内容梗概

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より，膨大な計算を必要とする科学技術計算を実用的に行うことができる計算機環境が整いつつある．これにともない科学技術計算によって得られる膨大な量の結果から，直接コンピュータの画像を生成する手法として，ボリューム・レンダリングと呼ばれる手法が重要視されている．

ボリューム・レンダリング法は，内部の情報を伴う3次元のオブジェクトをわかりやすい形でユーザーに提示することができ，計算結果の解析に非常に有用な手法である．しかし，計算量が大きいため，インタラクティブ処理やリアルタイム処理は困難である．

そこで，本研究では科学技術計算によって得られる等間隔3次元配列の可視化を目的とした，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を提案した．本計算機は，科学技術計算の可視化において不可欠である，半透明なボリューム表示，遠近法による表示，リアルタイム表示の3つの表示能力を持つ．特に，半透明なボリューム表示をリアルタイムで行う能力は，従来の専用計算機では考慮されていなかったものであり，本計算機の最大の特徴となっている．

本計算機では，半透明なボリューム表示を高速に行うために，任意の視線と交差する全ボクセル値を同時に取り出し，全ボクセル値からピクセル値を高速に計算するボクセル並列処理方式を選択した．また，任意の視線と交差する全ボクセル値を同時に取り出すために，簡略化されたレイ・キャスティング・アルゴリズムとボリューム・データを3重に格納する3次元メモリ構造を考案した．さらに，全ボクセル値からピクセル値を高速に計算するために，1メモリ・アクセス・サイクル・ピッチで動作する，ツリー構造のパイプラインによるピクセル値演算機構を採用した．

これらにより，ボリュームの大きさに関わらず，40ms程度の時間で，任意の位置にある512×512ピクセルのスクリーンに対して，半透明なボリューム表示を行うことができる．

また，本研究では3次元メモリの各メモリ・バンクを制御するために，VMU chip（Volume Memory Unit）を開発しており，本稿ではこのチップの機能について述べている．VMU chipは，メモリ・バンク内のボクセルのアクセスの他に，ボリューム・データの転送，ボクセル・マスク・ビットの制御，補間演算の実行を行う機能を持つ．

また，本計算機では，視線生成部とスクリーン処理部に汎用DSPを用い，アルゴリズムの選択に柔軟性を持たせている．これにより，現在研究されている多くのボリューム・レンダリング・アルゴリズムを実装することが可能となっている．本稿では一例として，tri-linear補間法を簡略化して本計算機へ実装する方法を提案している．

本計算機で採用した主軸等間隔サンプリング・アルゴリズムによって生成される画像の質を評価するために，本稿では汎用並列計算機AP-1000の上に本アルゴリズムを実装し，種々の画像を生成した．これらの画像の比較により，簡単な補正によって良好な画像が生成できることや，本計算機に実装した補間法がtri-linear補間法と同程度の能力を持つことが明らかになった．

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

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科学技術計算が比較的高速に行えるようになってきている．そして，計算機による科学技術計算は，化学，数学，気象学，地質学，天体物理学，流体力学，構造力学などの分野において，非常に重要な役割を果たしている．科学技術計算における問題の多くは，差分法または有限要素法によって計算され，3次元空間内の各位置に対応するスカラ値またはベクトル値が結果として得られる．

科学技術計算によって得られる3次元データ空間は，膨大な量のデータ値からなり，データ

値を直接観察して解析することは非常に困難である. そのため, これらのデータ値を, 直接２次元の画像データに変換するボリューム・レンダリングと呼ばれる手法が重要視されている. ボリューム・レンダリングとは, ３次元データ空間を直接視覚化し, わかりやすい形でユーザに提示し, 結果に対する洞察, 理解の機会を増やすことを目的とする手法である.

一方, 医学分野では, CT スキャナや MRI などの発展によって人体の断層面を得ることが可能となっている. このような断層面を立体的に表示し医療援助や手術シミュレーションを行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役割を果たしている.

ボリューム・レンダリングは, 以上に述べたように大変広い応用範囲を持つが, データの表示に時間がかかるため, インタラクティブな使用やリアルタイム処理は困難である.

そこで, 本研究ではボリューム・レンダリングを高速化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, まず 2章で関連研究について述べた後, 新しいボリューム・レンダリング専用計算機の必要性について述べる. 次に 3章で本計算機のアーキテクチャの設計方針について述べ, 4章で本計算機の構成について述べる. 5章では, 本計算機の中心となる PVM(Parallel Volume Memory) の設計について述べる. また, 6章では, AP-1000 上で行った本計算機のアルゴリズムによる生成画像の評価について述べる.

2 ボリューム・レンダリング法

2.1 ボリューム・ビジュアライゼーションの概要

科学技術計算や医学分野において, 内部の情報を持つ３次元の物体を画像データに変換するボリューム・ビジュアライゼーションが注目されている. ボリューム・ビジュアライゼーションとは, 従来の３次元コンピュータ・グラフィックスが３次元空間内の０次元, １次元, あるいは２次元の幾何図形（プリミティブ）を表示対象としているのに対し, 内部の情報をもつ３次元の物体（ボリューム）を表示対象とする３次元コンピュータ・グラフィックスの手法である. この手法により, 従来のコンピュータ・グラフィックスの手法においては見ることのできなかった物体の内部を探索し, 解析を行うことができる.

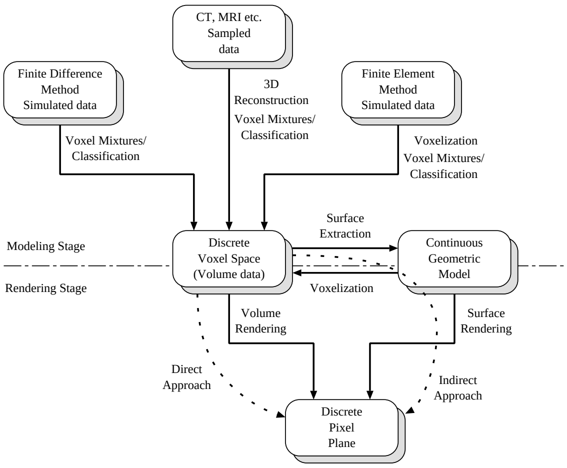


図 1: ボリューム表示の処理フロー

図 1にボリューム・ビジュアライゼーションの処理の流れを示す [3][7]. ボリューム・ビジュアライゼーションの処理は, モデリング・ステージとレンダリングステージによって分けられる.

モデリング・ステージ

モデリング・ステージは, ３次元データからグラフィックの対象となるデータ・セット（ボリューム・データまたはボリューム）に変換するステージである. ３次元データのソースとしては, 差分法や CT スキャナ, MRI によって得られる規則的な等間隔３次元メッシュと, 有限要素法で扱われる四面体構造などの不規則な３次元データ, の２種類が挙げられる. ボリューム・データは, ボクセルと呼ばれる部分空間の集合として表され, 前者のデータは立方体のボクセルからなる等間隔３次元メッシュ構造のボリューム・データとなり, 後者のデータは, 不規則な立体構造を持つボクセルの集合からなるボリューム・データとなる. 最近では後者の不規則な構造を直接ボリューム・レンダリングする試みも行われているが [4][12], 本研究では, 扱いやすさや応用分野の広さから, ボリューム・データは等間隔３次元メッシュ構造とする.

レンダリング・ステージ

レンダリング・ステージは, ボリューム データから画像を生成するステージである. ボリューム・データから画像を生成する方法として, 直接生成法と, 間接生成法の2種類がある. 直接生成法は, ボリューム・データを構成する各ボクセルを直接画面に投影することにより画像を生成する方法であり, ボリューム・レンダリングと呼ばれる. 間接生成法は, ボリューム・データを2次元のプリミティブに変換し, 従来のサーフェス・レンダリング法[1]を用いて画像を生成する方法である. 間接生成法では従来技術を使える利点があるが, 幾何プリミティブに変換するときに誤差が生じる場合があるため, 本研究では直接生成法 (ボリューム・レンダリング法) を用いることとする.

本研究で扱うボリューム・データは, モデリング・ステージの項で述べたように, 3次元データ空間を等間隔にサンプリングした各格子点ごとに, 科学技術計算などによって得られたデータの属性値 (材質, 温度, 密度など) が与えられているデータ構造であるとする. 各格子点の属性値をボクセル値と呼ぶ. なお, 有限要素法などによるデータ構造をボリューム表示する場合には, あらかじめ3次元等間隔メッシュ構造に変換しなければならない.

本研究ではボリューム・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: 各格子点を中心とした単位立方体をボクセルと呼び, ボクセル内の任意の点でのデータの属性値は, 格子点の値である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呼び, 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, フィールド値と呼ぶ.

本稿では, 特にことわりのない限り, ボリューム データを離散モデルとして説明を行っている.

2.2 関連研究

2.2.1 ボリューム・レンダリングのアルゴリズム

ボリューム・レンダリングのアルゴリズムは, ピクセル順アルゴリズム, ボクセル順アルゴリズム, およびハイブリッド・アルゴリズム, の3種類に分類することができる.

ピクセル順アルゴリズム

画面の各ピクセルから視線をボリュームに向かって発生させ, これと交差する全てのボクセルをピクセルに混合していくアルゴリズムである. このため, (ボリューム・) レイ・キャスティング・アルゴリズムとも呼ばれる. この方法では, 画面に投影されるボクセルの大きさがピクセルの大きさよりも小さい場合, ボクセルの一部がピクセルに反映されない. このような場合にも精度の高い画像を得るために, 以下のような方法が提案されている.

1. サンプルされる点におけるデータ値 (フィールド値) をサンプル点を囲む8ボクセル値から線形補間を行って求める方法 [14].
2. 全ボクセル値が反映されるように, 視線を分割していく方法 [15].
3. 各ピクセルから複数の視線を発生する方法 [18].
4. ボリューム・データを階層構造で表現し, 各階層の全ボクセルのデータ値をあらかじめ合成しておき, ボリュームに対するピクセルの広がりに応じた大きさをもつ階層からデータ値を得る方法 [18].

ボクセル順アルゴリズム

ボリュームの各ボクセルを画面に投影していくアルゴリズムである. この時, ピクセル順アルゴリズムの場合とは逆に, ボクセルの画面に投影される大きさがピクセルの大きさよりも大きい場合には, 1つのボクセルが複数のピクセルに反映される必要がある. このため, Westover のアルゴリズム [23] では, 各ボクセルが画面に投影される際に, 一定の広がりを持たせている.

---

[1] 3次元空間内の平面または曲面図形の集合から画像を生成する技法

ハイブリッド・アルゴリズム

ピクセル空間, ボクセル空間の両方において処理を行うアルゴリズムである. Drebin のアルゴリズム [2] では, 視線方向とボリュームの座標軸の一つが一致するようにボリュームを変換する（ボクセル空間における処理）. その後, ピクセルを通る視線と交差するボクセル列からピクセルの輝度（ピクセル値）を計算していく（ピクセル空間における処理）. Upson のアルゴリズム [22] では, ボクセルごとに現在処理中のスキャンラインと交差する部分を求めた後（ボクセル空間における処理）, スキャンラインに含まれる全ピクセルから視線を発生させる（ピクセル空間における処理）.

2.2.2 ボリューム・レンダリング専用計算機

ボリューム・レンダリングを用いて 3 次元空間内部の解析を行う場合, インタラクティブに処理を行うことができれば有用である. しかし, ボリューム・レンダリングは非常に処理量が多く, 時間がかかる. 一例として, Sabella のアルゴリズムの場合, Sun 4 で $64 \times 64 \times 400$ ボクセルのボリューム・データから 256×256 の画像を生成するために, 450 秒の時間がかかると報告されている [17]. このように処理量の多いボリューム・レンダリングを, リアルタイム処理するためには, ボリューム・データを 512^3 ボクセル, 表示速度を毎秒 30 画面とし, 1 画面の処理毎に全ボクセル値をアクセスすると仮定すると, $512^3 \times 30 = 4\mathrm{G}$ ボクセル / 秒程度の処理速度が要求される. このような処理速度に対する要求を満たし, ボリューム・レンダリングのリアルタイム処理やインタラクティブ処理を可能とするために, 特に 3 次元等間隔メッシュを対象として表示する専用計算機が多数設計されている（表 1）. これらの専用計算機は, 何れも高速化のために並列処理を行っている. そこで, 本節ではボリューム・レンダリング専用計算機を, 並列処理方式によって分類する.

ボリューム・レンダリングの並列処理方式は, ボクセル並列, ピクセル並列, およびボリューム並列, の 3 種類に分類することができる（図 2）.

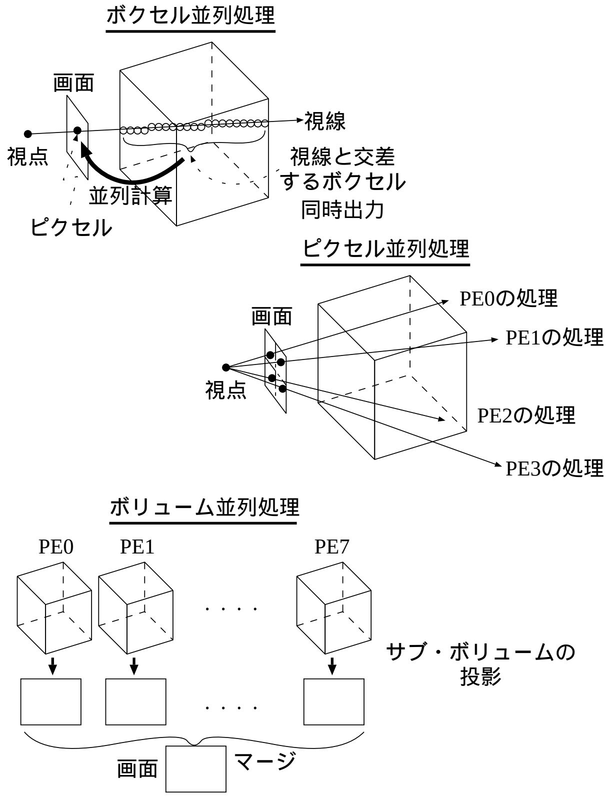


図 2: ボリューム・レンダリングの並列処理方式

ボクセル並列

レイ・キャスティング・アルゴリズムを用いて, 各ピクセルを通る視線と交差する全ボクセルを同時に取り出し, これらのボクセルの属性値（ボクセル値）からピクセル値を計算する方式である.

この方式においては, 視線と交差する全ボクセルをバンク・コンフリクトなしにアクセス可能なメモリ・システムと, 全ボクセル値からピクセル値を高速に計算する機構の設計が重要な課題となる.

CUBE[1][8][9][10] では, メモリ・システムの問題を, ボリュームの x, y, z 座標軸方向の 1 列のボクセル群を同時にアクセスできるスキュー・スキームを持つ多バンク・メモリを実現することで解決している. このメモリ・システムは, ボリューム・データを画面に平行投影する場合には, 視線方向がボリュームの座標方向と一致するようにボリュームを高速に回転した後, 視線と交差する全ボクセルをコンフリクトなしにアクセスして投影を行う. しかし, この方法では

表 1: ボリューム・レンダリング専用計算機

名称	アルゴリズム	並列処理方式	ボリューム	半透明表示	遠近法	速度
GODPA	Pixel-Based	ボリューム 並列	256^3 voxels	×	○	20-30 画面/秒
SCOPE	Pixel-Based	ボリューム 並列	—	—	○	—
3DP^4	Voxel-Based	ボリューム 並列	64^3 voxels ×256	×	○	10 画面/秒
PARCUM II	Pixel-Based	特殊な† ピクセル並列	512^3 voxels	×	×	—
CUBE	Pixel-Based (or Hybrid)	ボクセル 並列	512^3 voxels	△ ‡	△ ‡	投影 62ms 回転 511ms

† 4^3voxel 単位で読み書きを行う.

‡ リアルタイムでは不可.

遠近法のような各視線の方向が異なる場合の表示は難しく, CUBE では, 座標軸方向の投影の 10 倍程度の時間がかかる.

ピクセル値計算機構の問題は, 視点に一番近い不透明ボクセルを取り出すために, 各メモリ・バンク間の調停回路を実装することにより解決している. しかし, 半透明なボリュームの表示を行う場合には, 1 つの視線に対してこの調停回路を複数回使用する必要があり, 画像生成速度が遅くなる.

ピクセル並列

レイ・キャスティング・アルゴリズムにおいて, ピクセルの処理を 1 つの単位プロセッサ (**PE**) 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していく方式である.

この方式においては, 任意方向に対してボリュームを投影可能とするために, 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困難になる. そこで PARCUM II[6] では, ボリューム・データ内の任意の 4^3ボクセルの大きさのサブ・ボリュームを並列に読み出すことができるメモリ構造を実現し, 4×4 ピクセルを並列に処理することにより高速化している.

ボリューム並列

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, 各 PE はサブ・ボリュームを画面に投影し, 最終的に各 PE の画像をマージする方式である.

この方式においては, 各サブ・ボリュームの投影はデータの競合もなく独立に行える. 各 PE の画像をマージする時, 不透明なボリュームを表示する場合では, 各ピクセルごとに最も視点までの距離 (Z 値) が小さいピクセル値を最終画像のピクセル値とすればよい. しかし, 半透明な表示においてボクセルをマージする順序が規定されている場合には, 複雑な処理が必要となる. GODPA[5], SCOPE[21], 3DP^4[16] では, 不透明なボリューム表示を対象としているので, ツリー構造の比較器によって Z 値最小のピクセルを得る構造となっている.

2.3 従来の専用計算機の問題点と新しい専用計算機の必要性

現在研究されているボリューム・レンダリング専用計算機は, ボリューム・レンダリングをほぼリアルタイムで行える能力を持つ. しかし, これらの専用計算機には, 以下に述べる欠点がある.

1. 「半透明なボリューム表示や遠近法を用いた表示を高速に行なうことができない. 」
2. 「ボリューム・レンダリング・アルゴリズムとの親和性がない. 」

1の問題点は, これらの計算機のほとんどが医用画像の可視化を主目的として設計されてきたことによる. 医用画像においては伝統的に断面による内部表示が行われていたため, 半透明な表示はあまり必要とされていなかった. また, 遠近法についても医用画像の場合は必要とされない.

しかし, 半透明な表示ができない計算機を使って, 科学技術計算の結果を可視化し, 物体の内部を解析する場合, ボリューム・データの断面を表示させることにより探索を行う必要がある. このような探索による解析は, 半透明な表示による瞬時の理解と比較して, ボリューム・データの性質の理解しやすさを著しくそこなう. よってこれらの専用計算機は, 科学技術計算の可視化には向かない. また, 科学技術計算の可視化においては, 大きな空間を表すボリューム・データの表示を行う場合があり, 遠近法も必要とされる.

2の問題点は, ボリューム・レンダリング専用計算機では実行速度を上げることのみが目標とされている点にある. このため, 1の問題にも共通するが, 半透明表示や遠近法による表示を機構的に行うことができないものもある. また, ボリューム・レンダリング・アルゴリズムの研究において画質を向上するための種々の方法が提案されているが, これらを実装することは考えられていない.

科学技術計算の可視化におけるボリューム・レンダリングの必要性は, 非常に大きなものとなりつつある. また, 科学技術計算を指向した数々のアルゴリズムの存在も, ボリューム・レンダリングにとって大きな資産となっている. このため, 本研究では, 科学技術計算を指向し, 様々なアルゴリズムを選択可能でありながら, 従来の専用計算機の最大の特長であるリアルタイム処理が可能であるようなボリューム・レンダリング向き並列計算機を提案する.

3 設計方針

本章では, 3.1節で本アーキテクチャに要求される能力について述べた後, 3.2節で本アーキテクチャの設計方針を示す. また, 3.3節では, 本研究におけるボリューム・レンダリングの各種パラメータの定義を行っている.

3.1 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, わ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

2. 遠近法による表示

3. リアルタイム表示

半透明なボリューム表示は, ボリューム内部と透視しボリューム全体を把握することを可能とするため, 科学技術計算の可視化になくてはならない能力である. ピクセル順アルゴリズムにおいて, 不透明なボリューム表示を行う場合では, 視線と交差する最初の不透明ボクセルの色をピクセルの色とすればよい[6][8]. ボクセル順アルゴリズムにおいては, 各ピクセル毎にピクセルに投影されるボクセルのうちスクリーンまでの距離が最小となるボクセルの色をピクセルの色とすればよい[16]. これに対して, ピクセル順アルゴリズムで半透明なボリューム表示を行う場合には, 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必要がある. ボクセル順アルゴリズムでは, ピクセルに対応するボクセル値を適切な比率で配分する機構が必要となる.

遠近法による表示は, 地震波の解析[17]など, 大きな空間を処理する科学技術計算の可視化およびアニメーションを作成に不可欠である. ボリュームを画面に平行投影する場合には, 視線の方向をピクセル座標（ピクセル順アルゴリズム）またはボクセル座標（ボクセル順アルゴリズム）に関係なく一定に保てばよいが, 遠近法を使った表示を行う場合, 各ピクセル座標またはボクセル座標毎に異なる方向を求めなければならない. これを可能とするために, 異なる視線方向を高速に生成する機構を設ける. 従って,

視線生成の方法としては, CUBE や, PARCUM II 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.

リアルタイム表示は, アニメーション作成の効率をよくするために必要である. リアルタイム・アニメーションを行うためには, 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. また, ボリューム・データを解析するために, 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. 本計算機の目標とする, 512^3ボクセルからなるボリュームの半透明な表示をリアルタイムで行うためには, 2.2.2節で述べたように, 4G ボクセル / 秒程度の処理速度が必要となる. この処理速度の条件を満たすためには, 演算速度に対する制約以前にメモリ速度に対する制約がある. ボリューム・データは非常に大きいために DRAM を使用する場合, DRAM の速度を 8M アクセス / 秒と仮定すると, バンク・コンフリクトがない場合でも 512 バンク程度のバンク分けが必要となる. このため, 本研究の目標とする計算機では, 並列処理が不可欠となる.

3.2 設計方針

前節で述べた要件を満足するため, 以下の 3 つを本計算機の設計方針とする.

- 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う.

3.2.1 ピクセル値の計算方法

ピクセル値の計算を行う方法は, 大きく分けて 2.2.1 節で述べた 3 種類がある.

3.1節で示したように, 本計算機では, 半透明なボリューム表示を行うことを目的としている. 半透明なボリューム・データの表示を行う場合, 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. これは, 2.2.1節に示したアルゴリズムの多くが視点に近いボクセルをより強調する表示方法を採用していることからも明らかである. ボクセル順のアルゴリズムは, ボクセル同士の位置関係を直接知ることができないため不向きである. 従って, 本研究ではボリュームに対して視点からレイ・キャストを行う, ピクセル順アルゴリズムを採用する.

本計算機では, ピクセル順アルゴリズムにおいて半透明なボリューム表示を行うために必要なピクセル値演算機構を備えることで, 従来の専用計算機では実現されていない, 半透明なボリュームのリアルタイム表示を可能とする.

3.2.2 並列処理の方式

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は, 2.2.2節で述べたように, 3 種類ある.

本計算機は, 半透明なボリューム表示をリアルタイムで行うことを目的としている. この場合, メモリバンク数は 512 程度必要であると見積もった (3.1節参照). ピクセル並列処理をこの程度のプロセッサ数で行う場合, 2.2.2節で述べたようにボリュームの論理共有を行うか, または視線のパラメータを通信する必要があり, 実現が難しい. また, ボリューム並列処理においても, 半透明なボリューム表示のために, サブ・ボリュームを投影した画像を視点の位置によって異なる順でマージする, 複雑な処理が必要となる. これに対して, ボクセル並列処理は, 3.2.3節で示す問題の解決により, 高速化できることがわかった.

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.

3.2.3 ボクセル並列処理の方式

ボクセル並列処理を行う上で, 解決すべき問題点としては, 以下のものがあげられる.

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.

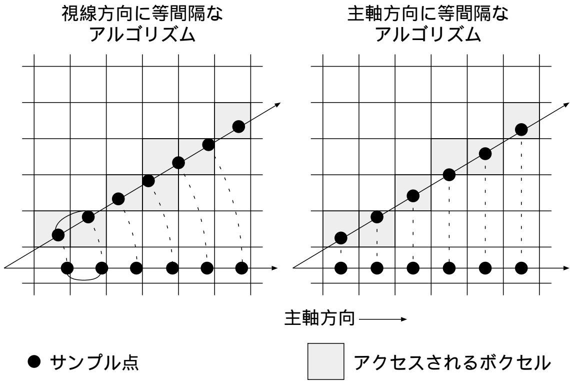


図 3: 主軸等間隔サンプリング ・アルゴリズム

ボクセル値の並列読み出し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読み出す機構があればよい.

これを実現するため, 本研究では, 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った.

通常のレイ ・キャスティング ・アルゴリズムでは, 視線と交差するボクセルのサンプリングを, 視線の方向に関わらず一定間隔で行う. このアルゴリズムの問題点は, 視線の方向によってアクセスされるボクセルの個数が異なり, 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た場合に, 視線と交差するボクセルの順序と各ボクセルに対応するメモリ ・バンクの位置との対応関係が, あるバンク関数によって関係付けられることである. この時, 各ボクセルの座標を計算する演算器とメモリ ・バンクの間に結合網が必要となり, 複雑になる.

そこで, 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（図 3参照）. 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.

この性質を利用すれば, 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ことができる.

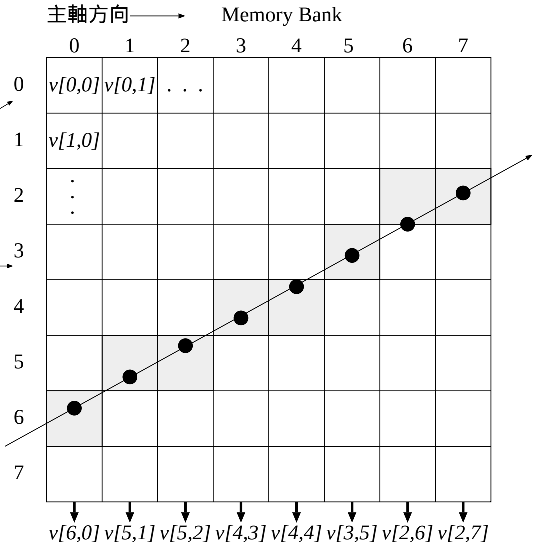


図 4: メモリ ・バンクの構成法

つまり, 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 ・バンクに格納することにより, 視線をバンク ・コンフリクトなしにアクセスすることができる（図 4参照）. したがって, 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持つ.

一方, ピクセル値計算の高速化の問題については, 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.

3.3 基本アルゴリズムとパラメータ定義

本節では, 本研究で採用しているアルゴリズムで使用する各種パラメータを定義し, アルゴリズムの説明を行う.

各種パラメータを表 2に, その位置関係を図 5に示す. ボリューム ・データは一辺に n 個のボクセル値がならんでおり, ボクセル値 n 個か

表 2: ボリューム ・レンダリング ・アルゴリズムの各種パラメータ

記号	意味	記号の内容
n	ボリュームの一辺の大きさ	—
$v(x,y,z)$	(x,y,z) のボクセル値	—
$m(x,y,z)$	ボクセル (x,y,z) のマスク ・フラグ†	—
$c(v)$	ボクセル値 v に対応するボクセルの色	(R, G, B)
$\alpha(v)$	ボクセル値 v に対応するボクセルの透明度	—
px	スクリーンの x 方向のピクセル数	—
py	スクリーンの y 方向のピクセル数	—
$P(sx,sy)$	(sx,sy) のピクセル値	(R, G, B, Vdis , FVC)‡
$\vec{E}$	視点の位置ベクトル	(Ex,Ey,Ez)
$\vec{S}c$	視点からスクリーンの左上端へのベクトル	(Scx,Scy,Scz)
$\vec{S}h$	画面上の x 方向 1 ピクセル分の方向ベクトル	(Shx,Shy,Shz)
$\vec{S}v$	画面上の y 方向 1 ピクセル分の方向ベクトル	(Svx,Svy,Svz)
$\vec{P}$	画面上でのピクセル座標	(sx,sy)
$E\vec{P}$	ピクセル P 処理中の視線の方向ベクトル	(EPx,EPy,EPz)
$\vec{r_0}$	視線と主軸座標=0 平面との交点	(r_0x,r_0y,r_0z)
$\vec{r_n}$	視線と主軸座標=n 平面との交点	(r_nx,r_ny,r_nz)
$\vec{\Delta r}$	$\vec{r_n} - \vec{r_0}$	$(\Delta rx,\Delta ry,\Delta rz)$
$\vec{L_i}$	光源 i の方向ベクトル	(L_ix,L_iy,L_iz)
L_icolor	光源$_i$の色	(R, G, B)

† マスク ・フラグが 1 の時, 対応するボクセル値は 0 となり, 透明ボクセルとなる.

‡ FVC および Vdis は, 視線と最初に交差する透明でないボクセル値と視点からの距離である.

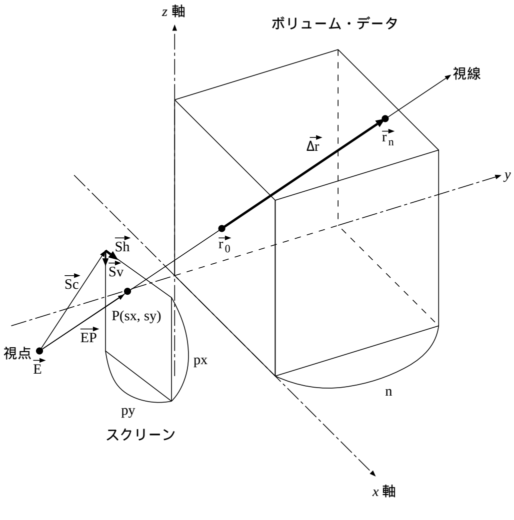


図 5: ボリューム ・レンダリングの各種パラメータ

らなる立方体で表されているものとする. ここで, ボリューム ·データが離散モデルの場合には, $v(x,y,z)$ は, 空間座標 $(x+0.5, y+0.5, z+0.5)$ でのデータの属性値 (ボクセル値) であるとする. 連続モデルの場合には, $v(x,y,z)$ は, 空間座標 (x,y,z) でのデータの属性値であるとする.

各ボクセル値に対応して, マスク ・フラグ $m(x,y,z)$ が定義される. このフラグは, 対応するボクセルが有効か無効かを示すビットであり, 1 の時は有効となるが, 0 の時は対応するボクセル値は実際の値に関わらず 0 とされ透明なボクセルとなる. ボクセルの色 $c(v)$ および透明度$\alpha(v)$ は, ボクセル値に対応して定義される.

スクリーンは, x 方向に px, y 方向に pyのピクセルを持つものとする. 各ピクセル Pに対して, ピクセル値 $P(sx,sy)$ が定義される. ここでピクセル値は (R, G, B, Vdis, FVC) の要素を持つ. R, G, B は, ピクセルの色の赤成分, 緑成分, 青成分であり, Vdis は, ピクセルを通る視線と交差する先頭の透明でないボクセルとスクリーンの距離を表す. ここで, このようなボクセルが存在しない場合には, 値 DMAX が代入される. また, FVC は, このボクセルのボクセル値である.

空間内に視点$\vec{E}$が定義され, この点からスクリーンを通してボリュームを観察する. スクリーンの全てのピクセルの空間座標における位置は, 空間座標における視点からスクリーン左上端へのベクトル$\vec{S}c$, x 方向 1 ピクセル分の方向ベクトル$\vec{S}h$, y方向 1 ピクセル分の方向ベクトル$\vec{S}v$によって表される. これらにより, あるピクセル $P(sx,sy)$ を処理中である時, 視線の方向ベクトル$E\vec{P}$は, $\vec{S}c + sx\vec{S}h + sy\vec{S}v$と計算される.

ボリューム ・レンダリングの基本アルゴリズムを図 6に示す. ボリューム ・レンダリングの

```
volume_rendering() {
        define volume, $\vec{E}, \vec{Sc}, \vec{Sh}, \vec{Sv}$;
        for($\vec{P} \in$ SCREEN) {
                $\vec{EP} = \vec{P} - \vec{E}$;
                define primary_axis($\vec{EP}$);
                define $\vec{r_0}, \vec{\Delta r}$;
/* get all voxels and calculate pixel value */
                $P(\vec{P})$=ray_cast($\vec{r_0}, \vec{r_n}, \vec{r_0}, \vec{\Delta r}$);
        }
        filtering(screen);
}
```

図 6: ボリューム ·レンダリングの基本アルゴリズム

処理の最初に，視点，スクリーンが定義される．本研究におけるボリューム ·レンダリングはピクセル順アルゴリズムを採用しているので，スクリーン内の各ピクセルのピクセル値を順に求めていく．各ピクセル値の算出のために，まずピクセルを通る視線の方向ベクトル$\vec{EP}$および主軸 primary_axis が求められる．また，$\vec{r_0}$および$\vec{\Delta r}$が視線の方向ベクトルと主軸の方向より求められる．

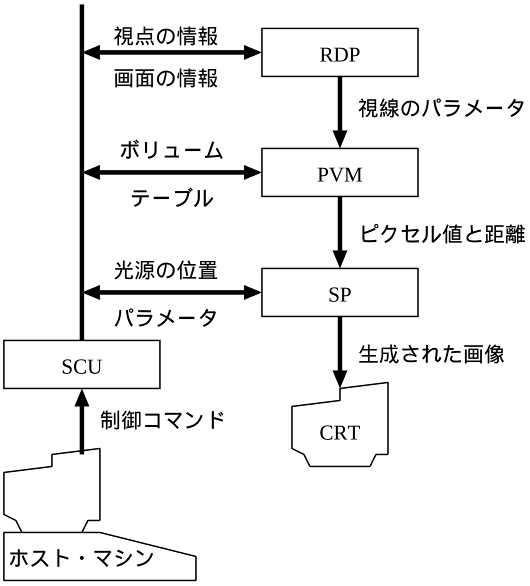
次に，これらの視線のパラメータを使って視線を生成し，この視線と交差する各ボクセル値からピクセル値を計算する関数が ray_cast である．ボクセル並列処理においては，視線と交差する全ボクセル値が同時にメモリ ·システムからアクセスされ，全ボクセル値を使ってピクセル値が高速に計算される．そして，全ピクセル値を計算済みのスクリーンに対して，シェーディングやフィルタ処理を関数 filtering によって行い，最終画像を生成する．

以上のアルゴリズムにおいて，本計算機ではボリュームの一辺のボクセル数 n，スクリーンの x 方向のピクセル数 px，スクリーンの y方向のピクセル数 pyは 512 であるとする．

4 本計算機の構成

4.1 全体構成

3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 7に示す．



SCU: System Control Unit
RDP: Ray Dispatch Processor
PVM: Parallel Volume Memory
SP: Screen Processor

図 7: 本システムの構成

本計算機は，以下に示す機能ユニットによって構成される．

(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．

(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，4.4節の式 (1) に従ってピクセル値を計算し，SP に与える．

(4) 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

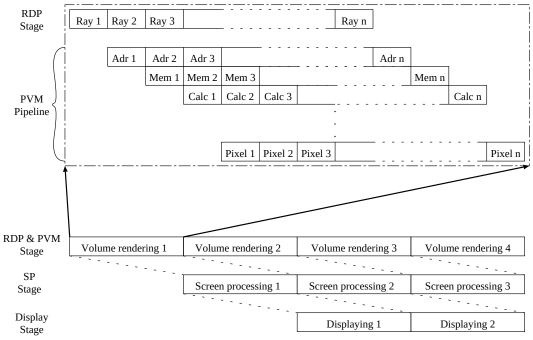


図 8: 本システムにおけるパイプライン処理

付けのパラメータを得て，depth gradient shading [5][16] やヒストグラム平坦化などのフィルタ処理を行い，CRT に表示する．

本計算機では，リアルタイム処理を可能とするために，機能ユニット間および機能ユニット内部においてパイプライン処理を行っている（図 8）．最上位レベルのパイプライン処理は，1 画面（フレーム）ピッチのパイプラインであり，RDP と PVM によるボリューム・レンダリング処理，SP によるスクリーン処理，および画像表示の 3 段パイプラインとなっている．2 番目のレベルのパイプライン処理は，RDP による視線生成処理と，PVM によるピクセル値計算処理の 1 ピクセル・ピッチのパイプラインである．PVM によるピクセル値計算処理は，ボクセル・アドレス計算，ボクセル・アクセス，ボクセル値計算，ピクセル値計算（多段）からなる．

ボリューム・データは非常に大きいため，ボリューム・データ格納用のメモリは DRAM を使う．このため，最下位レベルのパイプライン・ピッチを DRAM のサイクル・タイム程度である 160ns とする．また，SCU, RDP, SP は，Texas Instruments 社の汎用 DSP TMS320C40 をプロセッサとして使用し，25MHz(40ns) のクロックで動作させるものとする．

本章では以下，各機能ユニットの構成について述べる．

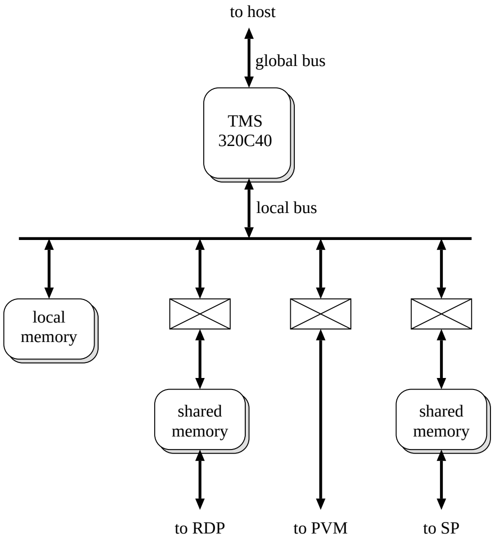


図 9: SCU の構成

4.2 SCU(System Control Unit) の構成

SCU は，システム全体の初期設定および制御を行う機能ユニットであり，TMS320C40 1 プロセッサで構成される．システムの初期設定時においては，同じく TMS320C40 で構成される RDP, SP にプログラムを転送する．また，PVM に対しては，各種の動作モードを設定する（4.4 節参照）．ボリューム・レンダリング処理中は，1 フレーム毎に画像生成のための各種パラメータを各機能ユニットに指定する．これらのパラメータには，RDP に対する視点とスクリーンの位置関係の指定，PVM の動作モード，ボクセル色／透明度テーブル，マスク・フラグの変更，SP に対する光源の変更などがある．

PVM のボリューム・データは非常に大きいため，描画処理をいったん停止しておき，SCU を通さずに，ホストマシンから直接 PVM に転送される．また，アニメーション処理時には，一画面間隔で RDP に視点とスクリーンの位置関係などのパラメータを与えながら画像を生成していく．このため，SP のフィルタ処理の終了判定として SP の同期検出，RDP に対する画像生成開始コマンドの送出機能が必要となる．

SCU の構成を図 9に示す．SCU はホストマ

表 3: SCU に関する各種パラメータ

記号	記号の内容	数値の型†
Host → SCU		
Hcmd	ホスト・コマンド	—
SCUprg	SCU プログラム	—
SCU → RDP		
$\vec{E}$	(Ex, Ey, Ez)	float
$\vec{Sc}$	(Scx, Scy, Scz)	float
$\vec{Sh}$	(Shx, Shy, Shz)	float
$\vec{Sv}$	(Svx, Svy, Svz)	float
RDPcmd	RDP コマンド	—
RDPprg	RDP プログラム	—
RDP → SCU		
RDPstat	RDP ステータス	—
SCU → PVM		
PVMmode	PVM 動作モード	—
$c(v)$	(R, G, B)	10.10
$\alpha(v)$	—	1.10
SCU → SP		
$\vec{L_i}$	(L_ix, L_iy, L_iz)	float
L_icolor	(R, G, B)	float
SPcmd	SP コマンド	—
SPprg	SP プログラム	—
SP → SCU		
SPstat	SP ステータス	—

!j†d1.d2 は, 整数部 d1 ビット, 小数部 d2 ビットの固定小数点であることを示す.

表 4: RDP に関する各種パラメータ

記号	記号の内容	数値の型†
RDP → PVM		
PVMcmd	PVM コマンド	—
axis	主軸識別子	2
axisdir	主軸方向	1
msk	マスク座標境界	10
mskdir	マスク方向	2
$\vec{r_0}$	(x_0, y_0) or (y_0, z_0) or (x_0, z_0)	16.4
$\vec{\Delta r}$	$(\Delta x, \Delta y)$ or $(\Delta y, \Delta z)$ or $(\Delta x, \Delta z)$	12.4
d_0	入射点の距離	16.8
Δd	距離の増分	2.8
$\vec{P}$	(sx, sy)	16

†d1.d2 は, 整数部 d1 ビット, 小数部 d2 ビットの固定小数点であることを示す.

シン, RDP, SP の間にそれぞれ共有メモリを持ち, この共有メモリを使って RDP に対するコマンド送出, SP の同期検出およびコマンド送出を行う. また, PVM に対しては, モード設定／テーブル転送用のデータ・バスを持たせる.

表 3に SCU が受渡しする各種パラメータを示す. PVM は, カスタム IC の使用を予定しているため, ハードウェア量の削減のために固定小数点演算を行う.

4.3 RDP(Ray Dispatch Processor) の構成

RDP は, 視線を生成し, PVM に対して視線と交差する各ボクセル値のアクセスを要求するユニットである. また, PVM 内のボリューム・データのマスク・フラグ設定命令の発行も行う. RDP が PVM に対して送出する各種パラメータを表 4に示す.

RDP において生成された視線は, パラメータ axis, axisdir, msk, mskdir, $\vec{r_0}$, $\vec{\Delta r}$によって表され, PVM に与えられる. ここで, axis は, 2 ビットで表される主軸識別子であり, 主軸が x 軸, y 軸および z 軸の時に, それぞれ 00, 01, 10 の値をとる. また, axisdir は, 視線の方向が主軸正方向および主軸負方向であることをそれぞれ 0 および 1 の値によって示す. msk, mskdir は, ボリュームの主軸座標が msk である点を起点とし, mskdir の示す方向にある全てのボクセルをマスクするために使われる. これらにより, 視点がボリューム内に存在する時に, 視点の後方の全ボクセルはピクセル値に影響を与えてはならないため, これらのボクセルを透明にすることができる. $\vec{r_0}$および$\vec{\Delta r}$は, 主軸方向成分を除いた 2 方向で表される. また, SP において, depth gradient shading を行うために, パラメータ d_0, Δd, $\vec{P}$も PVM に与えられる.

RDP によって指定される PVM の処理は, アイドル・サイクル, mask 設定サイクル, 視線処理サイクルの 3 つがあり, PVMcmd によって指定される.

RDP におけるピクセルごとの各種パラメータの算出方法を図 10に示す. 図では, スクリーン座標 (sx, sy) を通る視線を生成している. このアルゴリズムは以下の手順で行われる.

1. 視線ベクトルを, $\vec{Sc}$, $\vec{Sh}$, $\vec{Sv}$, $\vec{P}$を使って求める.
2. 主軸を決定する. ここで主軸とは, $\vec{EP}$の成分 (EPx, EPy, EPz) の絶対値が最大となる座標方向であり, 主軸方向の成分を EP'_p, 残り二つの成分を順に EP'_0, EP'_1としている. これにともない, 視点の座標$\vec{E}$ も並べ換えている.

3. 主軸方向基本ベクトル$\vec{EP''}$は主軸座標の増分が±1となるベクトルである．これは，$\vec{EP'}$を｜EP'_p｜で割ることにより求められる．
4. $\vec{r_0}$，視点$\vec{E}$と$\vec{r_0}$との距離d_0および$\vec{\Delta d}$を求める．
5. 視線をPVMにディスパッチする．ただし，視線がボリュームと反対方向の場合はディスパッチしない．

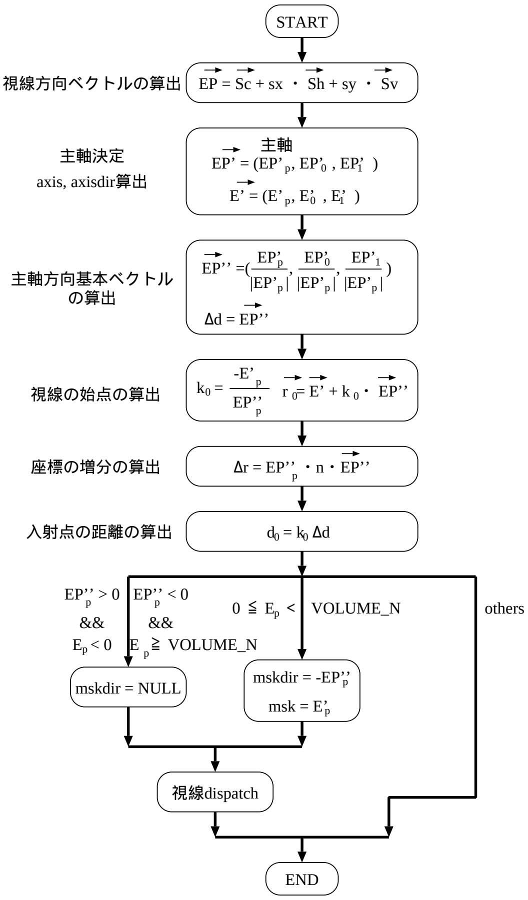


図 10: RDPの視線生成アルゴリズム

4.1節で述べたように，PVMは160nsサイクルで1視線の処理を行う事ができる．一方，RDPは，25MHzで動作するため，RDPは4サイクルで1視線の処理を行う必要がある．この速度でRDPを動作させるために，汎用プロセッサの並列化を仮定し，RDPで必要なプロセッサ数を見積もる．なお，RDPを1個のDSPと専用回路により構成する方法が，参考文献[11]で提案されている．

このアルゴリズムの各種演算の回数を数えると，乗算22回，加算，比較演算，絶対値演算18回，除算，ルート演算2回となっている．ここで，乗算，加算，比較演算，絶対値演算を1命令サイクル，除算，ルート演算を30命令サイクル[2]で実行できる汎用プロセッサを使用すると，1視線あたりの命令数は130命令となる．この時，4命令サイクルで1視線の処理を行うためには，$\frac{130}{4} = 32.5$個のプロセッサを並列に動作させる必要がある．ここでは，演算に必要なロード命令，ストア命令およびPVMのデータ・フォーマットへの変換などのコストは考慮していないため，RDPのプロセッサ数はさらに増加する．

RDPのハードウェア量の削減のために，本研究では，アルゴリズムの改良およびDSPの使用を考える．アルゴリズムの改良は，RDPの各要素プロセッサにスクリーンの横方向1ライン単位の処理を割り当てることによって行うことができる．これにより，視線の方向ベクトルの算出をインクリメンタルに処理できること，主軸の方向は1ラインあたり最大4回のみ変化するため，ピクセル毎の主軸算出処理をなくすことができること，の利点が得られる．そして，視点

---

[2] TMS320C30における除算の命令サイクルを基準とした．

の位置により動作プログラムを変更することにより, 比較演算を減らすことができる. また, 今回選択した DSP 汎用 DSP TMS320C40 により, 表 5に示す利点が得られる.

図 11に, TMS320C40 のアセンブラによるピクセル毎の視線生成プログラムを示す. 主軸方向毎に別のプログラムを用いるため, 主軸の各方向に対して同様のプログラムを用意する必要がある. この時, 主軸の各方向に対応するプログラムにおいては, 主軸方向基本ベクトルの$E\vec{P}''p$ (プログラム中では"EP2p"), axis, axisdir は定数となる. また, 視点の位置は 1 画面描画中には変更がないため, $\vec{E}p$, mask, mskdir も定数であるから, これらの値に関するものを定数としてはじめに計算しておくことにより, 実行命令数が短くなる. プログラム内の除算およびルート演算については, 文献 [19] で示されているプログラムを使用した. このプログラムは, 除算近似およびルート近似命令を使用し, 除算は 7 命令サイクル, ルート演算は 10 サイクルで行うことができる. 並列動作命令は, 図では || で結合されている 2 つの命令となっている. 並列動作可能な命令は, 乗算と加算, 乗算または加算とロードまたはストアなどがあり, 可能な組み合わせは限られている. 並列動作命令において間接アドレス指定を多用しているが, TMS320C40 内の RAM バンクからの読み出しについては, 1 サイクルに 2 回アクセスできるので, パイプラインのロックは起こらず, 1 命令サイクルで実行できる.

図のプログラムにおいては, 主軸方向の変更チェック, PVM に与えるデータのフォーマット変換, 1 ライン単位に実行する処理などは含まれていない. しかし, これらを合わせても, 先に述べた TMS320C40 の特長を利用して, RDP の視線生成処理は, 1 ピクセルあたり平均 50 命令程度で行えるものと思われる. よって, RDP のプロセッサ数は$\frac{50}{4} = 12.5$ 台程度あれば基本的な視線生成処理を行うことができる.

ここで, 画像の精度改善などのために, 基本アルゴリズムの拡張による RDP の処理量の増加に対処することを考慮し, 本計算機では, RDP は余裕を持たせて TMS320C40 を 16 台並列に動作させるものとする.

表 5: TMS320C40 の特徴

1. MMU がないため, メモリ・アクセスが高速である.
2. プロセッサ内部に 1 サイクルに 2 回アクセス可能な 4KB の RAM バンクを 2 バンク持つ.
3. 逆数の 16bit 近似, ルートの逆数の 16bit 近似を 1 命令で行えるため, 浮動小数点の除算, ルート演算がそれぞれ 7 サイクル, 10 サイクルで行なうことができる.
4. 乗算 + 加算, 乗算 / 加算 + ストア, ロード / ストア + ロード / ストアなどの並列動作命令により, 実行命令数を減らすことができる.
5. バスを 2 つもち, 通信ポートを 6 つ持つ.
6. DMA コントローラを 6 組持つ.

```
*
* TITLE RDP program inner loop(code for pixel)
*
* Arguments
* vec(EP)=(R8,*AR0,*+AR0(1))
*       vec(EP2)=(EP2p,R1 & *+AR7(1),R2 & *-AR7(1))
* vec(SV)=(*-AR2(1),*AR2,*+AR2(1))
*       vec(E)=(Ep,*AR3,*+AR3(1))
* *-AR5(1)=2.0, R3=1.0, *AR5=0.5, *+AR5(1)=1.5
* *-AR6(1)=EP2p*n, *AR6=-EP2p*Ep, *+A6(1)=—Ep—
* Outputs
* vec(R)=(r0,r1), vec(dR)=(dR0,dR1), d0, dd
*
DSPTCH ADDF3 R4,*-AR2(1),R4
         ABSF  R4,R6
         RCPF  R6,R1
         MPYF3 R1,R6,R0              —— ADDF3 *AR0,*AR2,R2
         SUBF3 R8,*-AR5(1),R0         —— STF   R2,*AR0
       MPYF3 R0,R1,R1          —— ADDF3 *+AR0(1),*+AR2(1),R2
         MPYF3 R1,R6,R0
         SUBF3 R0,*-AR5(1),R0         —— STF   R2,*+AR0(1)
         MPYF3 R0,R1,R1
         MPYF3 R1,R6,R0
         MPYF3 R0,*+AR0(1),R2
         MPYF3 R0,*AR0,R1             —— STF   R2,*-AR7(1)
         MPYF3 R1,R1,R0              —— STF   R1,*+AR7(1)
         MPYF3 *-AR7(1),*-AR7(1),R0  —— ADDF3 R3,R0,R2
         MPYF3 *-AR6(1),*+AR7(1),dR0 —— ADDF3 R2,R0,R0
         RSQRF R0,R7
         MPYF3 R0,*AR5,R0             —— STF   R7,*+AR7(1)
         MPYF3 R7,R7,R2
         MPYF3 R0,R2,R2
        MPYF3 R1,*AR6,R1            —— SUBF3 R2,*+AR5(1),R2
         MPYF3 R2,*+AR7(1),R7          —— STF   R1,*AR7
         MPYF3 R7,R7,R2
         MPYF3 R0,R2,R2              —— ADDF3 *AR3,*AR7,r0
        MPYF3 *AR6,R2,R1            —— SUBF3 R2,*+AR5(1),R2
         MPYF3 R2,R7,R7
       MPYF3 R2,*-AR6(1),dR1        —— ADDF3 R1,*+AR3(1),r1
         MPYF3 R0,R7,R7
         MPYF3 R7,EP2p,dd
         MPYF3 R7,*+A6(1),d0
```

図 11: RDP の視線生成プログラム (TMS320C40)

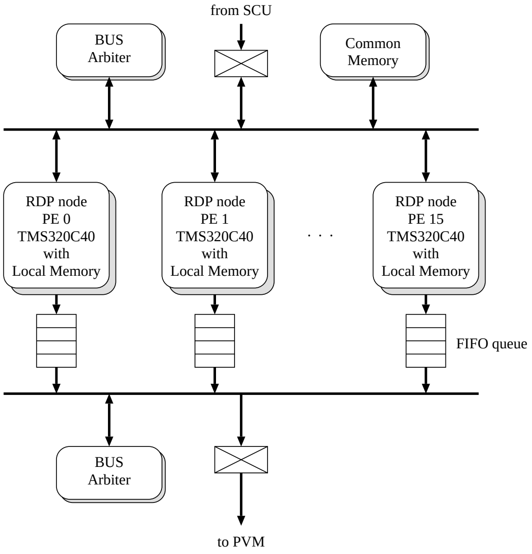


図 12: RDP の構成

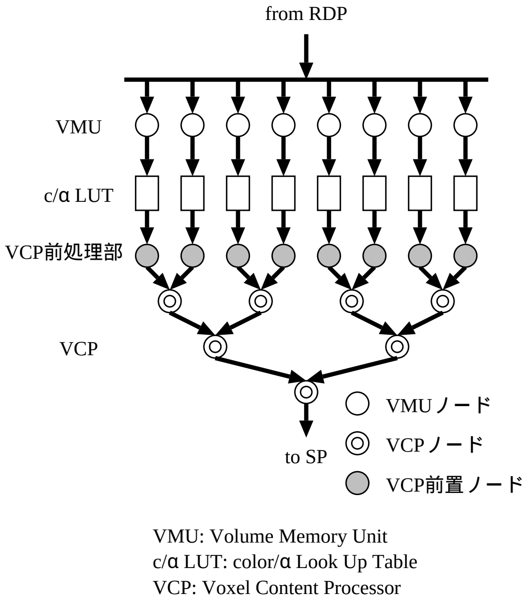


図 13: PVM の構成

RDP の各プロセッサ間で共有するデータとしては, SCU からのパラメータおよび次に処理するフレーム上ライン位置が挙げられる. これらの SCU からのパラメータは, 一フレーム生成中は変化しない. また, 次に処理するフレーム上のライン位置については, RDP の各プロセッサが 1 ラインの処理を終えた時に次のラインを得るために使う. このため, 各プロセッサのライン位置の参照は, 頻繁には起こらない. よって, 先に述べたように RDP のプロセッサ台数が 16 台と少ないこと, プロセッサ間の共有データの参照が稀であることから, RDP のプロセッサはバス結合によって SCU からのデータなどを共有すれば十分である.

RDP の構成を図 12に示す. プロセッサ TMS320C40 は, 内部バス, 外部バスと 2 つのバス持つので, 外部バスを結合して SCU, RDP 間の共有メモリを持たせ, 内部バスは PE のローカル・メモリに接続する. また, PVM とのインターフェイスは, 視線の生成速度が一定ではないため FIFO キューを介して行い, 生成された視線のパラメータを PVM へ送出する.

4.4 PVM(Parallel Volume Memory) の構成

PVM は, RDP の生成した視線と交差する各ボクセル値をメモリ・バンクから読み出し, 各ボクセル値からピクセル値を計算する機能ユニットである. この処理を高速に行うために, 以下の機能ユニットからなる（図 13参照）.

VMU(Volume Memory Unit)
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.

c/α LUT(Look Up Table)
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.

VCP(Voxel Content Processor)
ツリー構造により各ボクセルの色と透明度からピクセル値を計算する.

本稿では, 説明の便宜上, 4.4.1節で, ピクセル値の演算方法と題して c/α LUT について述べた後, 4.4.2節で VMU, 4.4.3節で VCP について述べる. また, PVM の設計については, 5章に詳細を述べる.

4.4.1 ピクセル値の演算方法について

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（2.1節参照）．両方のモデルにおいてボクセル値は 1 バイトで表現され，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．

4.4.2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 P は，以下のように計算される [2][14][17].

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，

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

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 [2].

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 $n-1-i, n-1-j$ に置き換えればよい．

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，透明度がボクセル値の関数となるモデル，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．前者のモデルでは，通常のボリューム・レンダリングを行なうことができ，後者のモデルでは，ボクセル値の変化の大きい部分を強調することができる．

また，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．サンプル間隔の違いが最大となるのは，視線が主軸方向と平行である場合と主軸方向と他の 2 つの座標方向がそれぞれ 45 度の角をなす時に起こり，最大 $\sqrt{3}$ 倍になる．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．透明度の大きい場合，補正後の c', α' は，$c' = c\alpha^{(\Delta d - 1)}, \alpha' = \alpha^{\Delta d}$ で近似される．実際には，この計算を行うかわりに，Δd の値によって c/α LUT のテーブル参照位置を変更する．

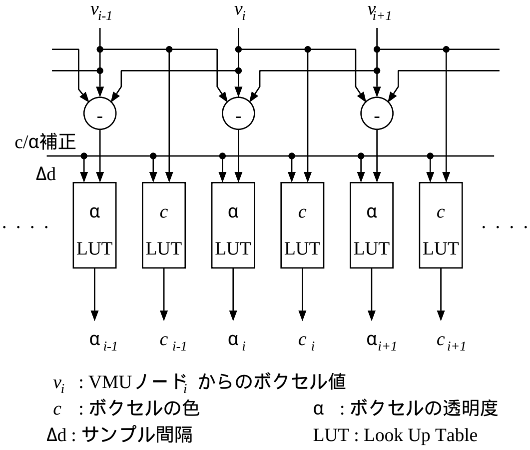


図 14: c/α LUT の構成

c/α LUT は，以上の処理を行うため，図 14 の構成とする．

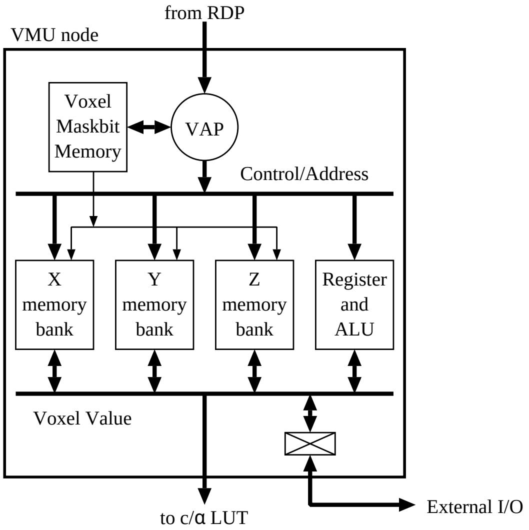
4.4.2 VMU の構成

ボリューム・データは，前述の通り，3 重化して n 個の VMU node に格納される．VMU node の構成を図 15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 3参照）．

これら全てのボクセル値を同時に出力するために，VMU node$_i$ 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 node$_l$ の x memory bank, VMU node$_m$ の y memory bank, VMU node$_n$ の z memory bank の 3 箇所に格納される．

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．



VAP: Voxel Address Processor
RDP: Ray Dispatch Processor
LUT: Look Up Table

図 15: VMUnode の構成

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 node に対して与える.
2. VMU node$_i$ は, VAP(Voxel Address Processor) により $x=i$ 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 VMUnode$_0$, VMUnode$_1$, $\cdots$, VMUnode$_{n-1}$ 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 によって行われる.

VMU ノードのレジスタ付 ALU （**RALU**）は, 5.1節で述べるボクセル値の補間を行うためのものである.

4.4.3 VCP の構成

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, 4.4.1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.

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

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

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,

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

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うノード （VCPnode と呼ぶ）は, 図 16の構成で実現できる. よって, これを図 17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

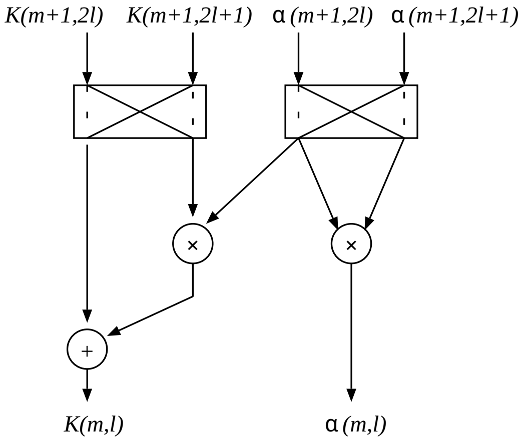


図 16: VCP ノードの構成

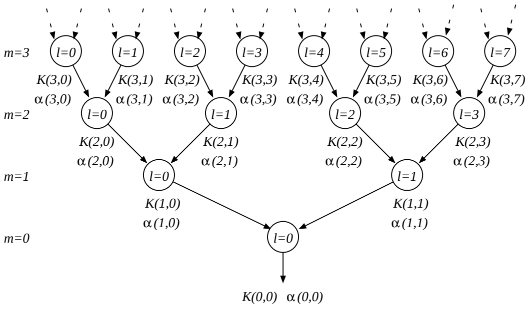


図 17: ピクセル値計算の再帰性

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.

4.5 SP(Screen Processor) の構成

SP は, PVM からのピクセル値データに従ってスクリーンを構築し, これにフィルタ処理を行なう機能ユニットである. SP で行なう重要なフィルタ処理としては, シェーディング処理が挙げられる. 一般にシェーディングとは, 物体と光源の位置関係から物体の表面に光沢あるいは陰影を与えたり, 他の物体を写し込んだりする手法であり, 3 次元形状の正確な表示, あるいは形状理解のための表示において必要不可欠である. レイ・トレーシング法においては, 視線と物体の交点における物体の法線を使ってシェーディングを行う. 一方, ボリューム・レンダリング法においては, ボリューム・データが元のデータを離散な座標でのボクセル値によって表したものであるため, 視線と物体の交点における物体の法線を近似によって求める必要がある.

本計算機におけるシェーディング処理は, 他のボリューム・レンダリング専用計算機において用いられている, depth gradient shading を採用する. depth gradient shading は, ボリューム・データを画面に投影した時に, 各ピクセル毎にピクセル値と Z 値を記憶しておき, 各視線と物体の交点における法線は, 視線を通るピクセルにおける Z 値と x, y 方向に隣接する数ピクセルの Z 値との関係により近似される方法である. この方法の利点は, ボクセル空間を探索することなく法線を求めることができるため, レイ・キャスティング処理とシェーディング処理を完全に分離し, パイプライン化できることである. 一方欠点としては, ボクセル空間を探索するシェーディング法と比較して, 法線の方向に誤差が出やすいことが挙げられる. しかし, 本計算機ではボリューム・データのリアルタイム表示を目標としているため, ボクセル空間を探索して法線を求めるアルゴリズムを実装するのは困難である. また, 本計算機においては 3 次元形状理解のためのシェーディングを目標としているので, depth gradient shading で十分目

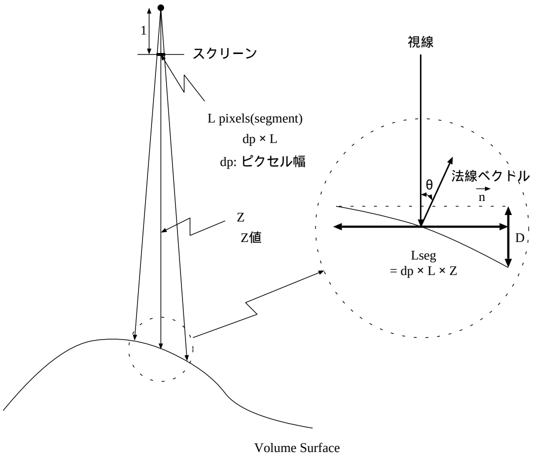


図 18: depth gradient shading

```
*
* TITLE SP program x co-ordinate loop
*
* Arguments
* z[x]=*AR0, theta[x]=*+AR3(IR1), left=AR2, right=AR3
* *-AR4(1)=dp
* R8=DMAX
*
01:XCORD CMPI  *AR0,R8           31: CMPI  *++AR3(1),R4
02:      BEQD  NEXT             32: LDIGE 0,R5
03:      LDI   AR0,AR2         33: ADDI  R5,AR6
04:      LDI   AR0,AR3         34: CMPI  R8,*+AR3(1)
05:      CMPI  *+AR0(1),R8      35: LDIEQ 0,R5
06:      BEQD  NEXT             36: CMPI  *+AR3(1),R4
07:      SUBI3 *AR0,*+AR0(1),R4 37: LDIGE 0,R5
08:      LDI   1,R5          38: ADDI  R5,AR6
09:      LDI   AR0,AR5         39: CMPI  R8,*+AR3(1)
10:      BNND  PLUS             40: LDIEQ 0,R5
11:      LDI   AR0,AR6         41: CMPI  *++AR3(1),R4
12:      ADDI  1,AR6          42: LDIGE 0,R5
13:      ADDI  1,AR3          43: ADDI  R5,AR6
14:      CMPI  R8,*-AR2(1)     44: SUBI3 AR5,AR6,R0
15:      LDIEQ 0,R5           45: ADDI  1,R0
16:      CMPI  *-AR2(1),R4     46: FLOAT R0,R0
17:      LDIGE 0,R5           47: MPYF  *AR0,R0
18:      SUBI  R5,AR5          48: MPYF  *-AR4(1),R0
19:      CMPI  R8,*-AR2(1)     49: RCPF  R0,R1
20:      LDIEQ 0,R5           50: MPYF3 R1,R0,R2
21:      CMPI  *-AR2(1),R4     51: SUBRF 2.0,R2
22:      LDIGE 0,R5           52: MPYF  R2,R1
23:      SUBI  R5,AR5          53: MPYF3 R1,R0,R2
24:      CMPI  R8,*-AR2(1)     54: SUBRF 2.0,R2
25:     LDIEQ 0,R5        55: MPYF3 R2, R2,R1——SUB *AR6,*AR5,R0
26:      CMPI  *-AR2(1),R4     56: MPYF  R1,R0
27:      LDIGE 0,R5           57: ST    R0,*+AR0(IR1)
28:      SUBI  R5,AR5
29:      CMPI  R8,*+AR3(1)
30:      LDIEQ 0,R5
```

図 19: SP の depth gradient 算出プログラム (TMS320C40)

標を達成できる.

depth gradient shading において, x, y方向にそれぞれいくつのピクセルの Z 値から法線を求めるか, という問題については, 参考文献 [16] で挙げられている. この研究によると, x, y方向それぞれ 7 ピクセルの Z 値から傾斜を求めればよいとしている. よって, 本計算機においても, x, y方向の各 7 ピクセル値から中心のピクセルにおける傾斜を求めることとする.

視点からスクリーンまでの距離を 1 と見た時のピクセルの幅を dp, ピクセル (sx, sy) の Z 値を $Z_{(sx,sy)}$, $D_k = Z_{(sx+1,sy)} - Z_{(sx,sy)}$とおくと, このピクセルにおける x 方向の傾斜は以下のようにして求められる (図 18).

1. $D_i (i = k + 0, k \pm 1, k \pm 2, k \pm 3)$ を求める.
2. D_iの中で, $D_i = 0$ または D_iの符号が D_k の符号と等しい, という条件を満たす, D_k を含む連続した D_iをセグメントとし, セグメントの長さを L とおく. セグメントの最左端の kの値を $k0$, 最右端の kの値を $k1$ とする.
3. $D = Z_{(k1+1,sy)} - Z_{(k0,sy)}$を計算する.
4. 傾斜値$\theta = \arctan(D/(dp \times L \times Z_{(sx,sy)}))$ を求める.

x 方向の傾斜を計算する TMS320C40 のアセンブラ ・プログラムを図 19に示す. このプログラムにおいては, 1 ピクセルあたり 60 命令程度で傾斜値のタンジェントが求められる. 同様に y 方向についても 60 命令程度で傾斜値のタンジェントが求められるので, ピクセルあたり最大 120 命令でボリュームの法線が求められる.

シェーディングはこれらの法線ベクトルと光源の方向ベクトルを使って行う. 1 ピクセルあたり最大 20 命令でシェーディングを行うことができるとすると, 光源数を lights とした時, SP の実行命令数はピクセルあたり $120 + 20 \times \text{lights}$ となる. よって, SP のプロセッサ数は lights=1 の場合$\frac{140}{4} = 35$ 台必要となる. ここで, SP はシェーディング処理の他にピクセル値平坦化などのフィルタ処理を行う必要があるため, 本計算機では SP は TMS320C40 を 64 台使用して画面分割によりシェーディング処理, フィルタ処理を行うこととする.

先に述べた傾斜を求めるアルゴリズムを使用した時, 隣接する部分画面を分担するプロセッサは, 各プロセッサが分割された画面の境界付近の傾斜値の計算時に, 他のプロセッサに割り付けられたピクセルを参照する必要がある. この場合にプロセッサ間でピクセルを共有するのを避けるため, 境界付近のピクセルについては,

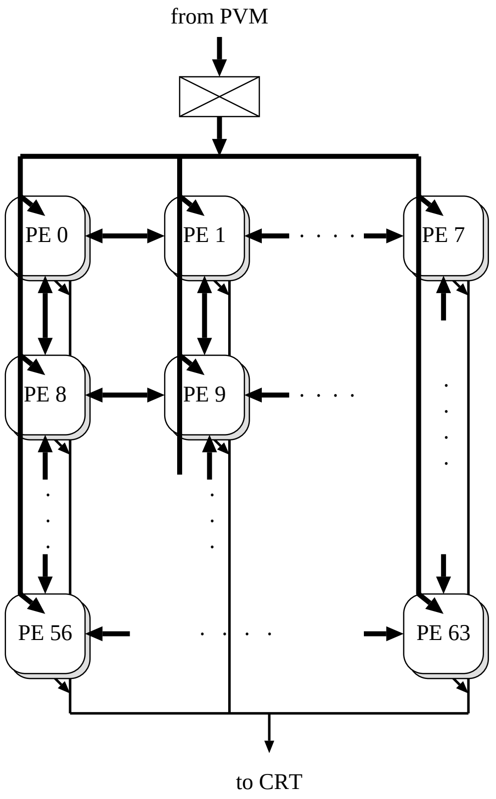


図20: SPの全体構成

PVMから出力された時点で, そのピクセルを必要とする全てのプロセッサがそのピクセルのデータを取り込めばよい. ただし, SPはフィルタ処理全般を行なうため, 汎用画像処理マシンの性格を持たなければならない. よって, SPの各プロセッサ同士は, 画像処理と親和性のよいメッシュ型相互結合網（あるいはトーラス型相互結合網）により結合する. この結合においては, 図5で示したTMS320C40の特長により, プロセッサの通信ポート4つをそれぞれ上下左右のプロセッサの通信ポートと直接接続し, DMAにより通信を行うことができる.

図20にSPの全体構成を示す. また, 図21にSPの要素プロセッサの構成を示す. SP要素プロセッサは, TMS320C40の通信ポートにより隣接する要素プロセッサに接続されている. SCUとSP要素プロセッサは, TMS320C40の外部バス上の共有メモリを使って通信を行う.

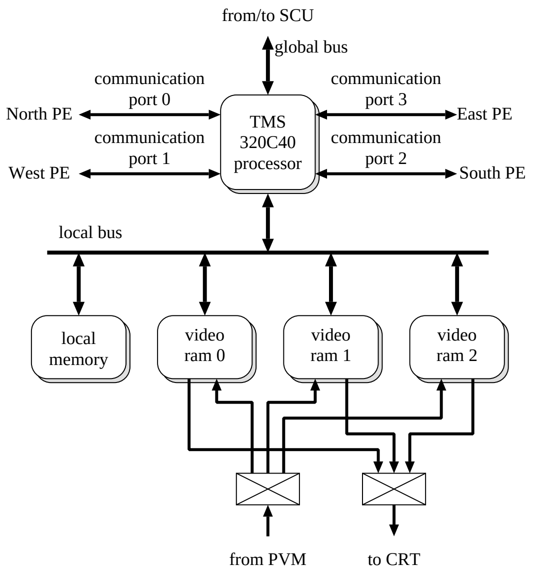


図21: SPの要素プロセッサ構成

また, SP要素プロセッサの内部バスには, ローカル・メモリの他にビデオRAMが3組接続されている.

これら3組のビデオRAMにより, PVMからのピクセル値書き込み, SPによるフィルタ処理, CRTへの表示, の3つの処理をパイプライン式に行うことができる.

5 PVMの設計

本章では, 本研究において設計した, VMU chipについての説明を行う. 本計算機の基本的な構成としては, 4章で述べた通りであるが, 実際に本計算機を設計するにあたって, より画質を向上させるために, 補間アルゴリズムの実装について考察した. よって, 本章では5.1節で補間アルゴリズムの実装について述べた後, 5.2節でVMU chipの機能について述べる.

5.1 補間アルゴリズムの実装

4.4.2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

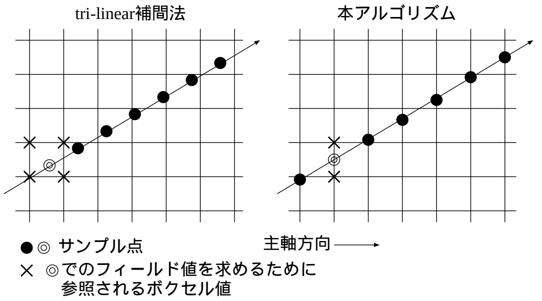


図 22: tri-linear 補間と本計算機における補間

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よくする方法として，tri-linear 補間法 [14][22] 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，図 22参照）．よって，本稿では 4 点補間法と呼ぶこととする．

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，

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

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うが，4 点補間法では 4 ボクセル値から補間を行う．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 VMUnode 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よく，VMUnode 間でデータを受渡しする必要がない．このため，VMUnode 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．

5.2 VMU chip の仕様

本節では，本研究において設計している VMU chip の仕様について述べる．VMU chip に要求される機能として，memory bank の DRAM リフレッシュ，視線と交差するボクセルの読み出し，ホスト計算機からのボリューム転送，ボクセル・マスクビットの設定がある．本計算機においては，全 VMU chip は同期している必要があり，50MHz での設計は困難であると予測されるため，VMU chip は 25MHz で動作させ，memory bank を 4 サイクル単位で制御する．DRAM を同期回路によって制御する場合，1 メモリ・アクセスに最低 8 サイクル程度必要であるから，VMU chip はディレイ・ラインの使用により DRAM 用の一部の信号線を生成することで 4 サイクルで 1 メモリ・アクセスを実現する．

DRAM リフレッシュ

DRAM のリフレッシュは，各 VMUnode を構成する VMU chip が，同期して VMUnode 内の memory bank にリフレッシュ信号を発することにより行われる．このため，外部から VMU

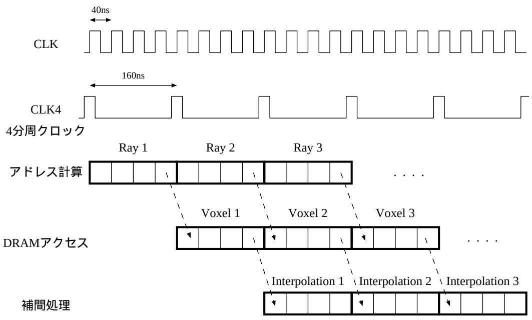


図 23: VMU chip におけるパイプライン処理

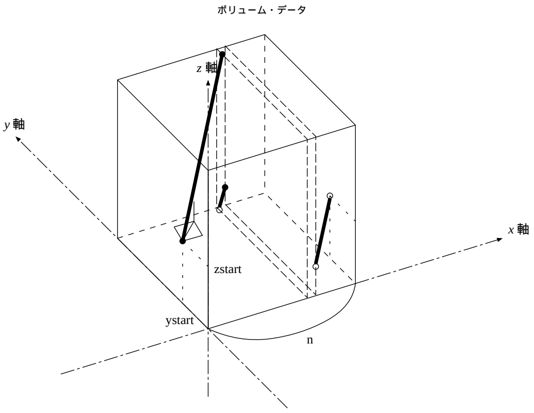


図 24: ボリューム転送時のパケットの幾何的位置

chip に対するリフレッシュ・リクエスト信号を与える.

ボクセル値の読み出し

視線と交差するボクセルの読み出しを行うためには, 視線のパラメータからボクセル・アドレスを計算する機能, ボクセル・アドレスを使ってDRAMを読み出す機能, 必要に応じて, 5.1節で示した補間を行う機能, が必要となる. VMU chip では, 高速化を図るために, これらの機能をパイプライン処理している.

図23に, VMU における各処理のパイプラインを示す. VMU chip には, 25MHz のクロックの他に, このクロックを 4 分周したクロック CLK4 を外部から与える. 離散モデルにおけるボクセル値アクセスでは4クロック・サイクル（160ns）単位でボクセル値を出力するが, 連続モデルにおけるボクセル値アクセスにおいては, 4 ボクセル値からフィールド値を求めるので, 16 クロック・サイクル単位でフィールド値を出力する. このため, VMU chip には, 視線のパラメータを読み出したことを外部に伝える信号線が必要となる.

ボリュームの転送

ホスト計算機からのボリューム転送については, ホストからのパケット転送によって行う. 本計算機では, ボリューム・データを 3 重化して持っているので, 転送速度をあげる工夫が必要となる. そこで, ヘッダとして $(\mathrm{ystart}, \mathrm{zstart})$ を持ち, パケットの $t(0 \le t \le n)$ 番目のボクセル座標が $(t, \mathrm{ystart}+t(\bmod\ n)), \mathrm{zstart}+t(\bmod\ n))$ である様なパケットを転送に使うことにする. この方法により, パケット内のボクセル座標は x, y, z ともに全て異なる, という性質を持つため, 1 パケットの転送で, 全ての VMU ノードの全ての memory bank にただ 1 回のみ書き込みが起こることになる（図24参照）. ボリューム・データ全体を転送するには, データ長が n バイトのパケットを $n \times n$ 回転送すればよい. ここで, メモリの速度以外に何も制約がないとすると, 1 パケットは 160ns の速度で転送できることになり, 512^3ボクセル値を転送するのにかかる時間は, 512^2パケット × 160ns = 40ms となり, リアルタイムでボリューム転送が可能となる. しかし, 今回の VMU chip については, 40ns で 1 ボクセル値を転送することとする. この場合のボリューム転送時間は, 512^2パケット $\times 512 \times$ 40ns =約 5 秒となる.

VMU chip におけるパケット転送は, 1 バイトのボリューム転送バスを用いて行われる. ホスト計算機は, CLK4 の立ち上がりとともに ystart, zstart（共に 2 バイト）をヘッダとして伝送する. そして, 次の CLK4 から 1 クロック 1 ボクセル値の速度でボクセル値を転送する. VMU chip は, ヘッダを受信した時点で, 3 つの 2 進 $\log_2 n$ bit カウンタ xcount, ycount, zcount をそれぞれ 0, ystart, zstart に初期化する. 以後 1 クロック毎にカウンタの値を 1 ずつ増加させ, 各

カウンタの値が VMU ノード番号と等しくなった時のボクセル値を xlatch, ylatch, zlatch に格納する．同時に，xaddress, yaddress, zaddress にボクセル・アドレスを格納しておく．そして，パケット転送終了後に各ボクセル値を対応する memory bank に書き込む．

ボクセル・マスク・ビットの設定

ボクセル・マスク・ビットの設定により，対応するボクセルを一時的に透明にすることができる．これにより，ボリューム・データの一部を転送することなくマスクできる．ボクセル・マスク・ビットは各ボクセルにつき 2 ビットずつ存在し，結果として 1VMUnode につき $3\times n\times n\times 2 = 6n^2$ ビットの容量を持つ．ボクセル・マスク・ビットを 2 ビット持つことにより，一方のマスク・ビットを有効にして表示している間に，もう一方のマスク・ビットを設定することができること，2 つのマスク・ビットの間に AND, OR, XOR などの種々の演算を定義し，これらの演算を利用することによってボクセル・マスク・ビット間の複雑な演算を行うことができること，などの利点が得られる．

ボクセル・マスク・ビットの設定は，VMU chip が RDP の支援のもとで行う．RDP は，msk, mskdir, axis, $\vec{r_0}$, $\vec{\Delta r}$ を設定して，PVMcmd を mask 設定サイクルに指定する．mask 設定サイクルには，mask 設定サイクル 0 と mask 設定サイクル 1 があり，msk のビットの最下位 1 ビットを利用して指定される．mask 設定サイクル 0 では，VMU の mask 処理方向は，$\vec{r_0}$の第一要素の座標軸方向となり，mask 設定サイクル 1 では，第二要素の座標軸方向となる．

ボクセル・マスク・ビットの基本的な設定法は，mskdir=正方向，axis=x 軸として mask 設定サイクル 0 を実行した場合，$z = r_1$平面上の x memory bank のボクセル・マスク・ビットに対して書き込みを行うものである（図 25参照）．この時 VMU node$_i$は，x memory bank の

$(x = i), z = r_1, y \geq r_0 + i\Delta r_0$で表される範囲をマスクする．このマスクは，VMU node$_i$が，$x = i, z = r_1$を満たす y軸方向の 1 ラインを全て書き込むことにより行う．VMU は，ボクセル・マスク・ビットを 8 ビット並列に処理することができ，Voxel Maskbit Memory を 40ns でアクセスできる SRAM で構成する．この時，ボクセル・マスク・ビットの書き込みは，$512 \times \frac{512}{8} \times 3 = 96$k サイクル$= 3.84$ms かかることとなり，RDP の速度に余裕があれば，リアルタイム処理を行うことができる．

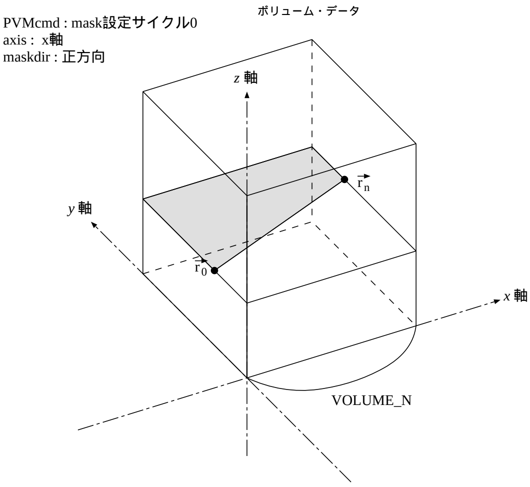


図 25: ボクセル・マスク・ビットの設定方法

表 6: ボクセル・マスク・ビット設定時の msk 指定

ビット位置	指定
0	0:設定サイクル 0/1:設定サイクル 1
1	ボクセル・マスク・ビットの書き込み先指定 (0/1)
2	ボクセル・マスク・ビット 0 の読み出し指定 (1:読み出し)
3	ボクセル・マスク・ビット 1 の読み出し指定 (1:読み出し)
4	VMU によるボクセル・マスク・ビットの指定 (1:使用)
5-6	2-4 ビットの無効なビットの指定 (00:2 01:3 02: 4 11: 全無効)
7-8	演算の指定 (00:OR 01:AND 10:XOR 11:NOT)

実際のボクセル・マスク・ビットの設定は，msk に表 6に示すパラメータを与えることにより，複雑な処理を行うことができる．この処理の基本は，VMU によって計算されたボクセル・マスク・ビット，ボクセル・マスク・ビット 0 の値，ボクセル・マスク・ビット 1 の値のうち，第 5-6 ビットで指定されるビットを除く 2 ビットに対して，第 7-8 ビットで指定される演算を行い，第 1 ビットで指定されるボクセル・マスク・ビットに結果を書き込む．ただし，NOT 演算の時には，第 5-6 ビットで指定されるビットに対して演算を

行う．また，第 5-6 ビットが全無効を示している場合，VMU によるボクセル・マスク・ビットの指定を直接書き込む．ボクセル・マスク・ビットの読み出しをともなう処理では，1 クロック目で読み出し，2 クロック目で書き込むため，ボクセル・マスク・ビットの設定処理は，先に計算した速度の 1/2 となる．

6 ボリューム・レンダリング向き並列計算機の評価

本章では，4章および5章で述べたボリューム・レンダリング向き並列計算機についての評価を行う．6.1節では汎用並列計算機 AP-1000 との速度比較を通じて，本計算機の処理能力を評価する．また，6.2節では，本計算機の生成する画像の質を評価し，補間アルゴリズムやピクセルあたりの視線の数の増加により画質がどの程度改善されるかを検討する．

6.1 汎用並列計算機 AP-1000 におけるボリューム・レンダリング

本節では，汎用並列計算機 AP-1000 上で行ったピクセル順アルゴリズムによるボリューム・レンダリングについての評価と，本研究において提案した専用計算機との速度比較を行う．

6.1.1 AP-1000 でのボリューム・レンダリング・アルゴリズム

本アルゴリズムは，ピクセル順アルゴリズムをピクセル並列で処理する．ここで，AP-1000 の構造が $nx \times ny$ プロセッサのトーラス結合であるため，画面を $nx \times ny$ に分割して各部分画面に含まれるピクセルの処理を対応するプロセッサに割り当てることとする．また，ボリューム・データの分割法は，n^3 個のボクセル値からなるボリュームを $\frac{n}{nx} \times \frac{n}{ny} \times n$ のサブ・ボリュームに分割し，隣接するサブ・ボリュームを隣接するプロセッサに割り当てることとした（図26参照）．

本アルゴリズムでは，以下の方針でボリューム・レンダリングを行う．

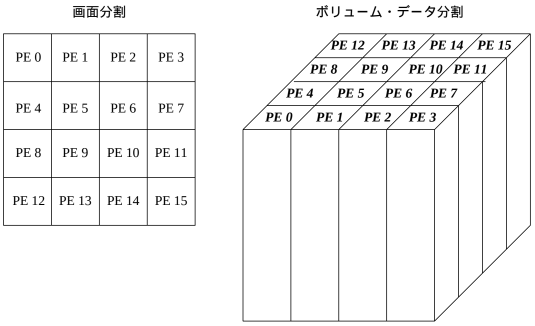


図 26: AP-1000 における画面分割および空間分割

1. 各プロセッサは担当する部分画面に含まれるピクセルを通る視線を生成する．
2. 1 で生成した視線のボリュームへの入射点を計算し，入射点を含むサブ・ボリュームを担当しているプロセッサに対して視線のパラメータを転送する．
3. 視線のパラメータを受けとったプロセッサは，担当するサブ・ボリューム内の視線上のボクセルを探索しながら，式 (1) に従ってピクセル値を計算していく．
4. サブ・ボリューム内のボクセルの探索を終了し，視線が別のプロセッサの担当するサブ・ボリュームに入射した時，そのプロセッサに対して，視線のパラメータおよび現在のピクセル値を転送する．ここで，本アルゴリズムのボリューム分割の方法により，視線のパラメータおよび現在のピクセル値の転送は，ほぼ隣接するプロセッサに対して行われることになる．
5. 3 の処理を視線がボリューム外に出るまで繰り返す．視線がボリュームの外に出た時，この視線を生成したプロセッサに対してピクセル値を転送する．
6. ピクセル値を受けたプロセッサは，部分画面にピクセル値を書き込む．

この方針により，各プロセッサ内の処理は，視線生成処理，視線探索処理，ピクセル値書き込み処理，の 3 種類に分割できる．各プロセッサは 1 視線を生成するたびに他のプロセッサからメッセージを受けているかどうかをチェックす

る．メッセージを受けている場合，メッセージの識別子をチェックして視線探索処理を行うか，ピクセル値を担当する部分画面に書き込むかを決定する．通信バッファの容量に制限があるため，各プロセッサは生成した視線の数と，視線探索が終了し担当する部分画面に書き込んだピクセル数の差が一定値（30）以上になると視線生成を行わずに視線探索およびピクセル値書き込み処理を行う．そして，全ての部分画面内の視線生成処理およびピクセル値書き込み処理が終了したとき，同期用のステータスを 0 にする．ボリューム・レンダリングの処理は，全プロセッサのステータスが 0 になったことをホスト計算機で検出した時に終了する．

6.1.2 AP-1000 におけるボリューム・レンダリングの速度評価

前節で述べたアルゴリズムを用いて，AP-1000 におけるボリューム・レンダリングの速度評価を行った．評価のパラメータは，表 7に示すものであり，ボリュームの中心座標は $(0,0,0)$ とし，ボリューム・データは各ボクセル値が同一のものを使用した．また，ボリュームは平行投影により，画面全体を覆うように投影した．

表 8に，ボリューム・レンダリングの実行時間を示す．この表は AP-1000 のパフォーマンス・アナライザを使用して計測したものであり，タスク処理時間は，視線生成処理，視線探索処理，および待ち状態でのアイドル・ループを含む．また，待ち時間はデータ受信チェックを行うライブラリ・コール時間の総計となっている．

また，表??に，トレースを解析して得られた各処理の実行時間等のパラメータを示す．ただし，t_{voxel}, t_{sover} は推測値である． 実行時間は，これらのパラメータを使って，以下の近似式で表される．

$$
\begin{aligned}
T_{all} &= T_{run} + T_{comm} + T_{wait} \\
T_{run} &= \frac{P}{C}(t_{ray} + t_{wr}) \\
&\quad + \frac{Pc_{ray}}{C}(\frac{v_{ray}}{c_{ray}} t_{voxel} \\
&\quad + t_{sover}) + \alpha_{idle} T_{wait} \\
T_{comm} &= \frac{P}{C}(t_{send} + t_{recv}) \\
&\quad + \frac{Pc_{ray}}{C}(t_{send} + t_{recv}) \\
\frac{T_{wait}}{t_{check}} &\geq \frac{P}{C} + \frac{Pc_{ray}}{C} \\
&\quad + \frac{(\frac{v_{ray}}{c_{ray}} t_{voxel} + t_{sover})c_{ray}}{t_{loop}} \\
&\quad + \frac{(t_{send} + t_{recv})(c_{ray} + 1) + t_{ray}}{t_{loop}}
\end{aligned}
$$

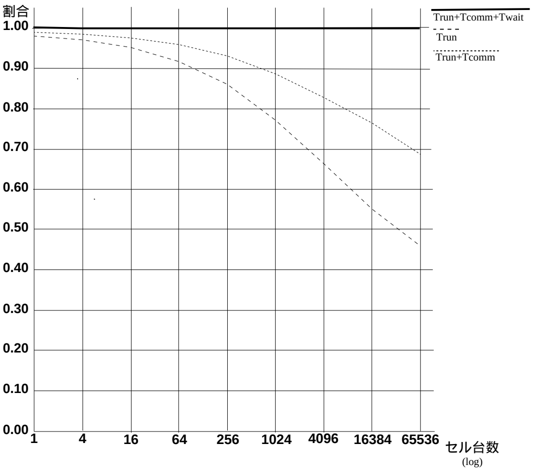


図 27: $T_{run}, T_{comm}, T_{wait}$ の比率

この式は，セル当りの視線生成処理の回数が $\frac{P}{C}$，セルを通過する視線数が $\frac{Pc_{ray}}{C}$ であることにより求められる．T_{run} の第 1 項はセルあたりの視線生成および書き込みにかかる時間，第 2 項はセルあたりの視線探索時間，第 3 項はデータ待ち時間である．T_{comm} の第 1 項は視線生成および書き込みにかかる通信時間，第 2 項は視線探索にかかる通信時間である．T_{wait} の中の第 1 項，第 2 項は視線生成時および視線探索時のループ回数であり，第 3 項は視線当りの全処理時間を空ループにかかる時間で割ることにより視線伝搬の遅延による空ループ回数を求めている．これらに t_{check} をかけることにより T_{wait} が求められる．また，v_{ray}, c_{ray} は視線と交差するボクセル数および視線が通過するセル数であり，各投影方向について表 9で示す値を用いた．

図 27に，512^3 ボクセルのボリュームを 512^2 のスクリーンに投影する実行時間に占める $T_{run}, T_{comm}, T_{wait}$ の比率を示す． この比率は近似式によって計算した推測値である．この図

表 7: AP-1000 におけるボリューム ・レンダリングの評価パラメータ

パラメータ	パラメータの値
セル台数	６４台
ボリュームの大きさ / ピクセルの大きさ	$64^3/64^2$, $64^3/128^2$, $128^3/64^2$
投影方向 (緯度, 経度)	$P_1(0, 0)$
	$P_2(0, \frac{1}{2}\pi)$
	$P_3(\frac{1}{6}\pi, 0)$
	$P_4(\frac{1}{3}\pi, 0)$
	$P_5(\frac{\pi}{2}, \frac{1}{6}\pi)$
	$P_6(\frac{\pi}{2}, \frac{1}{3}\pi)$
	$P_7(\frac{1}{4}\pi, \frac{1}{4}\pi)$

表 8: AP-1000 におけるボリューム ・レンダリングの処理時間 (μs)

ボリューム / ピクセル	投影	実行時間 T_{all}	タスク処理時間 T_{run}	通信時間 T_{comm}	待ち時間 T_{wait}
64^3 / 64^2	P_1	211264	140024(66.28%)	30565	40588
	P_2	116940	100453(85.90%)	6187	10215
	P_3	290104	166184(57.28%)	54149	88545
	P_4	292310	167039(57.14%)	35188	89993
	P_5	199410	131316(65.85%)	26445	41569
	P_6	166444	118197(71.01%)	18673	29478
	P_7	233702	140865(60.28%)	30047	62709
64^3 / 128^2	P_1	856534	566305(66.12%)	122298	167852
	P_2	480348	409951(85.34%)	24350	45969
	P_3	1035286	623064(60.18%)	146145	265997
	P_4	1033901	622739(60.23%)	146411	264664
	P_5	807799	531870(65.84%)	108401	167448
	P_6	684301	481420(70.35%)	76996	125800
	P_7	903987	553955(61.28%)	121512	228440
128^3 / 64^2	P_1	293949	221755(75.44%)	30070	42038
	P_2	201544	181267(89.94%)	6145	14044
	P_3	415556	259347(62.41%)	35620	120508
	P_4	421206	261383(62.06%)	35494	124242
	P_5	280771	207797(74.01%)	26164	46726
	P_6	249399	195186(78.26%)	18614	35509
	P_7	324766	217000(66.82%)	29918	77769

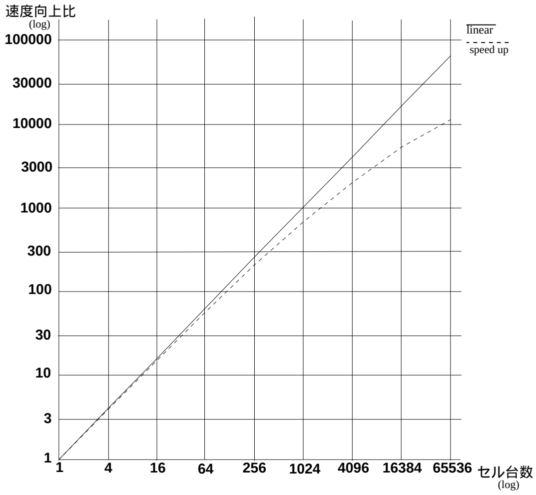


図 28: $512^3/512^2$における台数効果

より速度向上のためにセル台数を増やすと実行時間に占めるT_{run}の比率が減少し, 効率が落ちることがわかる. 1 セルでの実行時間の推測値に対する速度向上比を図 28に示す. セル台数が 65536 台の時の台数効果は 11401 倍, 全実行時間は 247813μs=0.248ms となっており, 台数の増加にともない効率が落ちることが確認できる.

6.2 生成画像の評価

本節では, 本稿で提案したボリューム・レンダリング専用計算機の生成する画像をシミュレーションした結果を示す. また, 画質向上のために提案した本計算機上での補間法が, どの程度有効であるかについて考察する.

6.2.1 評価項目と評価の指標

表 10に, 本研究において画像を比較評価した評価項目を示す.

サンプリング方法については, レイ・キャスティング・アルゴリズムにおいて一般的に使用されている視線方向等間隔サンプリングと, 本計算機で使用する主軸方向等間隔サンプリングを比較する. また, 主軸方向等間隔サンプリングについては, 4.4.1節で示した方法でボクセルの色および透明度を補正したものと, 補正しないものの両方を扱う.

計算精度については, 浮動小数点によって計算したものと, 本計算機で扱う固定小数点 (小数点以下 4bit) によって計算したものを比較する. ただし今回の実験では, 画像への影響が大きい各ボクセル座標についてのみ浮動小数点と固定小数点の双方を用いて計算し, ボクセルの色と透明度は浮動小数点でのみ計算した.

ボクセル値の補間については, 補間を行わないもの, 本稿で提案した 4 点補間を行うもの, および tri-linear 補間を行うものの 3 つを比較する.

ピクセルあたりの視線数については, ピクセルから 1 本の視線を発してピクセル値を求めるものと, ピクセルから 4 本の視線を発して, 各視線方向で計算した輝度を平均してピクセル値としたものとを比較する.

ボリューム・データはその大きさが, $64^3, 128^3, 256^3, 512^3$のものを用意し, これを 256^2のスクリーンに投影した. 対象となるボリューム・データは, 図 29に示す方法で, ベンゼンのϕ_1分子軌道を計算することにより生成した. また, ボクセル値の値域のの中から 3 つの小区間を取り出し, これらの区間に対してボクセル値の小さい方から青, 緑, 赤の色付けを行った. これらの区間以外の値を持つボクセルは透明であるとしたので, 3 つの等値面が重なった画像が生成される. そして, 評価要素の各組合せについて, ボリューム・データを上, 横, 斜めの 3 方向に投影した画像を生成した.

画像の評価は, 以下の式で定義される画像差を用いて行うとともに, 視察による評価も行った.

$$\text{画像差}\,(P_0, P_1) \;=\; \frac{\displaystyle\sum_{(x,y)\in\,\text{スクリーン}} |P_0(x,y) - P_1(x,y)|}{\displaystyle\sum_{(x,y)\in\,\text{スクリーン}} (P_0(x,y)+P_1(x,y))}$$

ただし, P_0, P_1は, それぞれボリューム・レンダリングによって生成した画像であり, $P_0(x,y), P_1(x,y)$ は, P_0および P_1のピクセル座標 (x,y) でのピクセル色成分[3]の合計値である.

---

[3] 画像は RGB 各 256 階調で生成した.

表 9: 投影方向とV_{ray}, C_{ray}の関係

投影方向	V_{ray}	C_{ray}
P_1	n	$\sqrt{n}$
P_2	n	1
P_3	$\frac{n}{\cos\frac{\pi}{6}}$	$\sqrt{n}(1+\tan\frac{\pi}{6})$
P_4	$\frac{n}{\cos\frac{\pi}{6}}$	$\sqrt{n}(1+\tan\frac{\pi}{6})$
P_5	$\frac{n}{\cos\frac{\pi}{6}}$	$\sqrt{n}$
P_6	$\frac{n}{\cos\frac{\pi}{6}}$	$\sqrt{n}\tan\frac{\pi}{3}$
P_7	$\sqrt{3}n$	$2\sqrt{n}\tan\frac{\pi}{4}$

表 10: 生成画像の評価基準

評価対象	選択肢
サンプリング方法	視線方向等間隔サンプリング
	主軸方向等間隔サンプリング（補正なし）
	主軸方向等間隔サンプリング（補正あり）
計算精度	浮動小数点
	固定小数点（小数点以下4bit）
ボクセル値の補間	補間しない
	4点補間
	tri-linear 補間
ピクセルあたりの視線数	1視線／ピクセル
	4視線／ピクセル

1. 炭素原子$i(i=1,2,\cdots,6)$のπ電子軌道ψ_iは, 水素様原子の原子軌道により,

$$\psi_i = (\frac{1}{4\sqrt{2\pi}})6^{\frac{5}{2}}e^{(-\frac{6r}{2})}z$$

   と近似される. ただし, rは炭素原子iからの距離, zは空間内のz座標である.

2. エネルギー準位1の分子軌道関数ϕ_1は, HMO法ではψ_iを使って,

$$\phi_1 = \frac{1}{\sqrt{6}}(\psi_1+\psi_2+\psi_3+\psi_4+\psi_5+\psi_6)$$

   と表される.

3. ボリューム空間内の全ての点に対してϕ_1を計算して, 各点でのエネルギー準位を求め, この値の絶対値をボクセル値として与える.

図 29: ベンゼンのπ電子軌道のボリューム・データの生成法

表 11: サンプリング方法による画像の差

ボリューム一辺	サンプル方法	視点の位置		
		横	上	斜め
64ボクセル	視線等間隔	0.414	0.260	0.275
	主軸等間隔（補正なし）	0.271	0.258	0.284
	主軸等間隔（補正あり）	0.271	0.258	0.287
128ボクセル	視線等間隔	0.366	0.156	0.160
	主軸等間隔（補正なし）	0.161	0.171	0.170
	主軸等間隔（補正あり）	0.161	0.171	0.166
256ボクセル	視線等間隔	0.361	0.085	0.084
	主軸等間隔（補正なし）	0.090	0.085	0.102
	主軸等間隔（補正あり）	0.090	0.085	0.089
512ボクセル	視線等間隔	0.351	0.047	0.045
	主軸等間隔（補正なし）	0.051	0.047	0.069
	主軸等間隔（補正あり）	0.051	0.047	0.048

6.2.2 生成画像の評価結果

以下に, 各評価項目について比較を行なった結果を示す.

サンプリング方法による画像の差

表11にサンプル方法による画像差の値を示す.

各画像差の値は, ボリュームの一辺が512ボクセルのデータを, tri-linear補間をともなう視線等間隔サンプリング法で投影した画像（以後見本画像と呼ぶ）との差を計算している. この表より, 視線等間隔サンプリングと主軸等間隔サンプリングの差は非常に少なく, ボリュームが大きい時には補正が有効であることがわかる.

ボクセル値補間法による画像の差

表12にボクセル値補間による画像差の値を示す.

各画像差の値は, 見本画像との差である. この表より, 本稿で提案している4点補間法は, tri-linear補間法と同程度の効果があることがわかる. 補間を行うものと行わないものの差は, 1つのボクセルが複数ピクセルに投影される場合に顕著であり, 一見してその効果がわかる.

ピクセルあたりの視線数による画像の差

表 13に, ピクセルあたりの視線数による画像差の値を示す. 各画像差の値は, 補正を行う主軸等間隔サンプリングと見本画像との画像差である. この表により, ピクセルあたりの視線数を増やすと, ボリュームが細かく, 1 ピクセルに多くのボクセルが投影される場合に有効であることがわかる.

計算精度による画像の差

表 14に, 計算精度による画像差の値を示す. 各画像差の値は, 主軸等間隔サンプリングと見本画像との画像差である. この表により, 小数点以下 4 ビットの固定小数点を使用しても, 画像の精度はほとんど変わらないこと, 固定小数点で 4 点補間を行っても, 画像の精度は向上することがわかる.

6.2.3 本計算機における画質の考察

前節における評価結果により, 本計算機によって採用した主軸等間隔サンプリング ·アルゴリズムは, 視線等間隔サンプリング ·アルゴリズムとほとんどかわらないことがわかった. また, 4.4.1節で示したボクセル色および透明度の補正は, 視線が多くの透明でないボクセルを横切る場合に有効であると考えられる.

本稿で提案したボクセル値の 4 点補間法は, tri-linear 補間法と同程度の効果があり, 特にボクセルが大きく投影される場合に有効であることがわかった. このため, 本計算機で扱うボリューム ·データの大きさを 256^3ボクセルとし, 4 点補間法でボリュームをリアルタイム表示しても, 512^3ボクセルのボリューム ·データの補間なしの表示より有効であると考えられる. また, 1 ピクセルから複数の視線を生成する方法は, ボクセルが小さく投影される場合に有効であることがわかった. この方法は, RDP と SP のプログラムを変更することにより, 本計算機に実装できる.

表 12: ボクセル値補間法による画像の差

ボリューム一辺	補間法	視点の位置		
		横	上	斜め
64 ボクセル	主軸等間隔 (補間なし)	0.271	0.258	0.287
	視線等間隔 (補間なし)	0.414	0.260	0.275
	主軸等間隔 (4 点補間)	0.368	0.222	0.273
	視線等間隔 (tri-linear 補間)	0.174	0.213	0.187
128 ボクセル	主軸等間隔 (補間なし)	0.161	0.171	0.166
	視線等間隔 (補間なし)	0.367	0.156	0.160
	主軸等間隔 (4 点補間)	0.088	0.109	0.140
	視線等間隔 (tri-linear 補間)	0.075	0.095	0.089
256 ボクセル	主軸等間隔 (補間なし)	0.090	0.085	0.089
	視線等間隔 (補間なし)	0.361	0.085	0.084
	主軸等間隔 (4 点補間)	0.354	0.060	0.068
	視線等間隔 (tri-linear 補間)	0.032	0.045	0.040
512 ボクセル	主軸等間隔 (補間なし)	0.051	0.047	0.048
	視線等間隔 (補間なし)	0.351	0.047	0.045
	主軸等間隔 (4 点補間)	0.358	0.033	0.037
	視線等間隔 (tri-linear 補間)	0.000	0.000	0.000

表 13: ピクセルあたりの視線数による画像の差

ボリューム一辺	視線/ピクセル	視点の位置		
		横	上	斜め
64 ボクセル	1 視線/ピクセル	0.271	0.257	0.287
	4 視線/ピクセル	0.259	0.249	0.271
128 ボクセル	1 視線/ピクセル	0.161	0.170	0.166
	4 視線/ピクセル	0.140	0.136	0.141
256 ボクセル	1 視線/ピクセル	0.090	0.085	0.089
	4 視線/ピクセル	0.075	0.069	0.069
512 ボクセル	1 視線/ピクセル	0.051	0.047	0.048
	4 視線/ピクセル	0.050	0.042	0.044

表 14: 計算精度による画像の差

ボリューム一辺	計算精度	視点の位置		
		横	上	斜め
64 ボクセル	浮動小数点	0.271	0.258	0.287
	固定小数点 (補間なし)	0.282	0.262	0.298
	固定小数点 (補間あり)	0.207	0.223	0.268
128 ボクセル	浮動小数点	0.161	0.171	0.166
	固定小数点 (補間なし)	0.179	0.160	0.172
	固定小数点 (補間あり)	0.105	0.114	0.137
256 ボクセル	浮動小数点	0.090	0.085	0.089
	固定小数点 (補間なし)	0.095	0.090	0.092
	固定小数点 (補間あり)	0.048	0.065	0.068
512 ボクセル	浮動小数点	0.051	0.047	0.048
	固定小数点 (補間なし)	0.056	0.048	0.050
	固定小数点 (補間あり)	0.029	0.034	0.034

7　おわりに

科学技術計算によって得られるデータを, ボリューム・レンダリング法により可視化する専用並列計算機のアーキテクチャを提案した. 本計算機は, 科学技術計算の可視化において不可欠である半透明なボリューム表示, 遠近法による表示, リアルタイム表示の３つの能力を持つ. 本計算機は, これらの能力を実現するために, 任意の視線と交差する全ボクセル値を同時に取り出すために, 主軸等間隔サンプリング・アルゴリズムとボリューム・データを３重に格納する３次元メモリ構造, および全ボクセル値からピクセル値をパイプライン式に計算することができるピクセル値演算機構を持つ.

そして, この３次元メモリ上で, tri-linear 補間法を簡略化した, ４点補間法を実装する方法を提案した.

また, 本計算機で使用する主軸等間隔サンプリング・アルゴリズムを AP-1000 上に実装し, 視線等間隔サンプリング・アルゴリズムで生成した画像との比較を行った. その結果, 主軸等間隔サンプリングは, 簡単な色および透明度の補正によって, 視線等間隔サンプリングとほとんど変わらない画像を生成できること, ４点補間法により, tri-linear 補間法と同程度の画像が得られることが明らかになった.

今後の課題は, 複数ボリュームの投影, 他の３次元図形処理アルゴリズムとの融合, リアルタイム表示システムの作成が挙げられる. 特に他の３次元図形処理アルゴリズムとの融合は, 本計算機の応用範囲をより広げるために重要な研究課題である.

謝辞

本研究の機会を与えて下さり, 数々の有益な御助言を与えて下さいました本教室教授富田眞治先生に深甚の謝意を表します.

御指導を賜り, また, 数々の有益な御助言を通して筆者を導いていただきました, 本教室助教授中島浩先生, 本教室助手森眞一郎先生に心から感謝致します.

本研究を進めるにあたって, 数多くの議論を通して本研究を支えて下さった, 共同研究者の北須賀輝明氏, 薄田昌宏氏に感謝致します.

筆者と労苦を共にし, 多くの貴重な御意見を与えて下さった, 京都大学工学部情報工学科富田研究室の諸氏に感謝致します.

設計支援システム「パルテノン」を御提供頂いた 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ァ研究グループの諸氏に感謝致します.

なお, 本研究では富士通研究所（株）の並列計算機 AP1000 を利用させていただきました.

参考文献

[1] R. Bakalash, A. Kaufman and Z. Xu: Building a Full Scale VLSI-Based Volume Visualization System, Advances in Computer Graphics Hardware V, pp. 109-115, 1992.

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

[3] 藤代 一成, 茅 暁陽: ボクセル集合モデルと可視化の枠組み, PIXEL, No. 119, pp. 127-132, 1992.

[4] Michael P. Garrity: Raytracing Irregular Volume Data, Computer Graphics, Vol. 24, No. 5, pp. 35-40, 1990.

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

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

[7] A. Kaufman: Introduction to Volume Visuualization, in Volume Visualization, A. Kaufman Ed., IEEE Computer Science Press, Ch. 1,1991.
(邦訳)藤代 一成: ボリュームビジュアライゼーション概論, PIXEL, No. 116, pp. 146-157, 1992.

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

[9] A. Kaufman: The voxblt Engine: A Voxel Frame Buffer Processor, Advances in Computer Graphics Hardware III, pp. 85-102, 1991.

[10] 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.

[11] 北須賀 輝明: ボリューム・レンダリング向き専用並列計算機のための高速視線生成機構, 京都大学特別研究報告書, 1993-2.

[12] K. Koyamada and T. Nishio: Volume visualization of 3D finite element method results, IBM Journal of Research and Development, Vol. 35, No.1/2, pp. 12-25, 1991.

[13] D.H. Lawrie and C.R. Vora: The prime memory system for array access, IEEE Transactions on Computers, Vol. C-24, 12, pp. 1145-1155, Dec. 1975.

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

[15] K.L. Novins, F.X. Sillion and D.P. Greenberg: An Efficient Method for Volume Rendering using Perspective Projection, Computer Graphics, Vol. 24, No. 5, 1990.

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

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

[18] G. Sakas and M. Gerth: Sampling and Anti-Aliasing of Discrete 3-D Volume Density Textures, EUROGRAPHICS '91, pp. 87-102, 1991.

[19] Texas Instruments: TMS320C4x User's Guide, 1991.

[20] H.K. Tuy and L.T. Tuy: Direct 2-D Display of 3-D Objects, IEEE Computer Graphics & Apprications, Vol. 4, No. 10, pp. 29-33, 1984.

[21] 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.

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

[23] L. Westover: Footprint Evaluation for Volume Rendering, Computer Graphics, Vol. 24, No. 4, pp. 367-376, 1990.