これにより, 任意の視線によってサンプリングされるボクセルの座標は, 主軸方向の座標が全て異なるという性質をもつことになる.

この性質を利用して, 任意の視線に交差するボクセルを, バンク・コンフリクトなしにアクセスできる3次元メモリを実現する.

視線をバンク・コンフリクトなしにアクセスするためには, この視線の主軸と垂直な平面をなすボクセル群を全て別のメモリ・バンクに格納すればよい. よって, 任意の視線をバンク・コンフリクトなしにアクセスするためには, 主軸として x, y, z 軸の3種類が考えられるので, x, y, z 軸と垂直な各々の平面をなすボクセル群を一つのメモリ・バンクに格納すればよいことになる.

この構造は, ボリューム・データの3倍のメモリ容量を必要とするが, 非常にシンプルな構造となるという長所をもつ.

(2) の問題については, ツリー構造の演算器をパイプライン動作させることによって, ピクセル値を1メモリ・サイクル毎に出力できる機構により解決した.

3 マシンの概要

3.1 全体構成

2章で述べた設計方針に基づくボリューム・レンダリング向き並列計算機の構成を図1に示す.

(1) SCU(System Control Unit)
ホストマシンから制御コマンドを得て, 視点の移動, シェーディング・パラメータの変更等の情報を各部へ伝え, システム全体を制御する. また, ホストから送られるボリューム・データを, 3次元メモリ（PVM, 3.2節参照）に分配する.

(2) RDP(Ray Dispatch Processor)
SCU から視点の位置／方向, 視野の広さの情報を得て, 各ピクセル毎にこれを通過する視線がボリュームと交差するかどうかを判定し, 交差するならば視線のパラメータを PVM に与える.

(3) PVM(Parallel Volume Memory)
RDP から視線の情報を得て, ホストマシンから転送されたボリューム・データの中から, 視線と交差するボクセルのボクセル値を並列に読み出す. そして, 3.2節の式 (1) に従ってピクセル値を計算し, ピクセル値と Z 値を SP に与える. PVM の構成に関しては, 次節で詳述する.

SP(Screen Processor)
PVM から送られるピクセル値からスクリーンを構築し, SCU から光源の位置や陰影付けのパラメータを得て, depth gradient shading[2][7] やヒストグラム平坦化などのフィルタ処理を行ない, CRT に表示する.

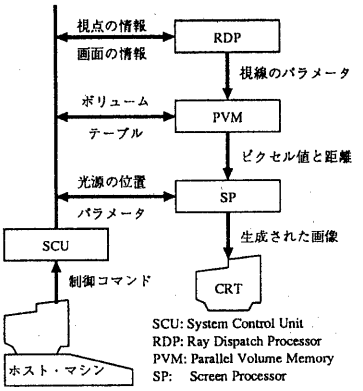

図 1: 本システムの構成

3.2 PVM の構成

本節では, 本アーキテクチャの最大の特徴である PVM の構成について述べる.

PVM は, 1) 任意の方向の視線と交差するボクセル列をなす各ボクセルの座標を計算し, ボクセル値を読み出す VMU(Volume Memory Unit), 2) 各ボクセル値からボクセルの色 c と透明度αを生成する c/α LUT(Look Up Table), 3) 各ボクセルの色と透明度からピクセル値を計算する, ツリー構造の VCP(Voxel Content Processor) からなる（図2参照）.

ボリューム・データは x, y, z 各座標軸方向にそれぞれ n 個のボクセルからなる立方体であり, 離散モデルと, 連続モデルの2種類を使い分けることが

できる．両方のモデルにおいてボクセル値は 1Byte で表現され，ボリューム全体は n^3個のボクセル値を使って表される．また，ボリューム・データは，n 個の VMU に分割して格納される．

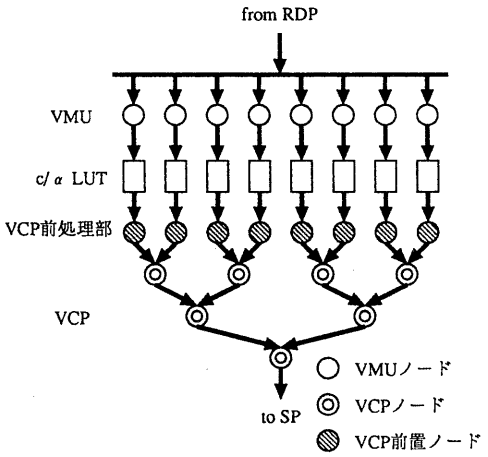

図 2: PVM のアーキテクチャ

4.1節で示すように，VMU は 1 サイクル毎に視線と交差する全ボクセル値 v_i（視線の方向により，視点に近い順または遠い順に $i = 0, 1, \cdots, n-1$ とする）を出力する．これらのボクセル値から，ピクセル値は，以下のように計算される [1][6][8].

ボクセル値 v_iが視点に近い順に $i = 0, 1, \cdots, n-1$ となる場合，

$$\mathrm{P} = \sum_{i=0}^{n-1} K_i \prod_{j=0}^{i-1} \alpha(v_j) \qquad (1)$$

ただし，$c(v_i)$ は，ボクセル値に対応する色

$\alpha(v_i)$ は，ボクセル値に対応する透明度

$K_i = c(v_i)(1 - \alpha(v_i))$

ただし，$K_i = c(v_i)$ とする場合もある [1].

ボクセル値 v_iが視点に遠い順に $i = 0, 1, \cdots, n-1$ となる場合，K, v の添字 i, jを，それぞれ $n-1-i, n-1-j$に置き換えれば良い．

c/α LUT では，ボクセル値によって c LUT および α LUT をテーブル参照することより，式 (1) の計算に必要な $c(v_i), \alpha(v_i)$ を求める．ボリューム表示における透明度の与え方としては，1) 透明度がボクセル値の関数となるモデル，2) 透明度が隣接するボクセル値の差の関数となるモデルの両方をサポートする．

また，5章で述べるように，視線方向のサンプル間隔Δd は視線の方向によって異なる．このため，Δd の値によって c およびαを補正する必要がある．補正後の c', α'は，$c' = c\Delta d, \alpha' = \alpha^{\Delta d}$で近似される．実際には，この計算を行なうかわりに，Δd の値によって c/α LUT のテーブル参照位置を変更する．

c/α LUT は，以上の処理を行なうため，図 3の構成とする．

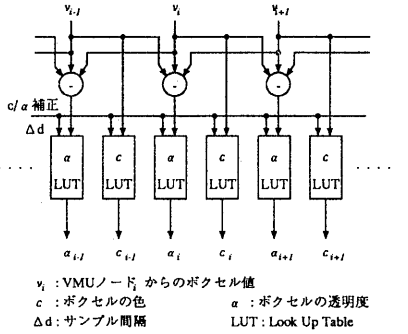

図 3: c/α LUT の構成

4 ボクセル並列処理のための高速化手法

4.1 3 次元メモリによるバンク・コンフリクトの解消

ボリューム・データは，前述の通り，3 重化して n 個の VMU ノードに格納される．VMU ノードの構成を図 4に示す．

本計算機では，視線の増分の中で，もっとも絶対値が大きいものを主軸とし，主軸と垂直なボクセル値群のなす平面 n 枚と視線との交点に存在するボクセルのボクセル値を視線上のボクセル値とする（図 5 参照）．

これら全てのボクセル値を同時に出力するために，VMU ノード i は，ボリューム・データのうち，$x = i$,

$y = i$, $z = i$ の各平面に属する全ボクセルを，それぞれ X memory bank, Y memory bank, Z memory bank に格納する．この時，ボクセル値 (l, m, n) は，VMU ノード l の X memory bank, VMU ノード m の Y memory bank, VMU ノード n の Z memory bank の 3 箇所に格納される．

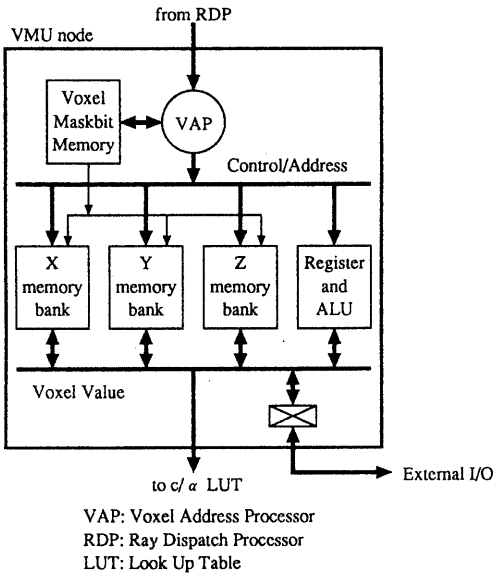

図 4: VMU ノードの構成

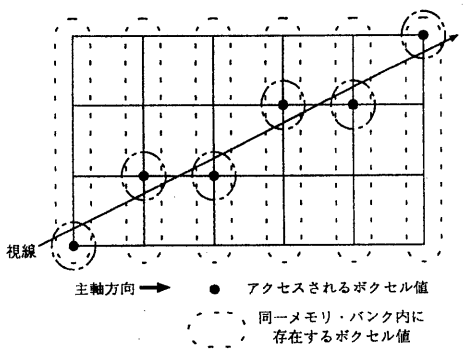

図 5: 視線と交差するボクセル値のアクセス

主軸が x 軸である場合の，視線と交差するボクセルの並列アクセスの方法を以下に示す．

1. RDP において，視線と $x = 0$ および $x = n$ の両平面との交点 $((y_0, z_0)$ および $(y_n, z_n))$ を求め，$\Delta y = \frac{y_n - y_0}{n}, \Delta z = \frac{z_n - z_0}{n}$ を計算し，(y_0, z_0), $(\Delta y, \Delta z)$ および主軸の識別子と方向を PVM の全 VMU ノードに対して与える．
2. VMU ノード$_i$は，$x = i$ での，$y = y_0 + i\Delta y$, $z = z_0 + i\Delta z$ を求め，x memory bank をアクセスして，ボクセル値 v_i を得る．ここで，ボクセル座標がボリューム外である時，または，ボクセルがマスクされている時にはボクセル値 0 を出力する．

主軸の方向が正方向である時には，得られるボクセル列は視点に近い方から順に VMU ノード$_0$, VMU ノード$_1$, $\cdots$, VMU ノード$_{n-1}$ より出力され，主軸の方向が負方向である時には，得られるボクセル列は逆の並びとなる．このため，前節で述べたように，VCP, c/α LUT は視線の主軸の方向が正逆いずれの場合についても，ピクセル値を計算することができるように構成している．

VMU ノードの Voxel Maskbit Memory は，ボリュームの断面を表示する時など，ボクセルを一時的に透明にするために使われる．マスク・ビットの設定は，RDP の支援の下で，VAP によって行なわれる．

VMU ノードのレジスタ付 ALU（**RALU**）は，5 章で述べるボクセル値の補間を行なうためのものである．

4.2 ピクセル値計算のパイプライン処理

VCP は，c/α LUT から出力されたボクセルの色および透明度から，3.2節の式 (1) に従ってピクセル値を計算するプロセッサである．

いま，$K(m,l)$ および$\alpha(m,l)$ を以下のように定義する．

$$\left.\begin{aligned} K(m,l) &= \sum_{i=\frac{nl}{2^m}}^{\frac{n(l+1)}{2^m}-1} K_i \prod_{j=\frac{nl}{2^m}}^{i-1} \alpha(v_j) \\ \alpha(m,l) &= \prod_{j=\frac{nl}{2^m}}^{\frac{n(l+1)}{2^m}-1} \alpha(v_j) \end{aligned}\right\} \quad (2)$$

ただし，$m = 0, 1, \cdots, \log_2 n - 1$, $l = 0, 1, \cdots, 2^m - 1$ とする．この時，$K(m,l)$ および$\alpha(m,l)$ は，更に，

$$\left.\begin{aligned} K(m,l) &= K(m+1, 2l) + \\ &\quad \alpha(m+1, 2l) K(m+1, 2l+1) \\ \alpha(m,l) &= \alpha(m+1, 2l)\alpha(m+1, 2l+1) \end{aligned}\right\} \quad (3)$$

と表現できる.

ここで, 式 (1) の P は $K(0,0)$ に他ならず, かつ, K_iおよび$\alpha(v_i)$ はそれぞれ $K(\log_2 n, i)$ ならびに$\alpha(\log_2 n, i)$ に他ならない.

したがって, 式 (3) を再帰的に適用することで, $K_i, \alpha(v_i)$ から P= $K(0,0)$ が求められることがわかる.

今, 式 (3) に相当する 1 回分の再帰計算を行なうノード (VCP ノードと呼ぶ) は, 図 6の構成で実現できる. よって, これを図 7のように, 逆ツリー状に結合することで, ピクセル値 P を求めるプロセッサを構成することができる.

この時, $K(x,l_i)$ と $K(x,l_j)$ $(l_i \neq l_j)$ は独立であるので, 幅方向は並列に, $K(m,l)$ と $K(m+1,2l)$ および $K(m+1,2l+1)$ は依存関係があるので, 深さ方向はパイプライン的に処理することで, P の値を毎サイクル連続して出力することができる.

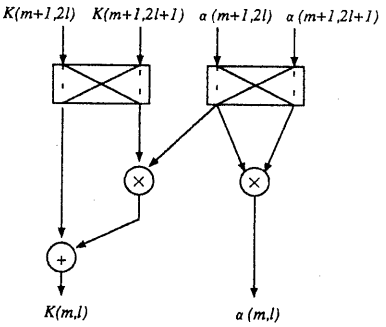

図 6: VCP ノードの構成

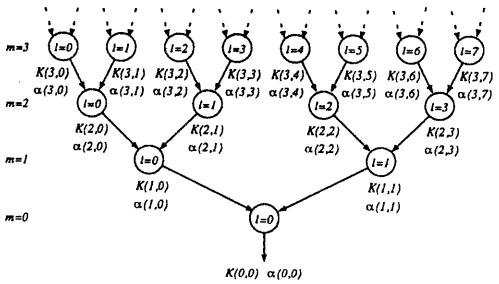

図 7: ピクセル値計算の再帰性

5 レイ・キャスティング・アルゴリズムの改良

4.1節で, 離散モデルにおけるレイ・キャスティング・アルゴリズムについて述べた.

本節では, 離散モデルと比較して, 速度が低下するがより表示精度の高いレイ・キャスティング・アルゴリズムを提案する.

レイ・キャスティングにおいて, ボリューム表示の精度を良くする方法として, tri-linear 補間法 [6][10] がある. この方法は, 連続モデルにおいて, レイ・キャスティング時にサンプル点を含むボクセルの 8 つのボクセル値から, tri-linear 補間を行なってサンプル点のフィールド値を求めるものである. このレイ・キャスティング・アルゴリズムは, 視線方向に対して, 一定のサンプル間隔で処理を行なう.

本アルゴリズムは, これを改良して, 主軸方向に対して一定のサンプル間隔で処理を行なうようにしたものである. サンプル間隔はボクセルの 1 辺の長さと等しくしており, その結果, 全サンプル点はボクセルの側面上に位置する. このため, サンプル点におけるフィールド値 v_iは, このボクセル側面の頂点をなす 4 つのボクセル値から以下のように補間することにより求められる (主軸が Z 軸の場合).

サンプル点の座標の小数部を (x_i, y_i), サンプル点を含むボクセル側面の 4 頂点の座標 (サンプル点の座標の整数部を引いた値) とボクセル値をそれぞれ $((0,0),v_{i,0}),((1,0),v_{i,1}),((0,1),v_{i,2}),((1,1),v_{i,3})$ とおく. この時, サンプル点におけるフィールド値 v_i は,

$$\begin{aligned} v_i &= \{(1-x_i)(1-y_i) - 2x_iy_i\}v_{i,0} + x_i(1-y_i)v_{i,1} \\ &\quad +(1-x_i)y_iv_{i,2} + x_iy_iv_{i,3} \end{aligned}$$

となる.

本アルゴリズムの長所は, 以下の 2 点である.

- 従来のアルゴリズムは 8 ボクセル値から補間を行なうが, 本アルゴリズムでは 4 ボクセル値から補間を行なう. よって, メモリ・アクセス量は半分になる. また, 本アルゴリズムは, 従来のアルゴリズムと比較して, 大幅に演算量を削減することができる.
- 各 VMU ノードは, ノード内のメモリ・バンクから 4 ボクセル値をアクセスすれば良く, VMU

ノード間でデータを受渡しする必要がない. このため, VMU ノードに, メモリ・バンクのアクセスと ALU における計算をパイプライン処理する機構を設けることで, 高速にフィールド値を求めることができる.

短所としては, 視線の方向がボリュームの対角線方向に近付くにつれて, 本アルゴリズムと従来のアルゴリズムのサンプル距離の差が開くことが挙げられる.

そこで, 本アーキテクチャでは c/α LUT において色と透明度を補正することにより, この差を補なっているが, これによる画像の差は今後の検討課題である.

6 おわりに

ボリューム表示を高速に行なうアーキテクチャを提案した. 本アーキテクチャの特徴は, 1) ボリューム・メモリを 3 重化した 3 次元メモリ, 2) ツリー構造の演算器によるピクセル値計算の高速処理, 3) 主軸等間隔サンプリングによるレイ・キャスティング・アルゴリズムの採用, である. これにより, 遠近法による表示や半透明なボリュームの表示を高速に行なう. このアーキテクチャによるボリューム表示の速度は, 画面の総ピクセル数を $512\times512 = \frac{1}{4}$Mpixel, メモリ・アクセス時間を 100ns とすると, $\frac{1}{4}\text{Mpixel} \times 100\text{ns} = 25\text{ms/frame}$ となる. 実際には RDP は, ボリュームと交差する視線のみを PVM に送るので, この時間はより短くなる.

さらに, 高精度な表示を行なうためのボクセル値の線形補間法を示した. この手法は, 主軸と垂直な平面内の 4 つのボクセル値からサンプル点のフィールド値を計算するため, VMU ノード間の通信が必要ない.

今後は SCU, RDP, SP の各機能ユニットについて検討した後, 128^3ボクセルのプロトタイプの設計を行なう予定である.

[謝辞]

日頃から熱心に御指導, 御討論いただく京都大学工学部富田研究室の皆様ならびに, 設計支援システムを御提供いただいた NTT コミュニケーション科学研究所並列処理アーキテクチャ研究グループ 中村行宏氏に感謝致します. なお, 本研究は一部, 平成 4 年度重点領域研究 (1)「超並列ハードウェア・アーキテクチャの研究」の補助を受けている.

参考文献

[1] R. A. Drebin, L. Carpenter and P. Hanrahan: Volume Rendering, Computer Graphics, Vol. 22, No.4, pp. 64-75, 1988.

[2] S.M. Goldwasser: A Generalized Object Display Processor Architecture, IEEE Compuuter Graphics & Applications, 4, 10, pp. 43-55,Oct. 1984.

[3] D. Jackel and W. Strasser: Reconstructing Solids from Tomographic Scans -The PARCUM II System-, Advances in Computer Graphics Hardware II, pp. 209-227, 1988.

[4] A. Kaufman and R. Bakalash: Memory and Processing Architecture for 3D Based Imagery, IEEE Computer Graphics & Applications, 8, 6, pp. 10-23, Nov. 1988.

[5] A. Kaufman, R. Bakalash and D. Cohen: Viewing and Rendering Processor for a Volume Visualization System, Advances in Computer Graphics Hardware IV, pp. 171-178, 1991.

[6] M. Levoy: Display of Surfaces from Volume Data, IEEE Computer Graphics & Applications, Vol. 8, No. 5, pp. 29-37, 1988.

[7] Toshiki OHASHI, Tetsuya UCHIKI and Mario TOKORO: A Three-Dimensional Shaded Display Method for Voxel-Based Representation, EUROGRAPHICS 85, pp. 221-232, 1985.

[8] P. Sabella: A Rendering Algoritm for Visualizing 3D Scalar Fields, Computer Graphics, Vol. 22, No. 4, pp. 51-58, 1988.

[9] 内木 哲也, 所 真理雄: 3 次元メモリを用いた立体図形表示機構 - SCOPE -, 電子通信学会論文誌, Vol.J68-D, No. 4, pp. 741-748, 1985.

[10] C. Upson and M. Keeler: V-BUFFER: Visible Volume Rendering, Computer Graphics, Vol. 22, No.4, pp. 59-64, 1990.