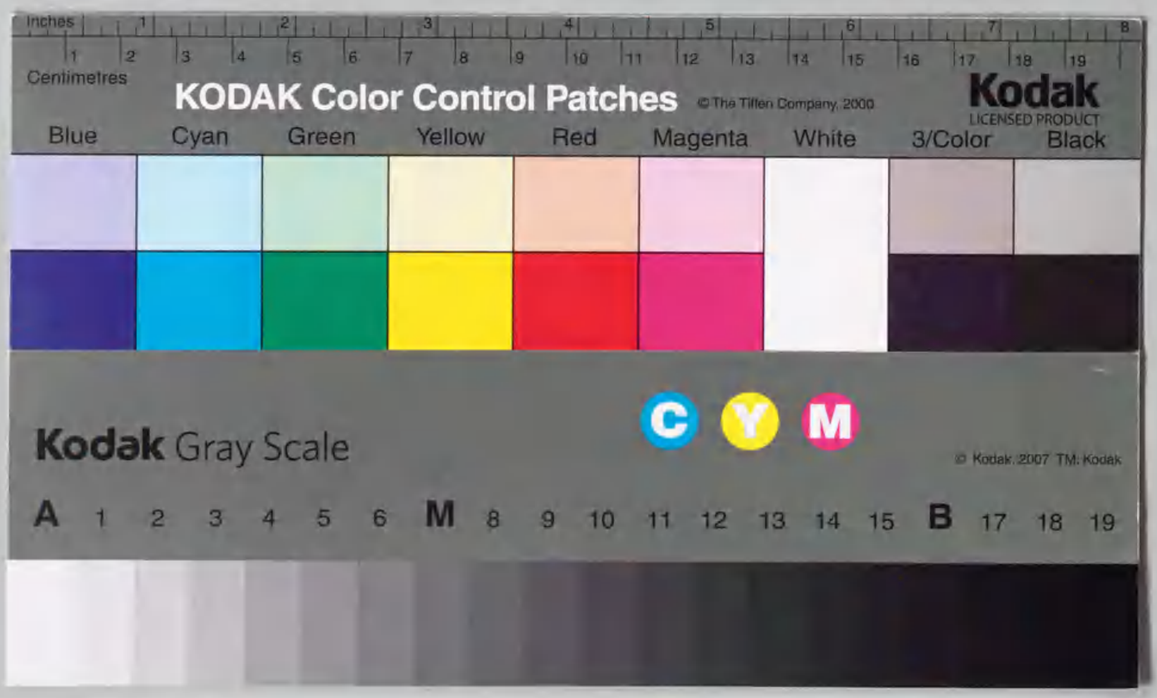
画像生成手法の高速化に関する研究

王 振威



画像生成手法の高速化に関する研究

成　瀬　　正

目　次

第１章　序論

１．１　研究の目的

情報の伝達手段としての画像メディアは、それが持つ情報量の豊かさから人間にとって極めて便利な伝達手段である。「百聞は一見にしかず」の諺はこれを如実に物語っている。従来より画像の提供手段として、カメラなど光学的撮像装置が一般的に使用され、ＴＶを代表とする画像情報伝達手段の発展を促してきた。現代の文明は画像無しには語れないほど画像の果たす役割は大きい。このような画像全盛の時代において、光学的撮像装置は現実に存在する環境に対して威力を発揮する。しかしながら、非現実世界の視覚化には光学的撮像装置は無力といわざるをえない。一方、計算機による画像生成は、より豊かなマンマシンインタフェースにおける画像情報の提供手段として登場した。それは、非現実世界の視覚化をも可能とする新しい画像情報の提供手段であり、コンピュータアートに代表される芸術への応用からシミュレーション結果の可視化などに代表される工学への応用にいたるまで幅広く応用されている。そして、計算機による画像生成の研究に対してコンピュータグラフィックスという新しい学問分野が開拓された。

コンピュータグラフィックスは、計算機による画像の生成を目的とした学問分野として、１９５０年代に誕生した。爾来、コンピュータ性能の向上とともにコンピュータグラフィックスの研究も成長を続けてきた。ワイヤフレームと呼ばれる線画から、面画へ、そしてソリッドモデルと呼ばれる３次元画像へと、対象の複雑さを増してきた。１９８０年はコンピュータグラフィックスにとって画期的な年となった。この年、T.Whitted[Whitted,1980]は光線追跡法（ray tracing）を発表し、その生成する画像の醸し出す写実性に世の中は驚嘆したものである。光線追跡法は、幾何光学モデルをベースとした光学系のシミュレーション手法であり、拡散反射光はもとより、鏡面反射光、屈折光をも対象とした汎用的な画像生成手法である。これを契機として、光線追跡法に関する研究が活発に行われるようになった。それらは、高速化手法と表現能力の向上の研究に大別される。

光線追跡法の一番の問題点は、計算時間である。画像を構成する各画素に対して光線の追跡計算、具体的には物体と光線の交差判定計算を行うために非常に多くの計算時

間を要す。この問題を克服せずして、光線追跡法が実用に供される道はない。そのため、高速化を目指して、アルゴリズム、計算機アーキテクチャ双方から種々の研究が行われてきた。アルゴリズムの観点からは、Bounding Volumeの利用、データの階層化、光束の追跡などが挙げられる。一方、計算機アーキテクチャの観点からは、並列処理アーキテクチャに関して多くの研究がなされてきた。

本論文の主題は光線追跡法の高速処理の実現にある。本論文では、「アルゴリズムと計算機アーキテクチャが互いの長所を引き出す一体となったシステムとなっていなければ、真の高速化は達成しえない」という基本思想の下で、研究の展開を図る。すなわち、本論文では、実時間光線追跡処理を究極の目標として、光線追跡法の高速処理をアルゴリズムと計算機アーキテクチャの双方から検討する。まず、光線追跡法の高速実行アルゴリズムについて検討する。その基本思想は、「画像を構成する各画素に投影される物体を効率よく求めることが光線追跡法高速化の本質である」ということである。そのために、動的部分木を用いた光線追跡法を提案する。このアルゴリズムはArvoのアルゴリズム[Arvo, 1987]より高速であることが示される。この検討から導かれる一つの結論は、「光線追跡アルゴリズムの主要な演算は3次元ベクトル演算であり、その並列実行が一層の高速処理を可能とする」ということである。そこで本論文では、その並列演算機構を内蔵した計算機アーキテクチャＳＩＧＨＴを提案する。試作したＳＩＧＨＴの性能評価結果を示す。さらに、その並列演算機構の有効性を明確にするため、3次元ベクトル演算をベースとする式の集合を考え、その集合に属す式がどの程度並列に実行できるかを集合論的観点から明らかにする。

１．２　研究の内容

本論文では、光線追跡法の弱点である計算時間の問題の克服を目指して、光線追跡法の高速処理手法を明らかにするため、以下の検討をする。

（１）動的部分木を用いた光線追跡法
（２）光線追跡法指向計算機ＳＩＧＨＴ
（３）３次元ベクトル演算の並列実行に関する理論的基礎付け

図１－１にこれらの研究の関連を示すとともに、以下の小節でこれらの概要を述べる。

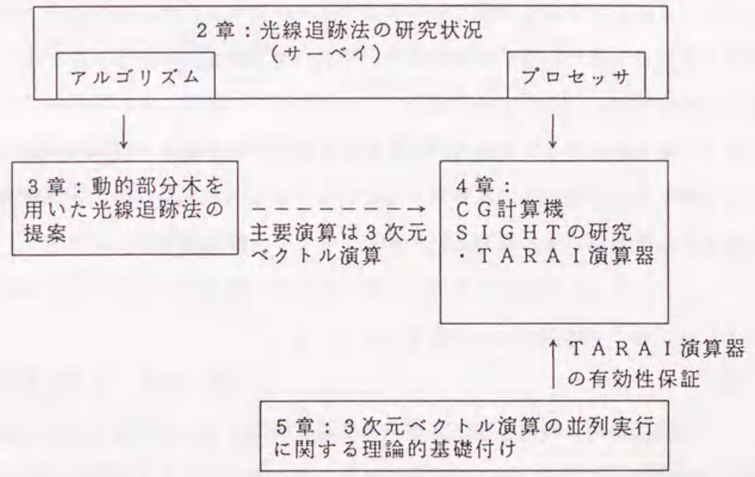


図１－１　本論文の構成と章間の関係

１．２．１　動的部分木を用いた光線追跡法

光線追跡法の高速化の本質は、各画素に対して、その画素に投影される物体を効率よく求めることにある。その方法として本論文第３章では、階層化されたバウンディングボリューム（ＢＶ）の木構造データを前提とし、それから作られる部分木を用いた手法を提案する。すなわち、画像の小領域に投影される物体のＢＶをノードとする部分木を、与えられた木構造データから動的に構成し、小領域内の画像生成は部分木を用いて行う手法である。ここで述べる手法ではさらに、①部分木に対して木の各ノードレベルで視点からＢＶまでの距離によるソーティングを行う、②小領域内の一様性テストを行う、という２つの高速化技法を併用することで、より一層の高速処理を実現する。

このアルゴリズムをインプリメントして実験を行う。部分木を用いることにより、交差判定回数を著しく削減でき、部分木を動的に作成するコストも小さいことを示す。また、従来高速であるといわれているArvoら[Arvo,1987]の手法をインプリメントして実験的比較を行い、本手法がより高速であることを示す。さらに、両アルゴリズムを解析し、計算コストの評価を行い、解析的にも本アルゴリズムが高速であることを示し、アルゴリズムの有効性を示す。

このアルゴリズムの計算時間を要する部分は、部分木の作成と光線追跡計算である。そこににおける主要な計算は、平面と物体あるいは光線と物体の交差判定計算であり、それはまた３次元ベクトル演算として記述できることを指摘する。

１．２．２　光線追跡法指向計算機ＳＩＧＨＴ

光線追跡法は幾何光学に基づいて光線を追跡する手法であり、その追跡は画素毎に独立に計算できるという特徴を有する。この特徴を利用して、マルチプロセッサ構成により画素毎の並列処理を実現できることが知られている。

一方、光線追跡法のアルゴリズムを詳しく調べると、その処理の大部分は物体と光線の交差判定計算に費やされることがわかる。それゆえ、交差判定の並列処理ができれば、結果として光線追跡法の一層の高速処理が可能となる。交差判定計算は、３次元空間のベクトル演算として特徴付けられる。したがって、その並列実行が鍵となる。本論文第

４章では、この並列実行を行うメカニズムを提案する。このメカニズムは３個の演算器と３個のメモリを簡単なネットワークで結合した構成であり、ＴＡＲＡＩと名付けている。

ここでは、さらに、ＴＡＲＡＩを中心に構成した要素プロセッサ（ＰＥ）を多数結合したマルチプロセッサシステムＳＩＧＨＴを提案する。ＳＩＧＨＴは、上記画素毎の並列処理と３次元ベクトル演算の並列処理の２レベル並列処理を実現した計算機である。

ＳＩＧＨＴのＰＥは、ＴＡＲＡＩを中心とする複数のユニットから構成される。これら複数のユニットを並行動作させる光線追跡アルゴリズムを示す。このアルゴリズムを用いてＳＩＧＨＴのＰＥの性能解析および評価を行い、アーキテクチャの有効性を示す。具体的には、シミュレータを用いて、ＴＡＲＡＩの演算並列度、ＰＥを構成するいくつかのユニットの並行動作を解析する。演算並列度の解析から、ＴＡＲＡＩ演算器の光線追跡計算への適用の有効性を明らかにする。また、典型的商用計算機ＶＡＸ１１／７８０との計算時間比較により、ＰＥの性能がＶＡＸの約１０倍であることを明らかにする。

１．２．３　３次元ベクトル演算の並列実行に関する理論的基礎付け

ＴＡＲＡＩの適用範囲、演算能力などを客観的に示すことが、ＳＩＧＨＴの本質を見極める上で重要である。本論文第５章では、集合論的観点からこの問題を考察する。

３次元定数ベクトルおよび３×３定数マトリクスからなる集合を考え、この集合上に基本演算を定義する。次に、この集合にベクトル変数を導入し、集合を拡張する。拡張した集合をＧと呼ぶ。Ｇは３次元ベクトル、３×３マトリクスから構成される式の集合である。本論文では、Ｇが持つ性質を論ずる。得られた主な結果は、

①Ｇが基本演算に関して閉じている。
②Ｇ上の微分演算を考えると、Ｇは微分演算に関して閉じている。
③Ｇ上の拡張演算を考える。拡張演算は、線形代数[たとえば佐竹,1971]で使用される主な演算である。これらの拡張演算は基本演算の組合せで記述できる。その結果、Ｇは拡張演算に関して閉じていることが自動的にわかる。

Ｇの作り方から、Ｇは３次元ベクトル、３×３マトリクスから構成されるほとんどの式を含む。したがって、Ｇ上の基本演算を効率よく並列実行する演算機構を導くこと

により、Gの式はこの演算機構で効率よく並列実行できる。本論文では、演算器を3個用いた場合の並列演算機構を示す。そして、Gの式をこの演算機構で実行した場合の演算並列度を論じる。3個の演算器を使用した場合、演算並列度は3が最大である。この演算機構により、Gの基本演算は演算並列度3で実行できる。また、拡張演算には、本質的に逐次的な演算が含まれるが、その部分を除けば、拡張演算も演算並列度3で実行できる。この演算機構は、ＴＡＲＡＩの演算機構と同一のものである。

本論文で示す手法は、集合の持つ性質を調べることにより、与えられた演算機構の能力を一般的かつ体系的に論ずる方法論の一つを与えるものである。

第２章　光線追跡法の並列処理と高速化の研究状況

２．１　まえがき

この章では、今日に至るまでの光線追跡法の並列処理および高速化技術に関する研究状況をまとめる。まず、２．２で光線追跡法の概要を述べる。ついで、２．３で光線追跡法の高速化手法、並列処理手法の研究状況をアルゴリズムの側面から概観する。２．４では、光線追跡法の高速処理を狙った並列処理プロセッサの研究状況を概観する。そして、２．５で本研究の位置づけを与える。

本論文で提案する高速化および並列処理技術は、①動的部分木を用いた光線追跡法、②光線追跡法の並列処理アーキテクチャ：ＳＩＧＨＴ、である。さらに、本論文ではＳＩＧＨＴアーキテクチャの基礎となる3次元ベクトル演算の並列実行法に関して理論的検討を加える。

この章の目的はこれらの課題を、これまでの研究の流れの中に位置づけることにある。

２．２　光線追跡法の基本概念

この節では光線追跡法の基本的考え方を述べる。尚、この節の記述にあたっては以下の文献を参考にした。

[Glassner,1989; 久保田,1972; 新谷,1990; 山本,1983; 横井,1986]

２．２．１　ピンホールカメラモデル

光線追跡法を一言でいうならピンホールカメラのシミュレーションというのが当を得ている。ピンホールカメラは図２－１に示すように、外界からピンホールを通して入ってきた光がフィルム面上に像を形成することを利用した装置である。ここで、ピンホールが像形成に本質的役割を果たす。すなわち、ピンホールにより、入射光の方向を限定することが本質的である。もし、ピンホールがなければフィルム面上の各点にはあらゆる方向からの光線が入射し、結果としてフィルム面は真白になるだろう。しかしながら、ピンホールが存在すれば、フィルム面上の各点に入射する光はその点とピンホールを結ぶ直線に沿った光のみに限定されるので、その入射光の強度が記録されフィルム面上に画像が形成されるのである。ここでシャープな画像を得るためには、ピンホールは点であることが要求される。なぜなら、ピンホールが面積を持つと、フィルム面上の点とピンホールを結ぶ円錐内の光が入射し、結果として画像がぼけることになるからである。

光線追跡法はこのモデルを拡張したモデルに基づいている。すなわちフィルム（あるいはスクリーン）をピンホールの前に置き、ピンホールを片目に置き換えたモデルである。このモデルを図２－２に示す。この場合、直接見ることのできる世界は目とスクリーンのコーナーを結んでできる四角錘の内部である。

２．２．２　光線の追跡

光線の追跡を行うことは、ピンホールカメラシミュレーションを実行すること、すなわち、光源から出て物体によって反射屈折して目に入ってくる光を追跡するシミュレーションを行うことである。追跡の方法に２通りある。

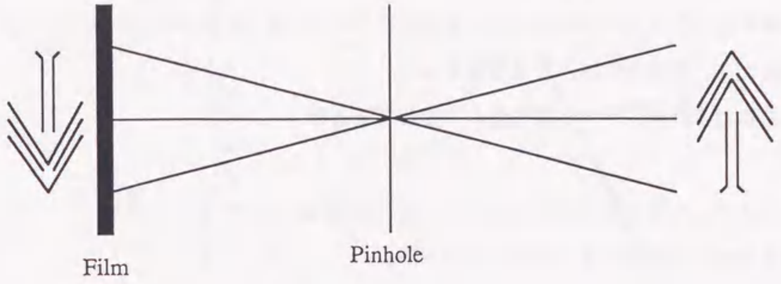


図２－１　ピンホールカメラモデル

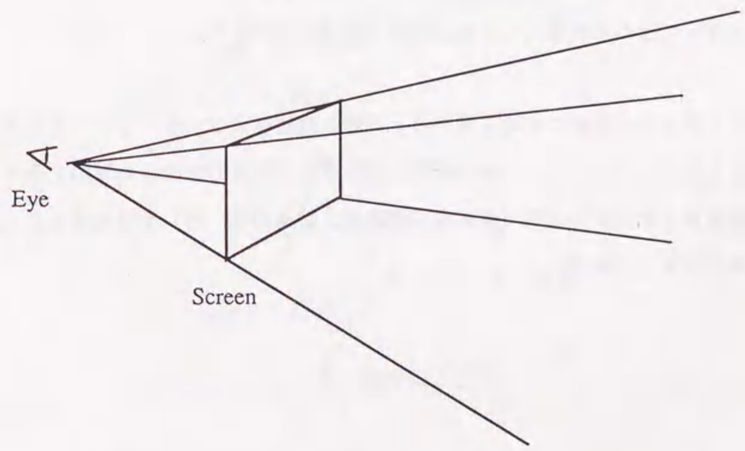


図２－２　修正したピンホールカメラモデル

・順方向に光線を追跡する方法：光源から目の方向に向かって光線を追跡する。

・逆方向に光線を追跡する方法：目から光源の方向に向かって光線を追跡する。

順方向追跡は光源からでるすべての光線を追跡する。そのため、目に入ってこない光線の追跡も行うことになり、そのシミュレーションは極めて計算量が多くなる。一方、逆方向追跡は目に入ってくる光線のみの追跡を行うので計算に無駄がない。このことから光線追跡法では、後者の追跡方法が採用される。

光線の追跡は以下の手順を踏んで行われる（図２－３参照）。

①目とスクリーン上の点（画素と呼ぶ）を結ぶ直線（光線と呼ぶ）を作る。

②光線と交差する物体があるか否かを調べる。

交差物体がない場合は、追跡を終了して輝度計算（２．２．５節）を行う。

交差物体がある場合は、交点の位置が目に最も近い物体を選択し、その表面属性により③の該当する手順を進める。

③表面属性により以下を行う。

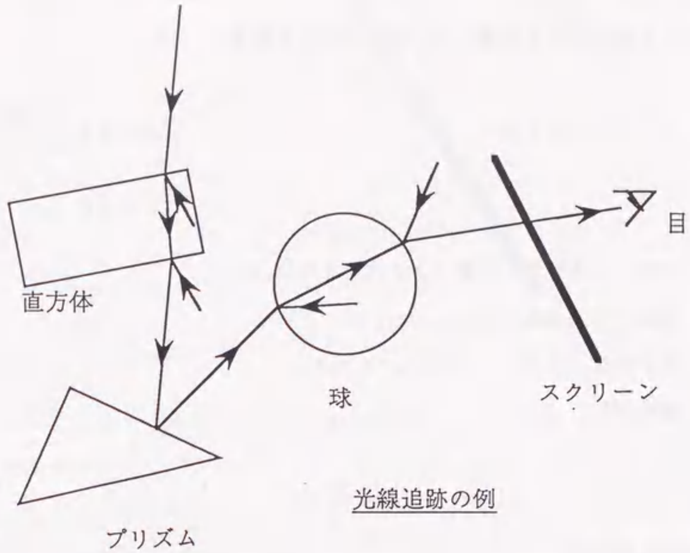
拡散反射面：追跡を終了して輝度計算を行う。

鏡面反射面：反射光の追跡を行う。反射光を新たな光線と考えて②の手順を進める。

透明体等の屈折面：反射光と透過光それぞれの追跡を行う。それぞれの光線を新たな光線と考えて②の手順を進める。

④スクリーン上の各画素について上記①～③の追跡を行う。

このように光線の追跡を進めると追跡の経路は図２－３に示すように木状になる。この木を光線追跡木と呼ぶ。光線追跡木の各ノードは交差した物体に対応するので、ノードに表面属性と関係する情報（反射率、透過率、表面の色、等）を持たせることにより、輝度計算が容易にできる。



光線追跡の例

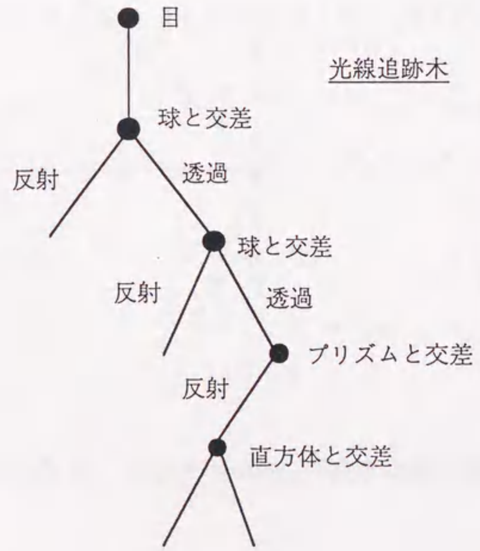


図２－３　光線追跡の例とその光線追跡木

2．2．3　光線、反射光、屈折光

（1）光線

光線の方程式は次式で与えられる（図2－4参照）。

$$r = a\,t + v \qquad (2-1)$$

ここで、

r：光線上の点を表す位置ベクトル，$r=(x,y,z)$

a：光線の方向余弦，$a=(a_x,a_y,a_z)$

v：目の位置ベクトル，$v=(v_x,v_y,v_z)$

t：媒介変数

である。

（2）反射光、屈折光

入射光に対する反射光、屈折光の関係は図2－5のようになる。図においてV、R、Pはそれぞれ入射光、反射光、屈折光の方向を表すベクトルである。また、Nは反射屈折面の法線ベクトルである。これらの方向を与えるベクトルR、Pは次式で与えられる。

$$R = V' + 2N \qquad (2-2)$$

$$P = k_1(N+V') - N \qquad (2-3)$$

ただし、

$$V' = \frac{V}{|(V\cdot N)|} \qquad (2-4)$$

$$k_1 = (n^2|V'|^2 - |V'+N|^2)^{-1/2} \qquad (2-5)$$

n：屈折率

この関係式により反射光、屈折光の方向余弦が与えられ、それぞれの光線の追跡ができる。

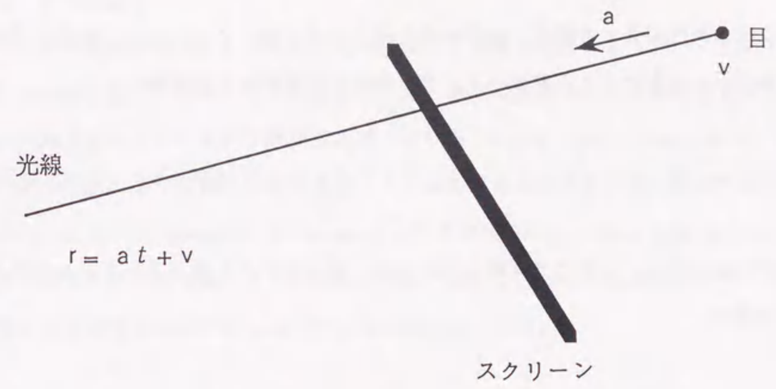


図2－4　光線の方程式

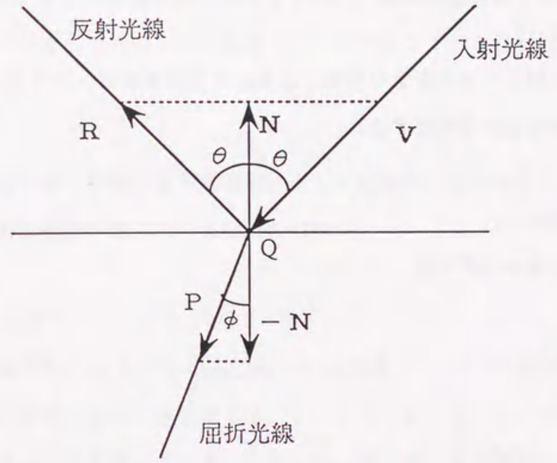


図2－5　入射光と反射光、屈折光の関係

2．2．4　交差判定

物体の形状を表す方程式と光線の式を連立させて解くことにより、物体と光線が交差するか否かを判定することができる。今、物体の形状がある陰関数

$$f(x, y, z) = 0 \qquad (2-6)$$

で表されているとする。このとき光線との交点は、式（2－1）を（2－6）に代入して得られる方程式

$$g(t) = 0 \qquad (2-7)$$

を解くことによって得られる。

具体的な交点計算の方法は物体の形状表現の方法に応じて異なってくる。代表的な形状に対する交点計算について以下に述べる。

（1）多面体

各面に対してそれを含む平面と直線との交点を求め、その交点が多面体の面をなす多角形の内部か否かを判定する。

（2）2次曲面

2次曲面は二次形式

$$r M {}^t r = 1 \qquad (2-8)$$

で表現できる。ここで、

r：　位置ベクトル　r＝(x,y,z)

${}^t r$：　rの転置ベクトル。

M：　係数マトリクス

である。この式と式（2－1）を連立させて解く。

（3）多項式曲面

多項式で表現される曲面の小片を接続して曲面を表現しようというもので、滑らかな自由曲面を表現するのに有効な方法である。多項式曲面との交点計算法に関しては、Kajiyaが双3次パッチに対する計算法を述べている[Kajiya,1982]。Hanrahanは、多項式曲面と光線の交点および交点における法線ベクトルを求めるために、記号処理を利用している[Hanrahan,1983]。SederbergはSteinerパッチと呼ばれるx,y,zの4次多項式で表現される曲面との交点計算法を導いている[Sederberg,1984]。Sweenyは自由形状のB－スプライン曲面との交点計算法について述べている[Sweeney,1986]。

（4）掃引体

掃引体は2次元図形を3次元空間内で掃引してできる軌跡の形状である。掃引体には平行移動掃引体と回転掃引体がある。

平行移動掃引体は、2次元図形を図形がのる平面に垂直な方向に掃引してできる形状である。この掃引体との交点計算法はKajiyaが論じている[Kajiya,1983]。

回転掃引体は、2次元図形を図形がのる平面内の直線を回転軸として回転させてできる形状である。この掃引体との交点計算法もKajiyaが論じている[Kajiya,1983]。

（5）メタボール

点電荷の集合が作る等電位面を曲面形状として定義するものである。この曲面との交点計算法に関しては西村が論じている[西村,1985a]。

（6）ＣＳＧ表現形状

ＣＳＧ（Constructive Solid Geometry）形状表現（あるいはＣＳＧモデルともいう）は、複雑な物体の表現に適した方法である。これは球、直方体、等の単純な形状（これをプリミティブと呼ぶ）を用意しておき、これに和、積、差の集合演算を適用して複雑な形状を表現する方法である。ＣＳＧモデルによる物体定義例を図2－6に示す。図において、たとえば、Bowlは半径r1の球の内側かつ半径r2の球の外側かつ直方体の内側として定義される。ここで、球は同心であり、直方体は同心球の下半分と交わるように配置する。図からわかるように物体の定義は、木構造をなす。この木をＣＳＧ木とよぶ。

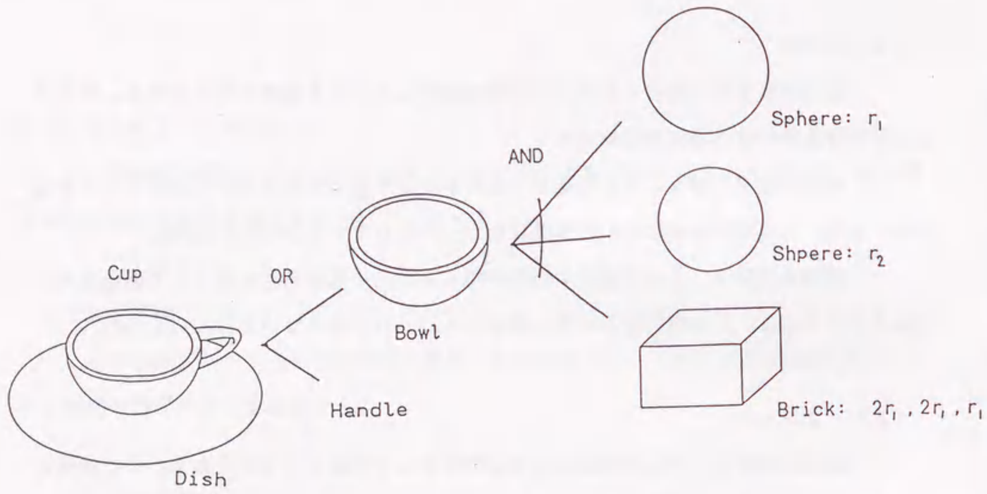


図２－６　ＣＳＧモデルによる物体定義例

ＣＳＧで表現された物体との交点計算法は、Rothが論じている[Roth,1982]。

２．２．５　輝度計算

２．２．２節で述べた手順に従って光線を追跡し、その追跡が終了すると光線が交差する物体表面の輝度を計算する。輝度計算のモデルとしてWhittedは次式を提案している[Whitted,1980]。

$$I = I_a + k_d \sum_{j=1}^{l} (N \cdot L_j) + k_s S + k_t T \qquad (2-9)$$

ここで、（図２－５参照）

Ｉ：物体表面の１点Ｑから出る光の強度

I_a：背景光による反射光強度

k_d：物体の拡散反射係数

ｌ：光源の個数

Ｎ：点Ｑにおける物体表面の単位法線ベクトル

L_j：第ｊ番目の光源の単位方向ベクトル

k_s：物体の鏡面反射係数

Ｓ：Ｒ方向（Ｖ方向に対する正反射方向）からの入射光強度

k_t：物体の透過係数

Ｔ：Ｐ方向（Ｖ方向に対する屈折方向）からの入射光強度

ただし、Ｖは視点と投影面上の点（画素）を結ぶ直線の方向であり、Ｐ，Ｒはそれぞれ、式（２－３），（２－２）で与えられるものである。

式（２－９）において右辺第１項は背景の一様な強度の光線の反射光を想定し、一定値を与えるものである。第２項は、光源から直接入射してくる光による拡散反射光強度をあらわす。第３項と第４項はそれぞれ鏡面反射光と透過光の強度をあらわす。

光線追跡木の各ノードに対し、葉から根の方向にボトムアップに式（２－９）を計算することにより、目に入ってくる光の強度（すなわち画素の値）が求められる。なお、Whittedは、式（２－９）においてk_s，k_tを一定値として扱っているのに対し、安田は光

学の理論に基づきそれらを可変にすることにより、より忠実な画像生成を行っている[安田, 1984]。

2．2．6　マッピング

マッピング手法は、物体表面の形状や属性を記述するパラメータ　－－　例えば、反射率、法線など　－－　の変化を物体表面の各点に対して与える手法である。代表的マッピング手法にテキスチャマッピング、バンプマッピングがある。

テキスチャマッピングは反射率（物体表面の光の吸収率）を物体表面の各点に与える。これにより、テーブルの木目などいろいろな模様をつけることができる。

バンプマッピングは物体表面の法線の変化を各点に与える。法線の変化に応じて陰影や反射屈折が変化し、形状のゆらぎが表現できる。代表的な応用として木肌の表現、水面の波などがある。

2．2．7　光線追跡モデルの拡張

Whittedのモデルは、光線追跡法が極めて写実的画像を生成できることを始めて示し、光線追跡法が脚光を浴びるきっかけとなったモデルである。これに対して種々の拡張が行われている。

（1）光の構成成分、分散効果を考慮したモデル

光を構成成分に分解して各成分の吸収を考える。これは内部吸収をもつ有色の透明物体の表示に有効である。安田は光を（R，G，B）三成分に分解しそれぞれの成分の吸収を考えるモデルを提案している[安田, 1985]。Hallは、光源、物体の諸特性（反射係数、透過係数、吸収係数等）のスペクトル特性を考慮したモデルを提案している[Hall, 1983]。Hallは光のスペクトルの分散効果は考慮していない。高木は、分散効果を考慮したモデルについて述べている[高木, 1984]。分散効果まで考慮するとレンズの色収差効果、宝石の色分散効果等が表現できる。

（2）光線の拡がりを考慮したモデル

光線を一本の直線として扱うのではなく、拡がりまで考慮する。ピンホールカメラモデルでいえば、ピンホールが面積を持つことに相当する。すなわち、目とピンホールを結ぶ円錐（コーン）を光線と考える。こうすることにより、ぼけや半影を表現できる。HeckbertはBeam-Tracingと呼ばれる手法を提案している[Heckbert, 1984]。これは、平面による反射を幾何学的鏡面対称変換に帰着させ、スクリーン面でのクリッピングと奥行きソートアルゴリズムを適用することにより高速な処理を実現したものである。しかしながら、この手法の適用できる対象物体は多面体に事実上限定され、また、屈折が扱えないなどの問題があるため、解決すべき課題が多い。Amanatidesは光線をコーンとしてとらえ、物体表面での反射、屈折に応じてコーンの中心角と拡がり角の変化を計算している[Amanatides, 1984]。Cookは反射光を1本だけ追跡するのではなく、反射光の拡がりに応じて複数本追跡するDistributed Ray Tracingを提案している[Cook, 1984]。この場合、追跡する反射光線の選択が問題となるが、Cookはモンテカルロ法のように乱数に基づきサンプリングする方法を提案している[Cook, 1986]。新谷は有限の拡がりをもった光束に拡張しその追跡法を論じている[Shinya, 1987]。

２．３　高速化アルゴリズムの研究状況

この節では光線追跡法を高速化する種々のアルゴリズムに関して研究状況を述べる。

２．３．１　高速化戦略の分類

光線追跡アルゴリズムの高速化に関しては、大きく分けて次の３つの戦略が考えられる[Glassner,1989]。

（１）交差判定の平均コストを下げる。
（２）追跡する光線の総数を減らす。
（３）光線を個々に追跡するのではなく、光線の束を追跡する。

図２－７にこれらを図示する。図に示すように、（１）はさらに交点計算自身を高速化する手法と交点計算回数を減らす手法に分類される。

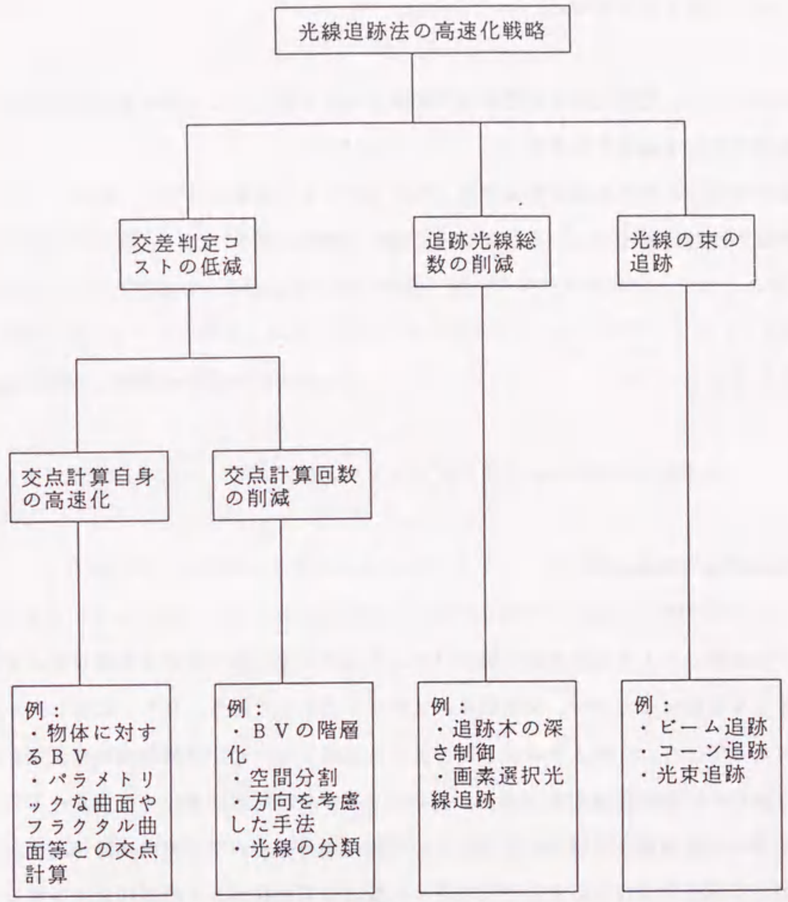
（１）に属する主な手法には、

（ａ）交点計算自身の高速計算
・Bounding Volume（ＢＶ）と呼ぶ、物体を包含する立体を用いる手法、
・パラメトリック曲面や代数的曲面等の特殊な曲面に対する効率のよい計算手法、
（ｂ）交点計算回数の削減
・ＣＳＧ形状表現においてＢＶを利用する手法、
・物体空間を分割して高速化を図る手法、

等がある。

（２）に属する主な手法には、

・適応的に光線追跡木を求める手法、



（注）ＢＶはBounding Volumeの略記

図２－７　光線追跡法高速化の戦略

・すべての画素に対して光線追跡を行うのでなく、適当に画素を間引いて残った画素に対して光線追跡を行い、間引いた画素は近傍の画素の値から補間する手法、

等がある。

（３）に属する手法には、

・多面体に対して、反射光や屈折光が線形変換で求められることを利用して、ビームと呼ぶ複数の光線を追跡する手法、
・光線をコーンとみなして追跡する手法、
・近軸光線の理論に基づいて、光線の束（光束）を追跡する手法、

等がある。

次節以下では、図２－７に示した例のうち主なものの内容をやや詳しく説明する。

２．３．２　交差判定の平均コストを削減するアルゴリズム

（１）Bounding Volume（ＢＶ）

交点計算コストの低い単純な形状（球、直方体、等）をＢＶとして用いる。物体を包含するＢＶを設定しておき、交差判定はまずＢＶに対して行う。ＢＶと交差したらさらに物体に対して行う。このようにすると、ＢＶと交差しない光線は物体との交差判定が不要となるから平均的な交差判定コストは低くなる。一般に物体数が多くなると一本の光線と交差する物体数の割合は相対的に小さくなる。すなわち、一本の光線に対し、それと交差しないＢＶの数が相対的に大きくなる。それ故、交差判定コストの平均値は下がる。

ＣＳＧモデルを用いて物体を記述した場合には、ＣＳＧ木の各ノードに、そのノードの子孫となる葉ノードのプリミティブをすべて包含するＢＶを設定する。この結果、ＢＶの階層構造ができる。この木に対する交差判定は次のようになる。①ルートノードに設定したＢＶと交差判定を行う。②交差しなければこの木に対する交差判定は終わり。交差すれば、子ノードに設定したＢＶと交差判定を行う。③すべての子ノードと交差しなければ交差判定終了。いくつかと交差すれば、それらのノードに対し、その子供との交差判定を行う。④このように、木の根から葉に向かって交差判定を進め、葉ノードのプリミティブと交差すれば、物体と交差したと判定する。

このような階層化を行うことにより、複雑な形状をした物体の交点計算は、単純な形状のＢＶの交点計算に帰着する。このような階層化により交差判定コストの削減ができる。

（２）Kayの方法

Kayは、ＢＶの体積をできるだけ小さくするという観点から、多面体で物体を包含する効率的な手法を提案している[Kay,1986]。彼の方法の基本的考えは、平行な平面で物体をはさむことであり、各物体をはさむ平面が互いに平行であれば、それらの平面に対する交差判定のコストが小さくなり高速処理ができるということである。また、KayはＢＶの階層化を同様の手法で行っている。

（３）空間の分割

空間分割法は空間の分割のしかたにより（ａ）非一様分割法と（ｂ）一様分割法の２通りに分けられる。非一様分割法で用いる代表的データ構造はOCTREEであり、一様分割法でのそれは３次元配列である。

（ａ）非一様分割法

空間の非一様分割法についてはGlassnerが論じている[Glassner,1984]。物体の存在する空間を８つの同じ大きさの部分空間に分割する。各部分空間に対し、その部分空間と交差する物体数がある閾値を越えると、その部分空間を８つの部分空間に等分割する。これを再帰的に行う。このように分割を行うと、OCTREE構造が得られる。OCTREEに構造化された物体と光線の交点計算は図２－８のアルゴリズムで実行される。図のプログラムはＣ風の言語で記述している。（以下プログラムは特に断わらないかぎりＣ風の言語で記述する。）

図２－８のプログラムにおいて、部分空間に隣接する部分空間を見いだす手順、すなわち、隣接部分空間を見いだすOCTREEのトラバース法、の詳細については[Sammet,19

84]を参照されたい。

（ｂ）一様分割法

空間の一様分割法に関しては藤本が論じている[Fujimoto,1986]。物体の置かれた３次元空間をVoxelとよぶ同じ大きさの部分空間（通常、立方体）に分割する。分割された３次元空間は３次元配列で表現できる。配列の要素はVoxelに対応する。配列要素には、対応するVoxelと交差する物体を指すポインタを格納する。そのような物体が複数個あるばあいには、それら物体のリストを指すようにしておく。

交点計算は、①光線と交差するVoxelを計算、②そのVoxelに属する物体との交差判定、という手順で進む。空間分割の一様性から、光線と交差するVoxelは、３次元ディジタル直線生成法（３ＤＤＤＡ）を用いれば高速に計算できる。図２－９にアルゴリズムの概略を示す。

この手法は、

①光線が貫くVoxelに属する物体との交点計算だけを行えばよいので、交点計算回数を著しく削減できる。

②３ＤＤＤＡによる高速直線発生法が利用できるので、Voxelの通過計算が高速化できる。

という利点を持つ反面、

①Voxelの辺の長さをrとすると、Voxel数は(1/r)の３乗に比例して増加するので、Voxelの体積を小さくすると必要メモリ量が急速に大きくなる。

という欠点がある。

２．３．３　追跡する光線総数を削減するアルゴリズム

（１）適応的光線追跡木

光線追跡木の探索の深さを適応的に制御する。この手法はHallが最初に導入した[Hall,1983]。これは、光線追跡木の探索を、あらかじめ定めた深さ、あるいは拡散反射物体と交差するまで追跡するのではなく、光線の強度がある程度以上小さくなればそこで追跡を打ち切るという手法である。これは、反射率の高い表面属性を持った複数の物体が互いに写り込んでいるような状況下で有効な手法である。


```
Octree_Intersection(ray)
struct ray_struct *ray;
{
  double Q;

  Q= ray.origin;
  do { /* walk through the subspaces */
    locate the subspace which contain Q;
    for(each object associated with subspace) {
      Intersection(ray,object);
      }
    if(no intersection has been found) {
      Q= a point in the next subspace pierced by ray;
      }
  } while(an intersection is found || Q is outside the space);
}
```

図２－８　ＯＣＴＲＥＥ分割法における交点計算手順

```
Voxel_Intersection( ray, node )
struct ray_struct  *ray;
struct node_struct *node;
{
  struct object_struct *object;

  compute i, j, k for the voxel containing ray->origin;
  set up 3DDDA based on ray->direction and ray->origin;

  do { /* Walk through the voxels */
    object= voxel[i][j][k];
    intersect(ray,object);
    if(intersection has been found) {
        node->object= object;
        node->position= pack(i,j,k);
        return;
        }
    compute new i,j,k using 3DDDA;
  } while (i,j,k is outside the limits of the voxel array);
}
```

図２－９　ＶＯＸＥＬ分割法における交点計算手順

（２）画素選択型光線追跡法

画像の小領域における画素間の輝度値の相関が高いことに着目して、正確に輝度を求める必要のある画素のみ光線追跡によりその値を求め、その他の画素は正確に求めた画素の輝度値から補間により求めて高速に画像を生成する手法である。これは、秋本と橋本が提案している[秋本,1986;橋本,1987]。

秋本の手法は、画面上を横方向、縦方向共にＤ画素間隔でサンプリングし、サンプル点に対して光線追跡を行う。正方形を構成する隣接４サンプル点から、その内部画素の平坦性を判定する。その結果、平坦であれば内部画素の値は４サンプル点の値から補間して求める。平坦でなければ正方形を４等分し、その各コーナーの画素の値を光線追跡して求める。そして、各小正方形領域に対して平坦性を判定する。このような処理を再帰的に行う。

平坦性の判定法として秋本は三つの基準を提案している。

①画素の輝度：４サンプル点の画素の値のばらつきが小さい。

②交差物体の一致：光線追跡木に交差物体名を記憶しておく。４サンプル点の交差物体が一致する。

③影を落としている物体の一致：光線追跡木に影を落としている物体名を記憶しておく。４サンプル点の影物体が一致する。

これらの基準を単独であるいは組み合わせて平坦性判定に使用する。

秋本は実験の結果から、この手法により画質を保ったまま従来の方法の２０％～６３％に画像生成時間を減少できると報告している。

一方、橋本は、秋本の考えをさらに進めて、境界画素選択型光線追跡法（ボーダーレイトレース、BorderRay Tracing）を提案している[橋本,1987]。ボーダーレイトレースは、物体の透視投影像を作り、それを参照して光線追跡が必要な画素と補間すればよい画素を選択する画素選択型の手法である。この手法では、透視投影像を用いることにより、物体の境界が容易にわかるので、光線追跡をすべき画素の選択が効率よくできる。透視投影像を作成する時間がオーバヘッドとなるが、橋本は、その時間は、全画素を光線追跡して求めた場合の１～３％と報告している。その結果、この手法では、従来の光線追跡法の約１０倍、秋本の手法の１．２～４倍の高速化が達成できる。

２．３．４　光線の分類による光線追跡

Arvo[Arvo,1987]は、光線と物体の交差判定回数を著しく削減するアルゴリズムを開発した。出発点と方向が類似する光線は、類似な振る舞いをするという考えのもとに、追跡の対象となるすべての光線を、類似した光線の集合Ｅ1，Ｅ2，・・・，Ｅmに分ける。また、物体の集合Ｃ1，Ｃ2，・・・，Ｃmを考える。ここで、Ｃiは、Ｅiに属す光線が交差しうる物体をすべて含む集合である。各iに対してＣiの要素数が少なければ、各光線が必要とする交差判定回数も少なくなる。光線は、出発点の位置（ｘ，ｙ，ｚ）と方向（θ，ϕ）を与えれば一意に決まる。したがって、光線は５次元空間の点として表すことができる。すなわち、Ｅiは５次元空間の部分集合である。Arvoのアルゴリズムは以下のようになる。

①考えるべきすべての光線の集合Ｅを作る。

②Ｅを互いに素な集合Ｅ1，Ｅ2，・・・，Ｅmに分割する。

③各Ｅiに対して、Ｅiに属す光線と交差する可能性のある物体からなる集合Ｃiを作る。

④Ｅに属す各光線に対して、それが属す部分集合Ｅiを見つけ、対応する物体集合Ｃiを見つける。

⑤与えられた光線とＣiから視点に最も近い物体を決定する。

Arvoは、画像の生成実験を行い、Kayの方法より４倍程度高速処理ができると報告している。

２．３．５　光線の束を追跡するアルゴリズム

光線の束を追跡する手法のうち、Beam-TracingとCone-Tracingは、標準的な光線追跡法の素直な拡張であり、処理の高速化やエイリアシングの問題の解決を実現している。一方、新谷の光束追跡法は、近軸光線の理論をベースにした注目すべき手法である。

（１）光束追跡法

光線の集合（光束）を追跡する手法として、新谷は近軸理論の適用を試みている[shinya,1987]。近軸理論は、レンズ設計で用いられる理論であり、基準となる光線（軸光線）の近傍にある光線（近軸光線）の振舞いを線形近似するものである。

軸光線に直交する平面を考える。そのとき、近軸光線はつぎのパラメータで表現できる。

①この平面と近軸光線の交点の座標　$(x1,x2)'$

②近軸光線のこの平面への射影ベクトル　$(\xi 1,\xi 2)'$

（これは近軸光線の方向を表わす。）

これらを用いて$\phi=(x1,x2,\xi 1,\xi 2)'$なる４次元ベクトルを定義し、近軸光線を表現する。

光線が光学系を通過するとき、その位置と方向（すなわちϕ）が変化する。この変化が小さい場合には、系を線形系とみなすことができる。すなわち、この変化は線形変換として記述できる。線形変換はマトリクスで記述できるから、光線の変化は次のように書ける。

$$\phi'=T\phi \qquad (2-10)$$

ここで、ϕは入射光線、ϕ'は出射光線をあらわす。Ｔは４×４マトリクスであり、光学系を記述している。Ｔはシステムマトリクスと呼ばれる。

光線の直進、反射、屈折に対してそれぞれのシステムマトリクスを求めておけば、光学系を通過する光線の振舞いはそれらのシステムマトリクスの積として記述される。したがって、軸光線を光線追跡して光学系の通過の振舞い――直進、反射、屈折――を調べ、対応するシステムマトリクスを求めておけば、近軸光線の振舞いはシステムマトリクスの積を計算することにより求めることができる。光線の追跡計算よりシステムマトリクスの乗算のほうが計算コストが安いので光線追跡の高速化が図れる。

２．４　並列処理プロセッサ研究状況

この節では、光線追跡法による画像生成を主たる目的として開発された種々の並列処理計算機について、その開発思想、アーキテクチャの特長、性能などを概観する。

２．４．１　並列処理プロセッサの分類

光線追跡法の高速実行の観点から並列処理プロセッサを見ると、そのアーキテクチャは、光線追跡法に内在する並列性を如何に反映しているかで分類することができる。その並列性には、

①画素間独立性：光線追跡法は、幾何光学の法則に従って光線を追跡する手法であるので、光線間の相互作用を考慮しない。したがって、追跡計算は画素毎に独立にできる。

②演算の並列性：光線の追跡計算は３次元空間のベクトル演算として特徴付られる。３次元ベクトル演算を並列実行する。

③演算と制御の並列処理：光線追跡アルゴリズムのフロー制御と交点計算等の演算を並列に実行する。

がある。

上記並列性のうち、①はマルチプロセッサ構成で実現する。すなわち、画素単位の処理をプロセッサに割り付ける。この並列処理は、マルチプロセッサ構成の計算機システムであればどのようなものでも基本的に実現できる。②と③は、プロセッサ内部のアーキテクチャを工夫して実現する。すなわち、②は３次元ベクトル演算装置を搭載して実現し、③は演算ユニットと制御ユニットを搭載し、これらを並列に動作させて実現する。これまでに、開発された画像生成計算機はこれらのうちの幾つかをインプリメントしている。それらを並列処理のレベルによって分類した結果を図２－１０に示す[吉田,1989]。図で、ＬＩＮＫＳは大阪大学で、ＣＡＰは富士通で、ＭＣは松下電器で、Pixel MachineはＡＴＴで、ＳＩＧＨＴとＭＡＧＩＣはＮＴＴでそれぞれ開発された。

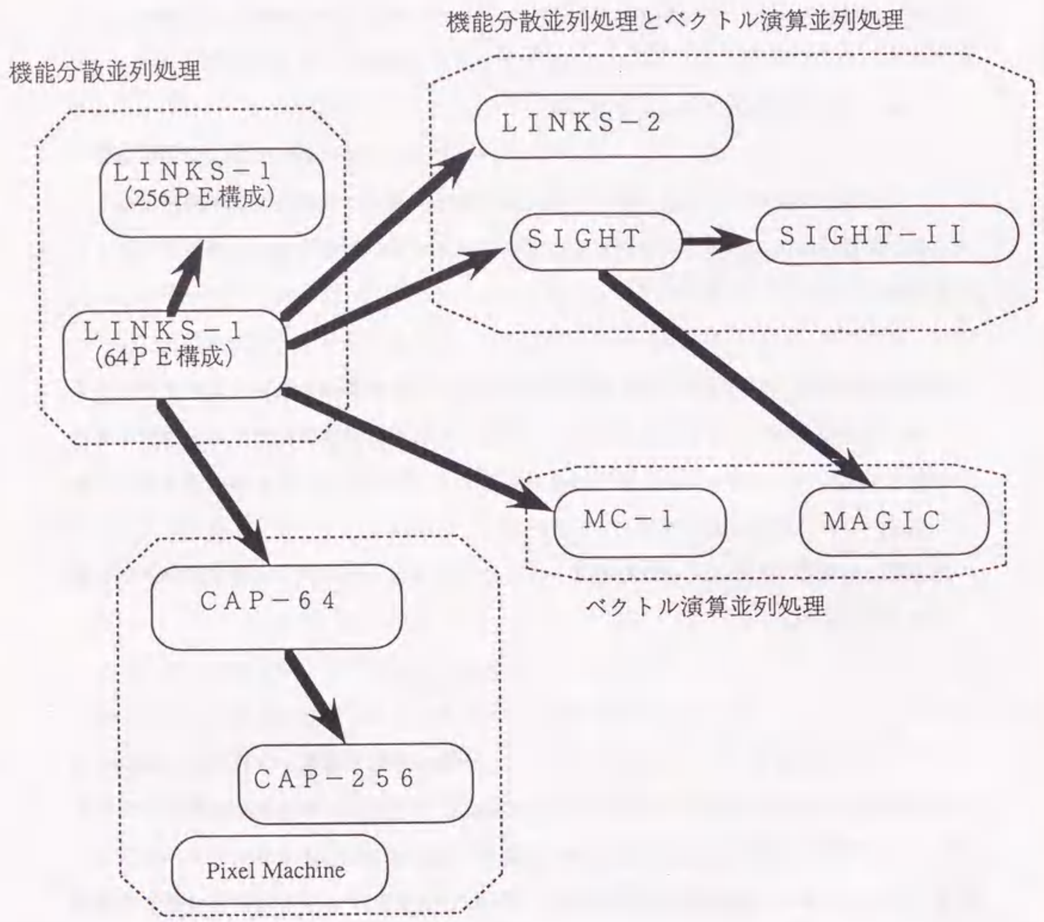


図２－１０　光線追跡法を指向したプロセッサの分類
(矢印の先のプロセッサは根本のプロセッサの影響を受けていることを示す。)

次節以下では、これらのプロセッサのうち、草分け的存在であるＬＩＮＫＳと、パイプライン制御に基づくPixel Machineについてその構成を示す。

２．４．２　ＬＩＮＫＳ－１

ＬＩＮＫＳ－１は大阪大学の大村助教授を中心とするグループで開発された、光線追跡法の高速実行を狙った最初の並列計算機である[Nishimura,1984]。ＬＩＮＫＳ－１では、画素間演算の独立性に着目し、マルチプロセッサによる並列処理を行う。図２－１１にＬＩＮＫＳ－１のシステム構成を示す。このシステムは以下の特徴をもつ。

①システムは１台のルートコンピュータ（ＲＣ）と複数台のノードコンピュータ（ＮＣ）からなり、それらは、疎結合星状接続されている。ＲＣとＮＣはＩＯ部分を除き同一の構成をとっている。
②メモリ交換ユニットによりＲＣとＮＣ間の通信およびデータ転送を行なう。これは、２組のメモリからなり、ダブルバッファリングを実現している。これにより、データ転送のオーバヘッドを軽減する。（当初、ＲＣとＮＣの通信にはシリアル回線が使われていたが、低速のため現在はメモリ交換ユニットを介した通信が行なわれている。）
③マッピングデータ等大規模データを格納するため、各コンピュータ（ＲＣ，ＮＣ）は、大容量メモリをもつ。

ＮＣは、図２－１２に示すように、制御ユニット、算術演算ユニット、メモリユニットの３ユニットからなる。制御ユニットは、ＣＰＵにＺ８００１を用いた構成で、ＲＣとの通信、算術演算ユニットの起動制御等を行なう。算術演算ユニットは、ＣＰＵにＩ８０８６、Ｉ８０８７を用いた構成であり、制御ユニットの制御下で、光線追跡計算を行なう。浮動小数点演算の高速実行を行なうために数値演算プロセッサ（Ｉ８０８７）が導入されている。システムバスにはメモリ交換ユニットが接続されている。制御ユニットは、システムバスを介してメモリ交換ユニット内のデータを算術演算ユニット内のメモリに転送する。そして、バススイッチを切り替えたあと算術演算ユニットを起動し、光線追跡計算を開始させる。計算の結果得られた輝度値はＢＤバスを介してデータコレクタに送られ、ディスプレイ上に表示される。

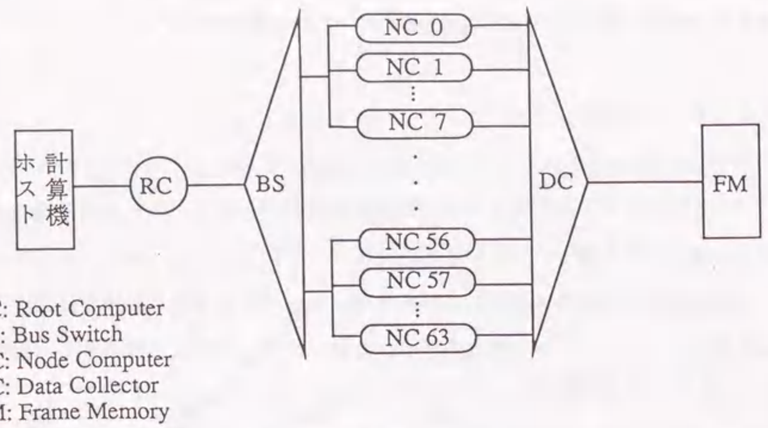


図２－１１　ＬＩＮＫＳ－１システム構成

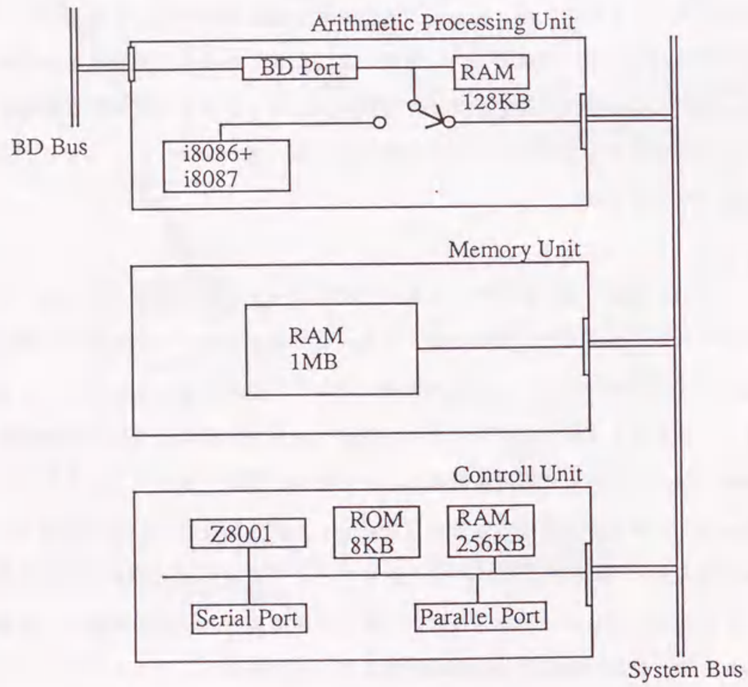


図２－１２　Ｎｏｄｅ　Ｃｏｍｐｕｔｅｒの構成

ＮＣでは、制御ユニットと算術演算ユニットによる機能分散処理が行なわれているが、それは、制御プロセッサによる算術演算ユニットへのデータ供給と算術演算プロセッサの演算実行という粗いレベルである。４章で述べるＳＩＧＨＴでは、光線追跡アルゴリズムのフローレベルできめ細かな機能分散処理を行なっている。

ＲＣは、図２－１２の構成に入出力ユニットを加えた構成である。ＲＣは、ホストプロセッサからノードプロセッサへのデータ転送、外部入出力機器とのＩＯを行なう。

２．４．３　Pixel Machine

ＡＴ＆Ｔ　Ｂｅｌｌ研究所のPotmesilらによって開発された画像生成計算機である[Potmesil,1989]。その設計思想は、すでに開発された多くの計算機から影響を受けているが、概ね以下の通りである。

①速度：ＲＩＳＣタイプのＤＳＰによる高速処理。
②並列処理：多数のノードプロセッサを用いた並列処理アーキテクチャ。ローカルメモリの採用。
③パイプライン処理：Geometry Engine[Clark,1982]で使われたパイプライン演算の導入。
④フレームバッファの分散：ノードプロセッサのローカルメモリの一部をフレームバッファとして使用。（この考えは、ＣＡＰ[Sato,1985]ですでに採用されている。）

Pixel Machineは図２－１３に示すように４つの主要ブロックから構成される。

① pipe nodeのパイプラインブロック
② pixel nodeをｍ×ｎの配列に並べた並列ブロック
③ pixel funnel　（ＬＩＮＫＳ－１のデータコレクタに相当する。）
④ video processor

Pipe nodeとpixel nodeは適宜使い分ける。完全に逐次なアルゴリズムはpipe nodeのみを使って実行されるし、完全に並列なアルゴリズムはpixel nodeのみを使って実行される。

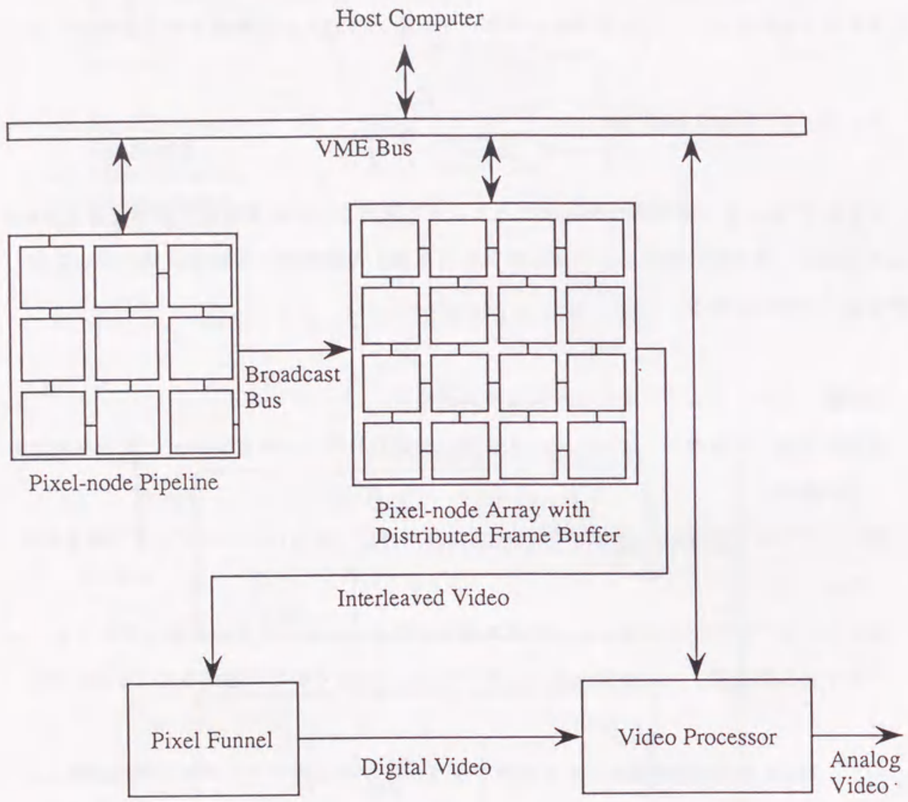


図２－１３　Pixel Machineの構成

たとえば、座標変換などの幾何学計算はpipe nodeで計算し、光線追跡法における各光線の追跡計算はpixel nodeで計算する。

Pixel Machineの問題点は、ノードプロセッサのＣＰＵにＤＳＰを用いているためにアドレス空間が狭い（１６ビットアドレス空間）ことにある。Pixel machineでは、データ領域に関してのみ仮想記憶をサポートしているので、大規模データを扱うことができるが、その場合、ページスワップに伴う処理速度の劣化が避けられない。ＳＩＧＨＴはこの点に関して階層化メモリ構成の採用とアルゴリズムの工夫により実質的に大規模高速メモリを実現し、この種の性能劣化を解消している。

２．５　本研究の位置づけとまとめ

本章では、まず、光線追跡法の基礎概念を述べ、そこから派生する種々の課題に関する研究状況を概観した。ついで、高速化アルゴリズムと並列処理プロセッサの研究状況を述べた。このような、研究状況のもとで、本論文の研究内容の位置づけを以下にまとめる。

（１）動的部分木を用いた光線追跡法

光線追跡法の高速処理を阻害する要因は、交点計算回数である。これを著しく削減するために、動的部分木の概念を導入した新しい光線追跡法を提案する。本論文では、ＢＶの階層化による高速処理を考える。提案手法は、画像の小領域に投影される物体は少ないという事実を利用して、小領域に投影されるＢＶと物体だけからなる部分木を小領域ごとに動的に構成し、その部分木を用いて光線追跡する手法である。動的部分木の利用に加えて、ソーティングと小領域の一様性テストを加えた新たなアルゴリズムを提案する。

（２）光線追跡法指向計算機ＳＩＧＨＴ

光線追跡法の高速処理には、内在する並列性を最大限に引き出すコンピュータアーキテクチャを実現する必要がある。具体的には、（１）の結果から３次元ベクトル演算の並列実行が鍵となる。本論文で提案する光線追跡法指向計算機ＳＩＧＨＴは、３次元ベクトル演算の並列実行と画素毎の並列処理の２つのレベルで並列処理を実現する機構を取り入れた新しいアーキテクチャである。

（３）３次元ベクトル演算の並列実行に関する理論的基礎付け

光線追跡法の基本演算は３次元空間のベクトル、マトリクス演算である。ＳＩＧＨＴのＴＡＲＡＩ演算器はそれらの演算を並列度３で実行するように設計されたものである。しかしながらすべてのベクトル、マトリクス演算が並列度３で実行できる訳ではない。したがって、いかなる演算が並列度３で実行できるか、また並列度を劣化する要因は何であるかを明らかにしなければならない。本論文では、集合論的観点からこの問題を考察し、その解を与える。このような理論的検討が、アーキテクチャの有効性を保証するために重要である。

第３章　動的部分木を用いた光線追跡法

３．１　まえがき

T.Whitted[Whitted,1980]により脚光を浴びた光線追跡法は、幾何光学に基づいて光学系を忠実にシミュレートする手法であり、極めて写実的な画像が生成できる。そのため、アニメーションばかりでなく、ＣＡＤ／ＣＡＭや都市景観シミュレーション、音場解析、電波障害解析等広範な分野に応用されている。

しかしながら、光線追跡法はその計算にかなりの時間を要するという問題点がある。この計算時間の大半は、光線と物体の交差判定に費やされる[Whitted,1980]。そのため、交差判定回数の削減を狙ったデータ構造やアルゴリズムの提案[Glassner,1989]、交差判定計算の高速化を狙ったプロセッサの提案[吉田,1988]など活発な研究開発が行われている。

光線追跡アルゴリズムの高速化の本質は、つきつめれば、各画素に対して、その画素に投影される物体を効率よく求めることにある。それを目的として従来より種々の高速化技法が提案されてきた。従来の手法を分類すると

①データの階層化とBounding Volume（ＢＶ）の使用、

②Ray coherencyの利用、

に大きく分けられる。

データの階層化とＢＶの使用により交差判定回数の削減を図る方法がよく知られている。OctreeやConstructive Solid Geometry（ＣＳＧ）モデルを用いた階層化はその一例である。これらのモデルでは階層化された木の各ノードに対し、そのノードの子孫となる葉ノードのプリミティブをすべて包含するようなＢＶを必要に応じて設定する。これらのＢＶに対して交差判定を行うことにより交点計算の回数を大幅に削減できる。「ＢＶの体積を小さくするほど物体と交差する光線の数は少なくなる」という事実に基づいて、体積ができるだけ小さくなるＢＶを効率的に求める検討が行われている[Weghorst,1984;Kay,

1986]。

一方、ある光線とその近傍の光線とは似た振る舞いをすることが多い。このような Ray Coherency をうまく利用すれば光線追跡法を高速化できることが知られている。この代表例として、Arvoら[Arvo,1989]の方法があげられる。Arvoらの方法では、類似した光線をグルーピングし、これらと交差する物体の候補を求めることにより、交差判定計算を削減しており、高速化効果は大きい。

本章では、階層化されたＢＶに対して Ray Coherency を利用した、より高速な光線追跡アルゴリズムを提案する。この手法では、まず木構造（２進木）のＢＶを構築しておき、光線追跡は画像をいくつかの小領域に分割して処理する。個々の小領域ごとに、木構造ＢＶを調べ、小領域に投影されるＢＶからなる部分木を動的に作成する。この部分木を用いて、領域内の画像生成を行う。一般に、部分木はもとの木に比べて階層が浅いため、ＢＶとの交差判定回数は大幅に削減される。本手法ではさらに高速化を図るため、

①部分木の各ノードレベルでの視点からの距離によるソーティングを行う、

②小領域内の一様性テストをする、

という技法も組み合わせる。これらの相乗効果により、本手法はArvoらの手法よりも高速に画像生成できることが、実験結果およびアルゴリズムの解析から示される。

３．２　光線追跡法の高速化技法

本節では、画素に投影される物体を効率よく求めるという観点から光線追跡法の種々の高速化技法を考察し、インプリメントに際して考慮すべき点を検討する。まず３．２．１節で、データの階層化方法と効率との関係を論ずる。次に３．２．２節で、視点から追跡する光線（１次光線）に対する高速化技法を提案する。３．２．３節では、それらの技法が反射・屈折光線や影（以下、２次光線等と呼ぶ）に対しても適用可能であることを示す。

３．２．１　物体定義データの階層化

物体定義データから木構造のＢＶを構築する場合、一般に以下の手順で行う：

①まず、形状定義の最小単位（プリミティブ）を階層的にクラスタリングして木構造とする。

②次に、木の各ノードごとに、その下のすべてのプリミティブを包含するＢＶを設定する。

光線追跡の効率を左右する要因として、①ではクラスタリングの方法および各ノードの枝（子ノード）の数が、②ではＢＶの形状が、それぞれあげられる。本章では、このうちの子ノードの数とＢＶ形状について考察する。

（１）子ノードの数と交差判定回数

各ノードにおける子ノードの数と、交差判定回数との関係を考察する。例として、完全なｋ進木で階層化した場合を考える。また、木は平衡しているとする。いま、

条件１：ある光線があるノードのＢＶと交差するならば、そのｋ個の子ノードのＢＶの中でただ１個と交差する。

という状況を考える。これがすべての交差ＢＶに対して成り立つとすれば、その光線に対する交差判定回数Ｎは、

$$N = k \cdot \log_k n$$

となる。ここで、ｎはプリミティブ数である。

ｎを固定したときＮを最小化するｋは、ｋ＝ｅ（自然対数の底）である。ｋは整数であるから実際にはｋ＝３がＮの最小値を与える。ついで、ｋ＝２，４となる。２と４は同じ値を与える。したがって、交差するＢＶに対して平均１個程度の子ノードのＢＶと交差するような条件下では、２～４進木による階層化が有利となる。

ｋ個の子ノードのＢＶのうち複数のＢＶと光線が交差すると条件１は成立しない。しかしながら、交差する複数のＢＶから何らかの方法で視点に最も近いプリミティブを含むＢＶを一意に選択できれば、この条件は成立する。本章では、この条件が多くの光線に対して成立するようにアルゴリズムを構築していく。それゆえに本章では、クラスタリング処理の単純さや３．２．２節で述べるソーティングの効率化を考え、２進木を採用する。

（２）ＢＶの形状と交差判定コスト

ＢＶはできるだけ体積が小さくなるように設定することが望ましいが、交差判定コストを考慮した設定をしなければならない。いま、図３－１(a)に示すような各ノードにＢＶが設定された木構造を考え、その画面（スクリーン）への投影が図３－１(b)となったとする。また、ノードn1の投影面積をＳ1とする。ノードn1に光線が当たったときに限り、その子ノードのＢＶとの交差判定が行われる。したがって、その子ノードとの交差判定に要する時間の、画面全体での総和Ｔは、

$$T = k \cdot C \cdot S_1$$

となる。ここで、ｋは子ノードの数（この例では２）、Ｃは子ノードのＢＶ１個との交差判定コストである。

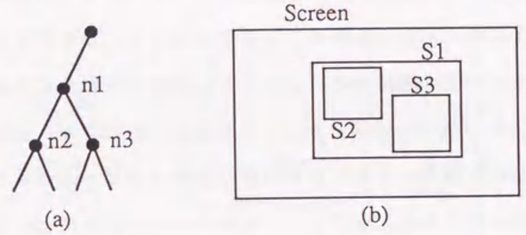


図３－１　ＢＶの投影

（1）で述べた条件1が成立するならば、サーチする木の各ノードに対してこの式が成立する。したがって、単純な形状のＢＶを用いても交差判定コストが小さければ、トータルの計算時間は少なくなり得る。（投影面積が2倍になってもコストが半分なら計算時間は同じである。）

本章では、投影面積は大きくなるが判定コストの小さな、軸（世界座標）に平行な辺を持つ直方体を採用する。

3．2．2　　1次光線に対する高速化

（1）動的部分木

画面の小領域を考えた場合、そこに投影されるプリミティブの数は限られている。そこで、小領域毎に与えられた木から投影されるＢＶを抽出し、動的に部分木を作る。小領域中では部分木を利用して光線追跡することにより、高速化を図ることができる。本章では、画面をメッシュ状の小領域に分割することを考える。このとき、部分木は、視点と領域の4隅の点から作られる角錐に入るかあるいは交差するＢＶから作られる。部分木を作る過程では、図3－2に示す最適化も行う。この部分木作成にさほどコストがかからないことは次の考察からわかる。

（a）角錐の内外判定

角錐は4つの平面から構成されるが、前述の小領域分割を行う場合、これらの平面とＢＶとの位置関係を小領域毎に求める必要はない。画面の大きさ（画素数）をｎ×ｎ、小領域の大きさ（画素数）をｍ×ｍとすれば、角錐を構成するために必要な平面の数は全体で2（ｎ／ｍ＋1）である。したがって、これらの平面について、各ＢＶがどちら側にあるか（もしくは交差するか）を計算すれば、角錐に対する内外判定は容易にできる。また、ＢＶが階層化されていると、あるＢＶに対し、そのＢＶが平面のどちらか一方の側に属すれば（すなわち、面と交差しなければ）、そのＢＶより下の階層にあるＢＶに対する交差判定は不要となる。この事実を使って、角錐と交差するか含まれるＢＶおよびプリミティブからなる部分木を次のようにして作ることができる。

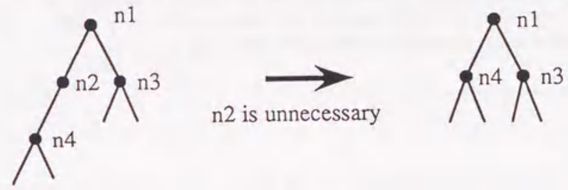


図3－2　部分木の最適化

（ｂ）動的部分木作成

木の各ノードに４つのフラグを与える。各フラグは、角錐の面に対応し、ノードに対応するＢＶあるいはプリミティブが面の内側にあるか、外側にあるか、あるいは交差するかを示す。面に対して木のルートノードからはじめて各ノードのフラグを設定していく。このときあるノードで面の内側かあるいは外側になったらその子ノード以下の当該面に対するフラグ設定はしない。

小領域の画像生成をたとえばスキャンライン方向に行うなら、角錐を構成するスキャンライン方向の２つの面は、その方向の各角錐に共通である。したがって、これらの面に対するフラグはスキャンライン方向毎に１回設定するだけでよい。他の２面に対するフラグ設定は小領域毎に行うが、それはスキャンライン方向の２面に挟まれるＢＶに対してのみ行えばよい。

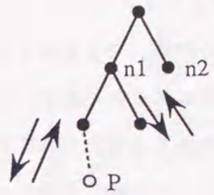
このようしてフラグが設定されたら、そのフラグを見ながら部分木を構成する。その際、面の外側あるいは内側と設定されているフラグは、その情報を子ノードに継承していく。

（２）ソーティング

より視点に近いプリミティブが画像面に投影され、そのプリミティブを包含するＢＶは他のプリミティブを包含するＢＶと比べてもより視点に近いであろう、という考えのもとに、部分木の各レベルで視点からの距離によるソーティングを行う。すなわち、あるノードのＢＶと光線が交差し、その子ノードに対して交差判定を行う必要が生じたら、子ノードを視点からの距離の近い順に並べ替える。これは、光線毎に行う。

深さ方向優先の木探索を行う場合、木の各レベルでソーティングが行ってあれば、図３－３に示すように、ある子ノード以下を探索して得られたプリミティブの視点からの距離が、次の子ノードに設定されたＢＶの視点からの距離より近いとき、その子ノード以下の探索は不要となる。このようにして、ソーティングを利用した枝刈ができる。

２進木を用いれば、子ノードのソーティングは１回の比較だけで済み、極めて効率がよい。



1. Searching descendants of node n1, get a primitive P closest to eye. Assume that distance is t1.
2. Assume that distance between BV set at n2 and eye is t2.
3. If t1<t2, it is not necessary to search descendants of n2.

図３－３　ソーティングを利用した枝刈

（３）領域内の一様性テスト

領域内が同一のプリミティブの投影像であるとき、この領域内は一様であると呼ぶ。領域内が一様であることが何らかの方法でわかれば、この領域では、そのプリミティブのレンダリングを行えばよい。このことは、領域内で木の探索が不要となることを意味し、高速化に与える効果は大きい。ただし、このテストは、できる限り単純なものであることが要求される。

ここでは、次のような簡単なテスト法を提案する。小領域の４隅の点に対して光線追跡する。この段階で、部分木は、４番目に追跡した光線でソーティングされている。今、４隅の点のプリミティブが同一であったとする。それをＰとする。このとき、小領域内でプリミティブＰより視点に近いところに他のプリミティブが無いことを調べればよい。部分木をLeft-most Depth-Firstでサーチし、プリミティブＱを見つける。Ｑは木の一番左端にある。木の構成のしかたから、ＱはＰより視点に近い。ＰとＱが同一でなければこの領域は一様でないとする。いま、ＰとＱが同一であったとする。このときＰの弟ノードを調べる。弟ノードはプリミティブかＢＶが設定されているかいずれかである。それをＲとする。Ｒが４番目に追跡した光線と交差するか調べる。これは、フラグをみるだけでよい。交差すれば、領域内は一様である。そうでなければ、一様でないとする。

このテストは領域内が一様であっても、一様でないという判定を与える場合があるが、領域内が一様と判定されたときには、プリミティブＰの前に他のプリミティブが無いことは保証される。図３－４、図３－５にいくつかの判定の例を示す。なお、４隅にプリミティブが無いとき、領域内にプリミティブが無いための条件は部分木が空であることである。

領域内が一様でないときは、領域を２分割して一様性テストをする。領域の大きさが閾値を下回らない範囲でこれを再帰的に繰り返す。

３．２．３　　２次光線等に対する高速化

（１）反射・屈折光束の追跡

画面の矩形領域を通過する光束は角錐台で表現することが可能であり、３．２．

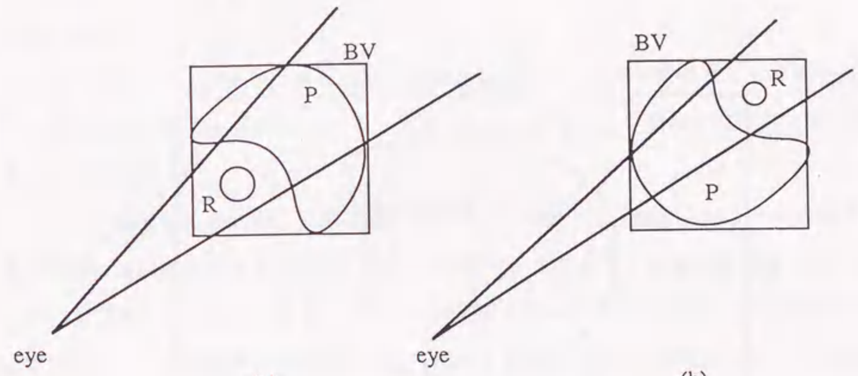


図３－４　一様でないと判定される例（２次元表示）
（ｂ）は本来一様であるにもかかわらず、
一様でないと判定される。

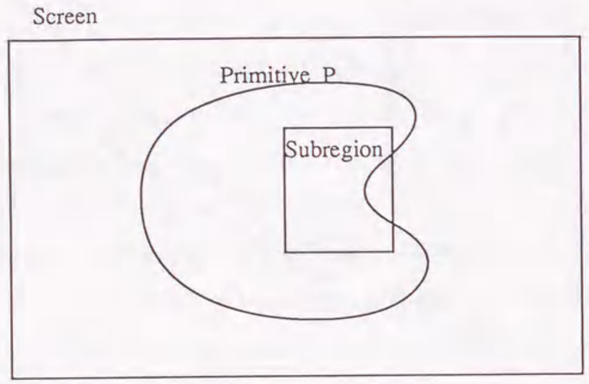


図３－５　一様と判定される例（投影面表示）
この場合、プリミティブＰの前面に別のプリミティブはないが、小領域内にＰでない部分がある。その部分に対しては、部分木を用いた追跡をする。

２節までに述べてきた手法が適用できる。しかし、一般の反射・屈折光束を正確に記述することは困難である。そこで、本章では、

①反射・屈折光束を近似し、それを含む近似的な光束のバウンディングボリューム（以下、ＰＢＶ(Pencil Bounding Volume)と呼ぶ）を求める。
②近似的なＰＢＶに対し、部分木を求め、一様性のテストを行う。
③各反射・屈折光線に対し、①で求めた近似ＰＢＶに含まれるか否かを調べ、含まれる場合には②で得られた結果を利用する。

というアプローチをとる。ここで問題となるのは、いかにして近似ＰＢＶを求めるかという点である。

光束の拡りが小さい場合には、近軸近似が反射・屈折の近似として有効であることが知られている[shinya,1987]。そこで本章では、近軸近似を利用して近似ＰＢＶを求めることにする。

近軸理論によれば、一点（たとえば視点）を発した近軸光束は２つの焦線を持ち、その焦線の方向は直交する[shinya,1987]（脚注）。すなわち、近軸光束は図３－６に示すように、互いに直交する交線を有する２つの平面の組に囲まれた領域として記述される。したがって、この領域を光束の近似として用いることは合理的である。しかし、実際の光束は理想的な近軸光束ではなく、多くの光線が近軸光束の領域外にはみ出すことになる（図３－７(a)）。そこで、光束の４隅の光線を基に、ＰＢＶを求めることを考える。この求め方は多数存在するが、簡単な方法として、４隅の光線の振れ角のうち最も大きな角をＰＢＶの拡り角とする手法がある。図３－７(a)に示した２次元の例では、$|\theta 1| > |\theta 2|$であるので、図３－７(b)のようなＰＢＶが得られる。具体的には、以下の手順で近似ＰＢＶが求められる。

①画面小領域の４隅に対応する光線の反射・屈折光線ｉをSnell則より求める。
②得られた４本の光線の平均をとり、軸光線とする。
③軸光線をｚ－軸とする適当な光線座標系ｘ－ｙ－ｚを求める。ただし原点は、４本の

（脚注）直感的には、光束の波面が２次曲面で近似されることに対応する。

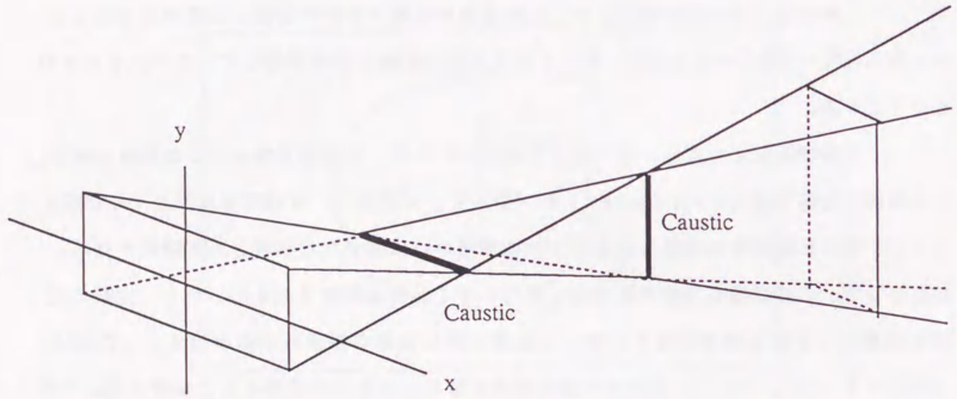
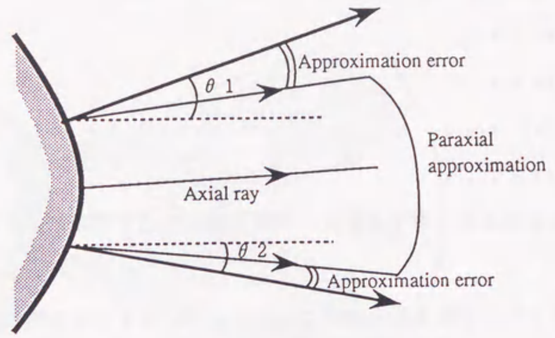
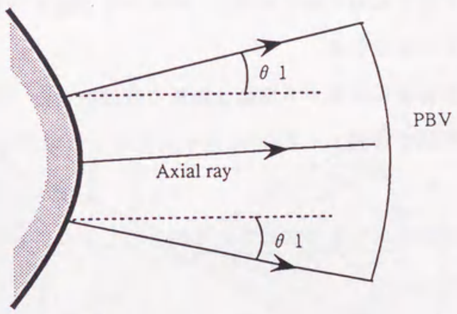


図３－６ 近軸光束の一例



(a)



(b)

図３－７ PBVの構成法

光線の始点の内、いちばん手前のものがｚ＝０となるようにとる。

④４本の光線の始点および方向を光線座標系に変換し、ｚ＝０の面との交点$r_i=(r_{xi}, r_{yi})$、および方向$s_i=(s_{xi}, s_{yi})=((dx/dz)_i, (dy/dz)_i)$を求める。

⑤$r_{xmax}=\max(r_{xi})$, $r_{xmin}=\min(r_{xi})$, $r_{ymax}=\max(r_{yi})$, $r_{ymin}=\min(r_{yi})$および$s_{xmax}=\max(|s_{xi}|)$, $s_{ymax}=\max(|s_{yi}|)$を求める。

⑥求める４平面は次式で与えられる。

$$x - s_{xmax} * z = r_{xmax}$$

$$x + s_{xmax} * z = r_{xmin}$$

$$y - s_{ymax} * z = r_{ymax}$$

$$y + s_{ymax} * z = r_{ymin}$$

なお、ｘ，ｙ軸のとり方は任意であるが、焦線方向にあわせてとるのが効率的である。

この近似ＰＢＶ作成法は最も単純なものであり、さまざまな変形が考えられる。その例として、以下のようなものが考えられる。

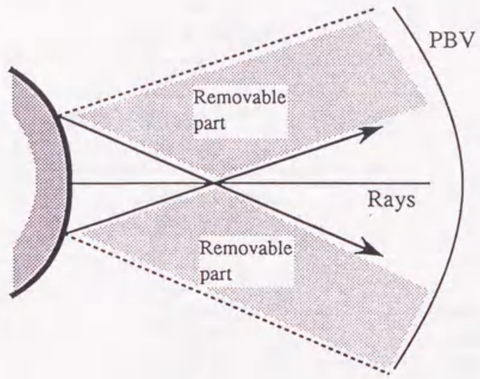
①４隅の光線だけでなく、すべての画素に対応する光線の反射・屈折光線を求め、これらを同様の手法で包含することも可能である。これにより追跡すべきすべての光線がＰＢＶに含まれることが保証される。

②このＢＶでは、光束が反射後に収束する場合は無駄な空間を含む（図３－８(a)）。したがって、収束光束については図３－８(b)に示すような２つの近似光束で包含する。

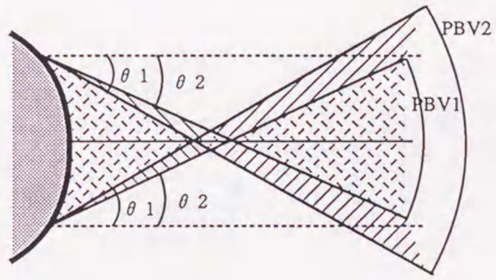
近似ＰＢＶ作成の最適化は今後の課題として残されており、処理効率の実験的な解析が必要であると考えられる。

（２）影

小領域が一様なら、４隅の点から光源を見込むＰＢＶにより部分木を作り、その部分木を用いて領域内の各点が影か否かをテストできる。特に、光源が平行光線なら、ＰＢＶは角柱となる。



(a)



(b)

図３－８　収束光束に対するＰＢＶ

3．2．4　光線追跡アルゴリズム

上で述べた高速化技法を用いて光線追跡アルゴリズムをインプリメントできる。図3－9に概略アルゴリズムを示す。

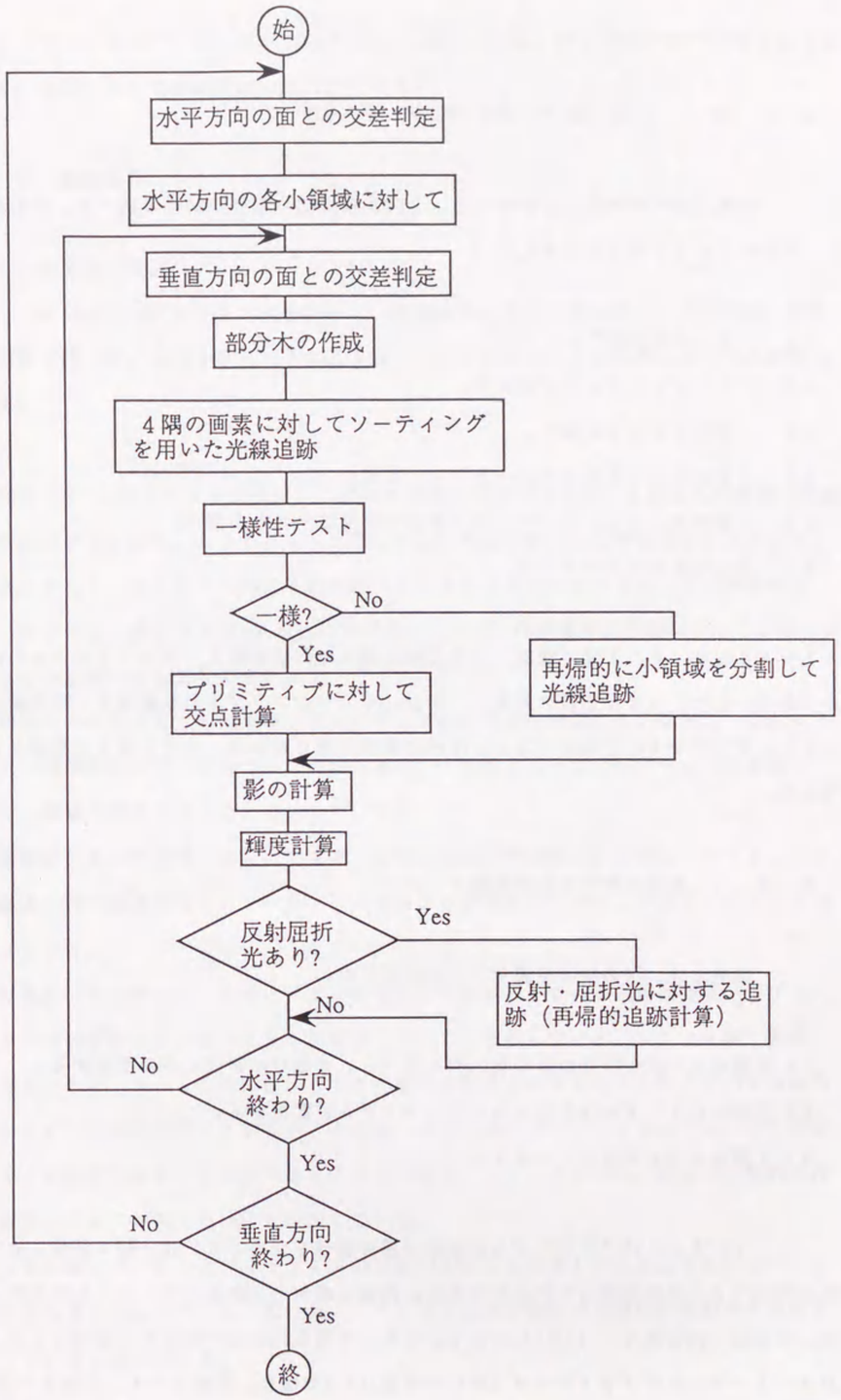


図3－9 提案アルゴリズムの概略フロー

３．３　高速化効果の評価

前節で述べた高速化技法のうち、１次光線に関してその効果を調べる。そのために、以下のアルゴリズムを用意した。

Ａ１　：ＢＶのみを使う。
Ａ２　：ＢＶとソーティングを使う。
Ａ２’：部分木とＢＶを使う。
Ａ３　：部分木とＢＶおよびソーティングを使う。
Ａ４　：部分木、ＢＶ、ソーティングおよび一様性テストを使う。
Ｂ１　：Arvoらのアルゴリズム

Ａ１～Ａ４は３．２．２節で提案した各技法の組み合わせを変えてインプリメントしたものである。また、Ｂ１は比較の対象としてArvoらのアルゴリズムの主要部を１次光線に対してインプリメントしたものである。手法の効率を論じるには、Ａ３とＢ１の比較が公平である。

３．３．１　実験に用いた物体定義データ

実験には、次の物体定義データを使用した。

１）空間内に一定の半径の球を規則的に並べる。２通りの半径の球を用意する。
２）空間内にランダムな半径を持つ球をランダムに配置する。
３）１個および８個のティーポット

１）と２）はアルゴリズムの基本特性を調べるため、３）は一般の画像に対する特性を調べるために用意したデータである。画像生成例を写真３－１～３－５に示す。生成した画像の画素数は、512×512である。また、小領域は8×8の大きさを基本とした。写真３－１～３－３のプリミティブ（球）の総数は512であり、写真３－４、写真３－５のプリミティブ（三角形パッチ）は、それぞれ552、4416である。ＢＶは軸に平行な直方体を使用した。実験に用いた計算機はsun4/260である。

３．２　実験結果

（１）画像生成実験

それぞれのアルゴリズムを用いて、写真に示した画像を生成し、生成時間、交差判定回数を調べた。結果を表３－１および図３－１０に示す。この結果から次の点が指摘できる。

・表３－１(a)からわかるように、いずれの場合もアルゴリズムＡ４が最も短時間で画像を生成する。また、Ａ３とＢ１を比較しても、Ａ３の画像生成時間がＢ１より少ない。
・Ａ１とＡ２’の比較から部分木が高速化に寄与する効果が極めて高いことがわかる。このことは、表３－１(b)に示されるように、Ａ２’の交差判定回数が著しく減少することから得られる効果である。
・領域の一様性テストの効果は、プリミティブの大きさに依存する。写真３－２のように一様領域の少ない場合でも、実験結果は、一様性テストのオーバヘッドを加味しても、高速化効果があることを示している。
・写真３－２の例では、Ａ３よりＡ２’の画像生成時間が短い。これは、ソーティングによる枝刈効果がなく、ソーティングすること自体がオーバヘッドとなったことを示している。
・写真３－４の例では、Ａ３とＢ１の画像生成時間が同じである。画像面上でのプリミティブの重なりが少ないとＢ１が有利となることを示している。（３．４節　参照）
・写真３－４、３－５の例では、Ｂ１の交差判定回数がＡ４やＡ３より少ないにもかかわらず、計算時間はＢ１が多い。これは、Ｂ１では、プリミティブのグルーピングに多くの時間を割かれるためである。逆に、提案アルゴリズムでは、部分木を動的に作成するコストが高くないことを示している。
・Ａ２において、ＢＶとプリミティブの交差判定回数をみるとわかるように、ソーティングを用いることにより、木の各ノードレベルでＢＶ選択の一意性が近似的に成り立っていることがわかる。

（２）小領域の大きさ

小領域の大きさと、画像生成時間の関係を調べる。表３－２および図３－１１に結果を示す。小領域が大きくなるとその中に入るプリミティブの数が大きくなり、部分木が大きくなって、交差判定回数が増える。一方小領域が小さくなると部分木は小さくなるが小領域の数が増えるので部分木を作るコストが大きくなる。したがって、小領域の大きさに最適点がある。表２の結果はそれを裏付ける。最適点はデータに依存するが、8×8の小領域が概ねよい結果を与えることが実験からわかる。

写真３－１

写真３－２

写真３－３

写真３－４

写真３－５

表３-１　光線追跡法の計算時間と交差判定回数

（ａ）計算時間

アルゴリズム	写真１	写真２	写真３	写真４	写真５
A1	36'50"	20'46"	55'18"	12'19"	29'08"
A2	14'07"	17'52"	21'18"	8'42"	18'52"
A2'	5'12"	2'13"	7'16"	3'08"	7'56"
A3	4'39"	2'17"	6'46"	2'29"	5'19"
A4	3'45"	2'10"	5'22"	2'16"	5'03"
B1	6'55"	2'51"	9'34"	2'29"	6'06"

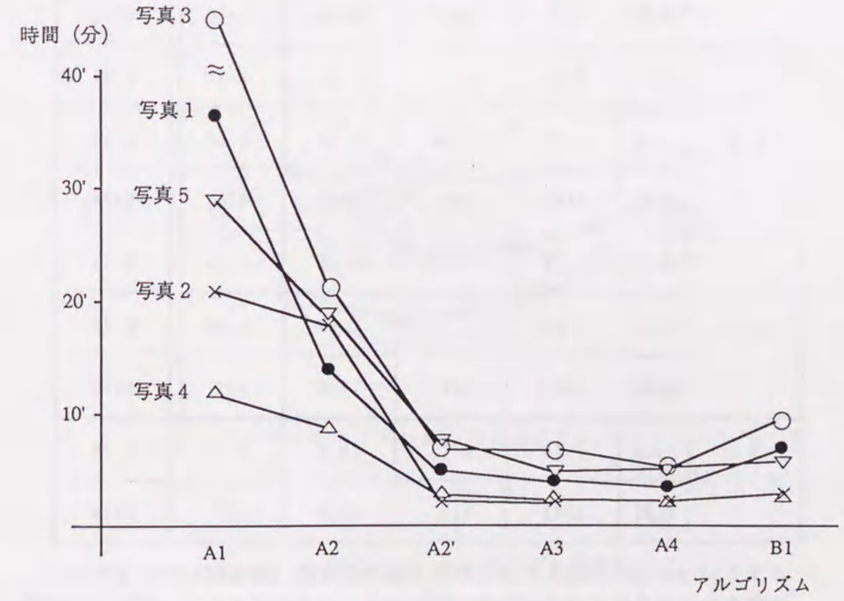


図３-１０　表３-１（ａ）のグラフ表示

表３－１　光線追跡法の計算時間と交差判定回数（続き）
（ｂ）交差判定回数

アルゴリズム		写真１	写真２	写真３	写真４	写真５
Ａ１	ＢＶ	57.9	37.9	86.6	20.4	50.9
	prim	19.8	6.88	28.8	4.61	9.60
	総数	20355	11744	30245	6552	15849
Ａ２	ＢＶ	20.2	29.7	30.9	13.5	30.5
	prim	3.75	5.01	7.28	2.53	4.69
	総数	6281	9087	10008	4190	9228
Ａ２’	ＢＶ	3.72	0.66	5.68	2.56	8.12
	prim	3.38	1.61	4.39	2.78	7.04
	総数	1859	595	2640	1400	3974
Ａ３	ＢＶ	3.16	0.65	5.09	1.86	4.88
	prim	2.27	1.56	3.48	1.60	3.42
	総数	1424	581	2248	908	2174
Ａ４	ＢＶ	1.93	0.65	3.18	1.75	4.61
	prim	2.04	1.79	2.99	1.52	3.20
	総数	1029	546	1608	893	2115
Ｂ１	prim	7.80	3.10	10.8	2.51	5.79
	総数	2045	813	2829	657	1516

ＢＶとprimの項は光線１本あたりの交差判定回数（全光線に対する平均）
総数の項は画像面全体での総交差判定回数（ＢＶ＋プリミティブ）（$\times 10^3$）

表3-2　小領域の大きさと計算時間（アルゴリズム：Ａ４）

小領域の大きさ	写真１	写真２	写真３	写真４	写真５
４×４	5'29"	2'57"	7'23"	2'34"	5'27"
８×８	3'45"	2'10"	5'22"	2'16"	5'03"
１６×１６	3'45"	2'35"	5'17"	2'39"	6'30"
３２×３２	4'39"	4'05"	5'57"	3'30"	9'13"

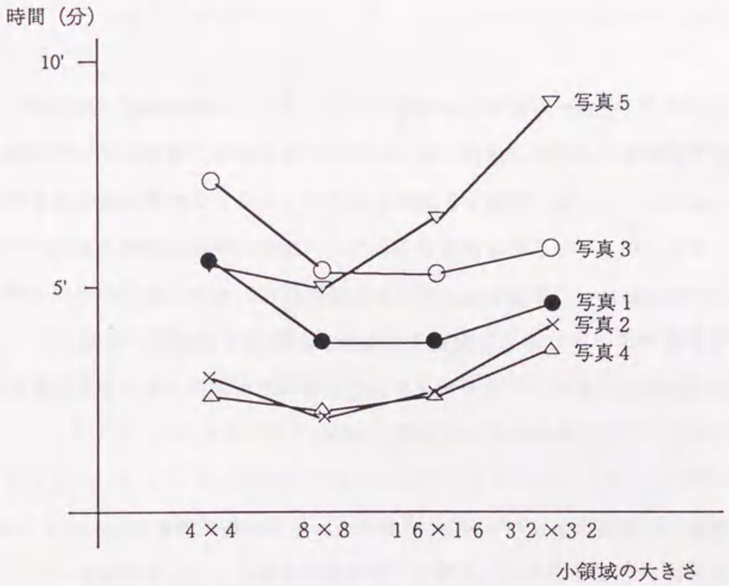


図3-11　表3-2のグラフ表示

３．４　考察

３．４．１　アルゴリズムの解析

アルゴリズムの計算時間のオーダを解析し、アルゴリズムの良さを検討する。以下は、一次光に対する解析である。

（１）提案アルゴリズム

我々のアルゴリズム（Ａ４）の計算コストは、動的に部分木を作るコスト（すなわち、角錐によるＢＶおよびプリミティブのクリッピングコストと部分木を作るためのポインタ操作のコスト）、および、ソーティングとエリアコヒーレンシーを用いた交差判定コストの合計である。これらのコストを、以下で検討する。なお、以下では、全画面の画素数をｎ×ｎ、小領域の画素数をｍ×ｍ、プリミティブ数をＮとする。

クリッピング：　提案アルゴリズムでは、スキャンライン方向の面に対するＢＶの交差判定が主要項となる。その面の数は（ｎ／ｍ＋１）面である。各面に対して必要な交差判定回数は、少なくとも、面と交差するＢＶおよびプリミティブ数の程度は必要である。この回数は、プリミティブの分布に依存するので、一般的な評価は困難である。ここでは、いくつかの分布を仮定して評価する。明らかに最悪値は、各面に対して２Ｎとなる（すべてのプリミティブがすべての面と交差する場合）。最良の場合は、各面に対してｌｏｇ（Ｎ）の程度となる。また、プリミティブが空間に一様に分布しているとすると各面と交差するプリミティブ数は平均ｍＮ／ｎ程度となる。

部分木作成：小領域の数は（ｎ／ｍ）^{2}である。与えられた木をトラバースして部分木を作るが、すでにクリッピングのところで、交差情報がわかっているから各ノードで必要な操作は比較とポインタ操作である。部分木作成の平均コストをγとすると、全体では、

$$\gamma\,(n/m)^2$$

となる。このコストは次の光線追跡のコストと比べると無視できる程度に小さい。

光線追跡：部分木に対する平均の交差判定回数をβとすると、βn^2が画面全体の光線追跡回数となる。

（２）Arvoらのアルゴリズム

Arvoらのアルゴリズムは、プリミティブのグルーピングのコストとグルーピングしたプリミティブに対する交差判定コストの合計である。

グルーピング：Ｎ個のプリミティブをグルーピングするための交差判定回数は、Ｎｌｏｇ（Ｎ）の程度である。

光線追跡：グルーピングされたプリミティブ数の平均値をαとすると、交差判定回数は、αn^2である。

上記において、αはプリミティブの投影面における重なりに依存する。また、β、γは、角錐内に入るプリミティブ数に依存する。したがって、β、γは投影面における重なりとも関連する。

（３）両手法の比較

プリミティブ数Ｎが大きい場合、クリッピングとグルーピングが主要項になる。その場合、上記考察から多くの場合提案アルゴリズムのほうが交差判定回数は少なくなる。

一方、画素数n^2が大きい場合には、提案アルゴリズムにおいては部分木に対する交差判定のコスト（βn^2）が、またArvoらのアルゴリズムにおいては光線追跡における交差判定コスト（αn^2）が主要項となる。その場合、プリミティブの投影面における重なりが少ないとArvoらの手法、多いと提案手法が有利となる。実験結果はそれを裏付けている。

３．４．２　小領域の大きさ

提案手法で、β、γはｍの大きさに依存する。ｍを小さくするとクリッピングコ

ストが大きくなり、mを大きくすると光線追跡計算コストが大きくなる。したがって、総コストを最小とするmがあることが裏付けられる。これは、実験の結果を支持する。

3．4．3　メモリ量

提案手法は、与えられた木の中ですべての処理ができる。ただし、部分木を作るためのポインタと、クリップ情報を保持するフラグのエリアが各ノードに必要である。また、小領域を処理の単位とするので、計算機内で、メモリアクセスのローカリティが高くなる。したがって、キャッシュメモリのミスヒットが少なくなり、一層の高速処理につながる。

3．4．4　主要な演算

提案アルゴリズムは、以下の部分から構成される。

1）部分木の作成、

2）部分木に対する光線追跡。

各部の主な演算を調べる。部分木の作成では、平面とＢＶの交差判定計算、部分木を作るためのポインタ操作、が主な処理である。交差判定計算は内積計算などの3次元ベクトル演算である。これらの処理のうち計算時間を要するのは交差判定計算である。部分木に対する光線追跡計算では、光線とＢＶあるいはプリミティブとの交点計算が最も計算時間を要する。この計算も3次元ベクトル演算が主要な演算である。したがって、3次元ベクトル演算の並列処理ができれば、一層の高速処理が可能となる。

3．5　まとめ

本章では、新しい光線追跡手法を提案し、その高速化効果を検証した。この手法は、与えられた木構造データから動的に部分木を作り、その木に対して、ソーティングと領域の一様性テストを併用して光線追跡を行う手法である。本手法で用いた技法の高速化への寄与は、

①部分木を用いることにより、木の高さを減少させ、木の探索コストを低くする。

②ソーティングにより、木の枝刈効率を高める。

③一様性テストにより、領域内でレンダリングすべきプリミティブがただ1個か否かを判定する。

である。

実験により、部分木が高速化に寄与する効果の大きいことが実証された。また、これらの手法を組み合わせることにより極めて効率の良い光線追跡アルゴリズムを構成できることが示された。光線追跡法の計算コストは、交差判定コストの占める割合が極めて大きいことが指摘されているが、本手法はこの交差判定回数を著しく減少させる（実験の範囲では1/4～1/20に減少）。

さらに、従来、高速と言われているArvoらの手法と比較しても本手法は高速であることを計算機実験およびアルゴリズムの解析を通して示した。

本章で提案したアルゴリズムの主要な演算は3次元ベクトル演算であり、その並列実行を行えばさらに短時間で画像生成ができることを指摘した。第4章では、その並列実行を行う機能を有する計算機ＳＩＧＨＴについて述べる。

第４章　光線追跡法指向計算機ＳＩＧＨＴ

４．１　まえがき

光線追跡法により生成される画像の写実性は、他の生成手法による画像のそれを凌駕するものである。そのため、ACMのSIGGRAPHをはじめとして、各地のＣＧショーで光線追跡法によるフィルムが公開され高い評価を受けている。しかしながら、計算コストが高いという問題点のためいま一つ普及が進んでいないように思われる。この問題点を解決すれば幅広く利用されることは明白であると信じる。

さらに光線追跡法はＣＧの他に音場の解析、電波障害計算、アンテナ設計等広範囲の応用分野を有し、処理の高速化がこれら分野に与える波及効果はきわめて大きい。

このような状況から高速計算機の要求が高まっている。阪大のLINKS[Nishimura, 1984]をはじめとして種々の専用計算機の開発が行われている。これらの計算機はいづれも並列処理による高速化を狙っている。そして、並列処理の手法にそれぞれの特徴がある。

多数のプロセッサを用いればプロセッサ数に比例して性能が上がるということは一般には成り立たない。それは、

①プロセッサ数が増加するにつれてホストプロセッサとの通信やプロセッサ間通信の時間が無視できなくなる、

②システム規模の増加につれシステムの信頼性が低下し、稼働状態にないプロセッサが生じ得る、

からである。そのため一層の性能向上を図るにはプロセッサ自身を高速化し、少ないプロセッサ数で高い性能を引き出さねばならない。プロセッサ自身の高速化を考えるとき、素子技術には限界があるからプロセッサ内並列処理による性能向上を図らなければならない。従来の（光線追跡法の高速処理を狙った）計算機はこの点について十分な考慮がなされていなかった。本章は、この点について注意深く考察するものである。

以下、この章では、光線追跡法の並列処理を狙った専用計算機ＳＩＧＨＴについてそのアーキテクチャ、ハードウェア構成、性能解析結果、ソフトウェア開発環境について述べる。３章で指摘したように光線追跡法の高速処理には、３次元ベクトル演算の並列実行が不可欠である。ＳＩＧＨＴの主眼は、３次元ベクトル演算の並列実行機構を内蔵したアーキテクチャにある。

以下、本章の各節では次の内容について述べる。４．２では光線追跡法に内在する並列性を解析する。光線追跡法は、画素レベルおよび演算レベルの２レベル並列処理が可能であることを示す。４．３ではこれらの並列処理を実現した専用計算機ＳＩＧＨＴのアーキテクチャを示す。ＳＩＧＨＴでは、画素レベルの並列処理をマルチプロセッサ構成で実現する。また、演算レベルの並列処理を複数の演算器で構成されるＴＡＲＡＩと名付けたユニットで実現する。さらに、プロセッサ（ＰＥ）をＴＡＲＡＩを含めた複数のユニットで構成し、これらのユニットを並行動作させることにより一層の並列処理効果を高める。４．４ではＳＩＧＨＴ上の光線追跡アルゴリズムを示す。これはＳＩＧＨＴのＰＥの各ユニットを最大限並列に動作させることを目標としたアルゴリズムである。４．５ではＳＩＧＨＴの性能解析・評価を行う。ＰＥ構成の有効性を実証するために、１ＰＥ構成の試作機およびマイクロプログラムレベルでＳＩＧＨＴを模擬するシミュレータを開発した。シミュレータ上で光線追跡プログラムを実行し、演算器使用効率等の統計情報を収集する。これを解析して、ＰＥ構成の有効性を示す。また、試作機上で実行時間を計測し商用機との実行時間比較により性能評価を行う。４．６ではＳＩＧＨＴ用高水準言語ＳＩＧＨＴ／Ｃ（Ｃ言語）の設計思想を述べる。４章付録ではＳＩＧＨＴのソフトウェア開発環境を示す。

４．２　光線追跡法に内在する並列性

光線追跡法の実行に膨大な時間を要する主な理由は、すでに述べたように

①各画素毎に光線の追跡を行なうこと、

②光線と個々の物体との交点計算に多くの計算を必要とすること、

の２点にある。これらの並列処理ができれば、光線追跡法の高速処理が可能となる。

②に関してT. Whittedは、交点計算に要する計算時間の光線追跡法全体の計算時間に占める割合は、単純なシーンで約７５％、複雑なシーンになると９５％を越えると報告している[Whitted,1980]。したがって、②に関して交点計算の並列計算が行えれば、光線追跡法の高速化効果は極めて大きい。

この節では、上記の並列処理を狙って、光線追跡法に内在する並列性を探る。

４．２．１　画素レベルの並列性

光線追跡法の基礎理論である幾何光学では、光線間の干渉を考慮しない。したがって、スクリーン上の１画素に対応する光線の追跡計算は、他の画素に対する追跡計算とは独立に処理できる。それゆえ、光線追跡法の処理を光線１本に対する処理に分割することにより、マルチプロセッサで容易に並列処理できる。この場合、プロセッサ間の通信は基本的に不要であり、疎結合マルチプロセッサシステムで並列処理できる。そのため、ｎが大きくないときはｎ台のプロセッサでｎ倍の高速化という台数効果を達成することが可能である。LINKS-1、CAPは、このレベルの並列処理を実現したマルチプロセッサシステムである[Nishimura,1984;Sato,1985]。実際、出口[出口,1984]は、LINKS-1上で、６４台までのプロセッサで実験を行い、ほぼ台数に比例した性能向上（６４台の時６３倍の性能向上）が達成できたと報告している。しかしながら、メモリ容量の制限で、画像生成に必要なデータがすべてメモリに入らないときには、データをロードするためにホストプロセッサとの通信が発生し、性能の低下が起きる。（このような状況は、たとえば、マッピングデータを多用する時に起きる。）また、システムの信頼性がプロセッサ数の増加とともに低下し、いくつかのプロセッサが故障した状況下でシステムを運転することも起きる。したがって、なるべく少ないプロセッサ数で高速な処理が要求される。すなわち、プロセッ

サ内並列処理が要求される。

４．２．２　演算レベルの並列性

光線追跡法の主要な計算は、３次元（３Ｄ）空間のベクトル演算として特徴づけられる。そこで、以下では、光線追跡法の主要部を占める交点計算に関して、３Ｄベクトル演算の並列処理を検討する。

光線と二次曲面の交点計算について考える。二次曲面の方程式は、次の二次形式で与えられる。

$$r M' r = 1 \qquad (4-1)$$

ここで、'ｒ＝'（ｘ，ｙ，ｚ）はｒの転置ベクトル、Ｍは係数行列であり、

$$M = \begin{pmatrix} A & F & E \\ F & B & D \\ E & D & C \end{pmatrix}$$

で与えられる。

一方、光線は３次元空間の半直線であり、その方程式は、パラメータ表示で次式で与えられる。

$$r = a t + v \qquad (4-2)$$

ここで、

$a = (a_x, a_y, a_z)$：方向余弦ベクトル

$v = (v_x, v_y, v_z)$：視点ベクトル

である。

式（４－２）と式（４－１）より、次式を得る。

$$a M' a t^2 + a M' v t + v M' v = 1 \qquad (4-3)$$

この二次方程式を解くことにより、交点の位置がわかる。（４－３）式を具体的に書けば、

$$
\begin{aligned}
{}^{a}M^{t}a \;=\;& A\cdot a_x^2+B\cdot a_y^2+C\cdot a_z^2+\\
& 2D\cdot a_y a_z+2E\cdot a_z a_x+2F\cdot a_x a_y\\
{}^{a}M^{t}v \;=\;& A\cdot a_x v_x+B\cdot a_y v_y+C\cdot a_z v_z+\\
& D\cdot a_y v_z+E\cdot a_z v_x+F\cdot a_x v_y+\\
& D\cdot a_z v_y+E\cdot a_x v_z+F\cdot a_y v_x\\
{}^{v}M^{t}v \;=\;& A\cdot v_x^2+B\cdot v_y^2+C\cdot v_z^2+\\
& 2D\cdot v_y v_z+2E\cdot v_z v_x+2F\cdot v_x v_y
\end{aligned}
$$

となる。図４－１に${}^{a}M^{t}a$の計算のフローグラフを示す。

計算コストは、二次方程式を解くことよりも係数を求めるほうにかかる。その理由は、平方根の計算は、テーブル引きによる高速計算法（４．３．３節参照）が使えるからである。したがって、交点計算を高速に行なうことは、式（４－３）の係数を高速に求めることに帰着する。

図４－１からもわかるように、これらの計算に共通する処理は、

（１）積項がX,Y,Z 軸成分に関して対称性を持ち、独立に計算できる。

（２）$a_y a_z$, $a_z a_x$, $a_x a_y$のようなクロス積項は添字X,Y,Z に関して順繰りに並んでいる。

という特徴を持つ。

平面と光線、直方体と光線の交点計算も、内積計算を主体とし、また、輝度計算も、内積、ノルム計算等の３次元（３Ｄ）ベクトル演算であり、それらの演算には上記の特徴がある。

さらに色情報も(R,G,B) をベクトルとみなすことにより、３Ｄベクトル演算の枠組みのなかで取り扱うことができる。

上記の特徴を持つ３Ｄベクトル演算は、３組の浮動小数点演算器（ＦＯＰＵ）とメモリをネットワークで結合した構成のプロセッサで効率良く実行できる。以下では、３

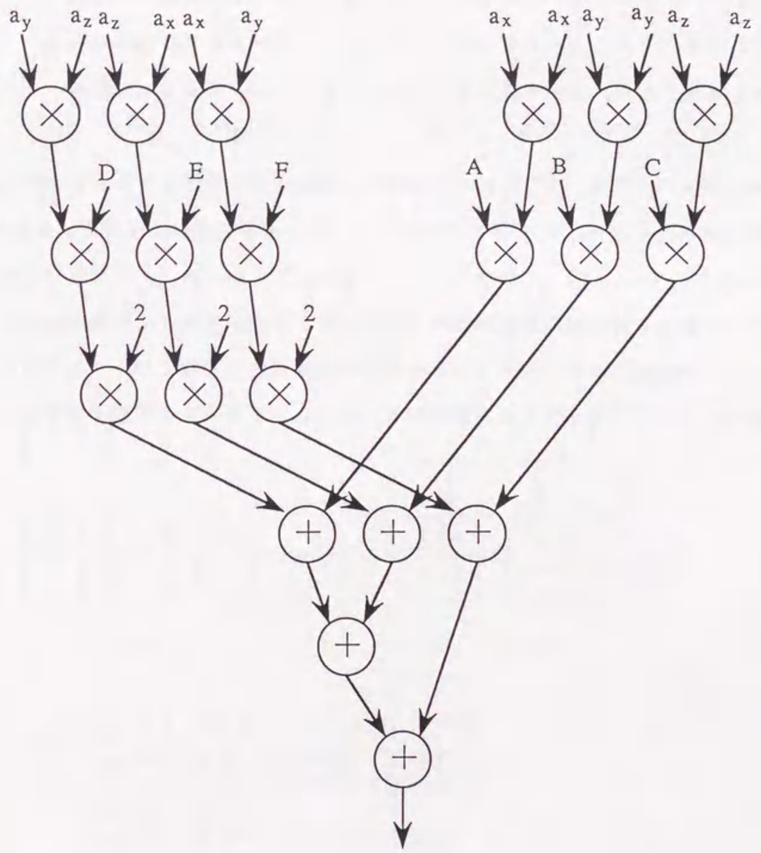


図４－１　${}^{a}M^{t}a$の計算のフロー

個の演算器をそれぞれｘ－，ｙ－，ｚ－ＦＯＰＵ、３個のメモリをそれぞれｘ－，ｙ－，ｚ－メモリと呼ぶことにする。各々のＦＯＰＵ－メモリ対は、割り当てられた軸(x,y,z)に関する演算を実行する。各軸に対応する変数は対応するメモリに格納し、対応するＦＯＰＵはそれらの変数に対応するベクトル演算をおこなう。そして、演算結果は対応するメモリに格納される。ネットワークは、次のような結合ができればよい。

（１）ｘ－，ｙ－，ｚ－メモリからｘ－，ｙ－，ｚ－ＦＯＰＵへの転送
（２）ｘ－，ｙ－，ｚ－メモリからｙ－，ｚ－，ｘ－ＦＯＰＵへの転送
（３）ｘ－，ｙ－，ｚ－メモリからｚ－，ｘ－，ｙ－ＦＯＰＵへの転送
（４）ｘ－，ｙ－，ｚ－ＦＯＰＵからｘ－，ｙ－，ｚ－メモリへの転送

例えば、積$a_y a_z$，$a_z a_x$，$a_x a_y$の計算は、図４－２に示すように実行される。すなわち、あらかじめa_x, a_y, a_zをそれぞれX-，Y-，Z-メモリに格納しておく。最初に、a_y，a_z，a_xつづいてa_z，a_x，a_yがネットワークを介して　X-，Y-，Z-ＦＯＰＵに転送される。次いで、対応する変数間の乗算が並列に実行され、結果がメモリに格納される。

この構成により、メモリアクセス競合を生じることなくデータを並列にREAD/WRITEできる。また、メモリアクセス競合がないのでネットワークの構成が非常に簡単になる。

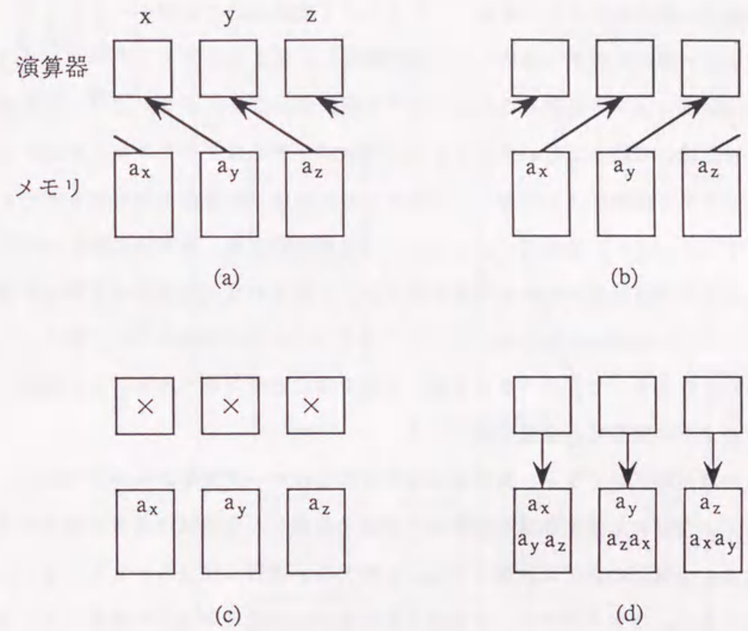


図４－２　積 $a_y a_z, a_z a_x, a_x a_y$ の計算
(a)データを左上の演算器に転送する。
(b)データを右上の演算器に転送する。
(c)乗算を行なう。
(d)結果を真下のメモリに転送する。

４．３　ＳＩＧＨＴアーキテクチャ

この節では、光線追跡法の２レベル並列処理を実現した専用計算機ＳＩＧＨＴのアーキテクチャについて述べる。

４．３．１　ＳＩＧＨＴマルチプロセッサシステム

光線追跡法の画素間独立性に着目し、ＳＩＧＨＴは基本的には図４－３に示すようなマルチプロセッサ構成を採る（画素レベル並列処理）。各ＰＥとホストプロセッサは２通りの通信路（制御コマンドを送るためのシリアル回線およびプログラム／データを送るためのパラレル回線）で結合され、各ＰＥとフレームバッファはデータコレクタを介して結合される（シリアル回線による結合）。データコレクタは、ＰＥから送られるデータの調停（アービトレーション）を行なう。さらに、（光線追跡法は、画素毎に独立に計算できるから、ＰＥ間の通信は基本的には不要であるが、）将来的な利用形態の拡張を考慮して、ＰＥ間のシリアル回線を用意する。各ＰＥは４本のシリアル回線を持つ（図４－３には示してない）。したがって、ＰＥを格子状に結合することが可能である。この構成で、ＳＩＧＨＴは次のように動作し、画像を生成する。

まず、光線追跡プログラム、画像生成のための各種データ等をあらかじめ各ＰＥにロードしておく。生成する画面領域を多数の小領域に分割し、小領域の画像生成処理をＰＥに割り付ける。小領域の割付は静的とする。すなわち、実行に先立ち、ＰＥが受け持つ小領域を決めておく。ホストプロセッサは各ＰＥに画像生成処理の実行を指令する。ＰＥは、割り付けられた小領域の画像を生成し、データコレクタを介してフレームバッファに出力する。このようにして、すべての小領域の処理が終了すると一枚の画像が生成される。

４．３．２　ＰＥの構成

演算レベルの並列処理をＰＥで実現する。図４－４にＰＥの構成を示す。ＰＥは、４つのユニット（ＴＡＲＡＩ、ＭＰ、ＤＭＡ、ＤＢＭ）から構成される。ＴＡＲＡＩは、

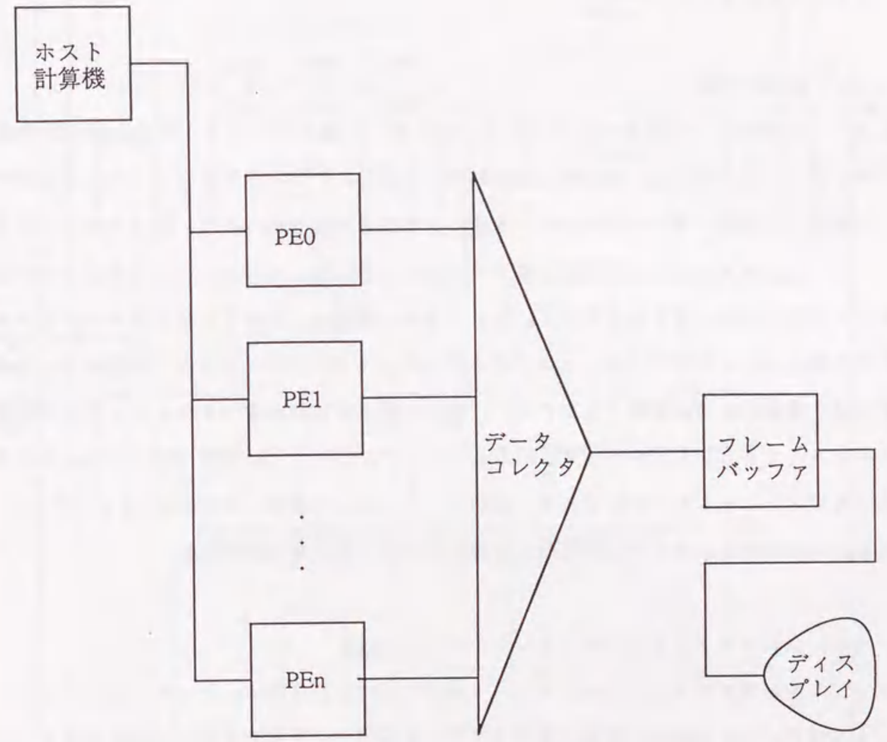


図４－３　ＳＩＧＨＴシステム構成

前述の３Ｄベクトル演算の並列処理を行なう。ＭＰは光線追跡法のフローの制御、データ転送制御、ホストプロセッサとの通信等を行なう。ＤＭＡはＤＭＡ（Direct Memory Access）転送制御を行なう。ＤＢＭには、画像生成のための各種データを格納する。ＭＰとＴＡＲＡＩは、それぞれ独自のマイクロプログラムで制御される。これらのユニットを４．４節で示すように並列動作させ光線追跡法の効率的な並列処理をＰＥで実現する。図４－５にＰＥの詳細構成を示す。以下、この図に従って、各ユニットの構成を説明する。

（１）ＴＡＲＡＩユニット

（ａ）構成の概要

３Ｄベクトル演算を行なうＴＡＲＡＩは、３組の３２ビット浮動小数点演算器（ＦＯＰＵ，ここではWeitek社WTL1164/1165を使用した）とレジスタファイルを３本のバスで結合して構成する。このバスは、後述のような時分割バスであり、図４－４に示すＦＯＰＵ・レジスタファイル間を結合するネットワーク機能とＤＢＭ・レジスタファイル間のデータ転送機能の双方を実現する。ネットワーク機能は、ＦＯＰＵがアクセスするバスを切り替えることで実現する。バスの切り替えは、マイクロプログラムで制御する。ＦＯＰＵは、浮動小数点乗算器（ＦＭＰＹ）、浮動小数点算術演算器（ＦＡＬＵ）および関数テーブル（ＴＡＢＬＥ）から構成される。レジスタファイル（XREGF～ZREGF）は、それぞれ４ＫＷ（３２ビット／Ｗ）である。関数テーブルは、平方根、SIN/COSの各テーブルからなる。一次補間によりこれらの関数を計算する（４．３．３節参照）。

（ｂ）レジスタファイルのアッドレッシング

各レジスタファイルは６４ワードのページに分割される。従って、各レジスタファイルは６４ページ構成となる。レジスタファイルはページ番号とページ内オフセットを指定してアクセスする。その指定方法の詳細は（ｃ）で述べる。

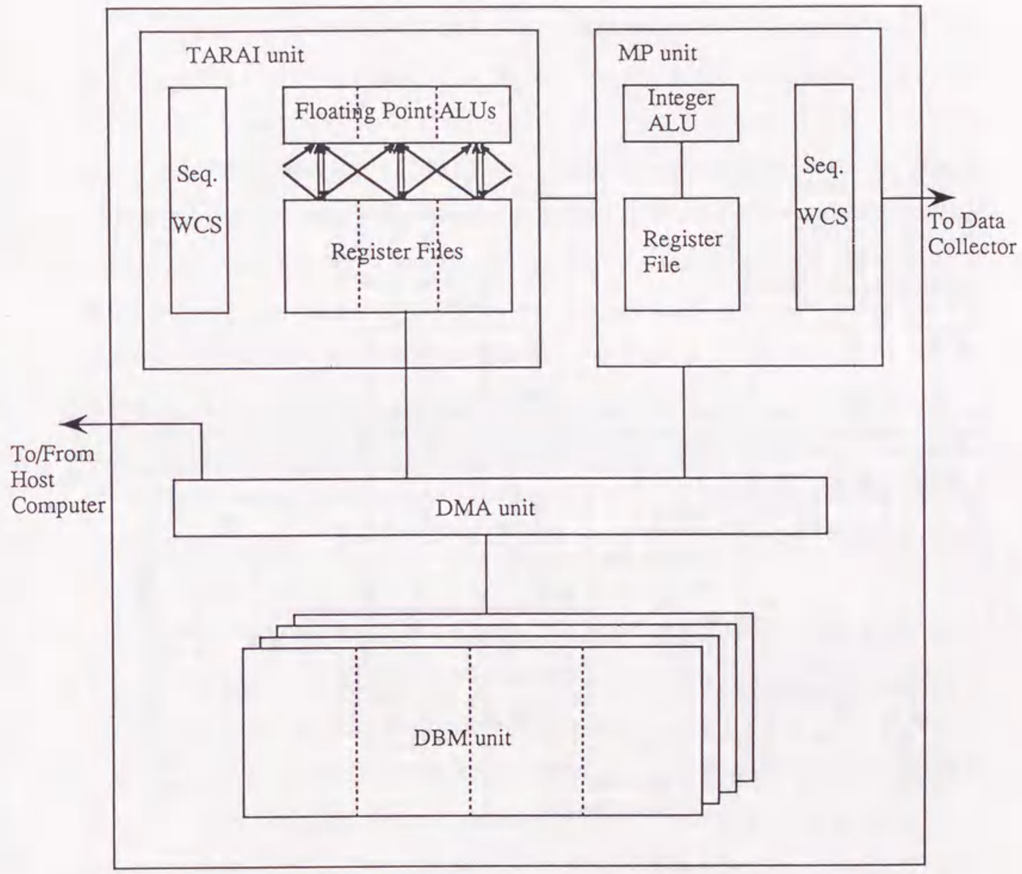


図４－４　ＰＥの構成

MM: Memory Manager
DBM: Data Base Memory
Seq.: Microprogram Sequencer
WCS: Writable Control Store

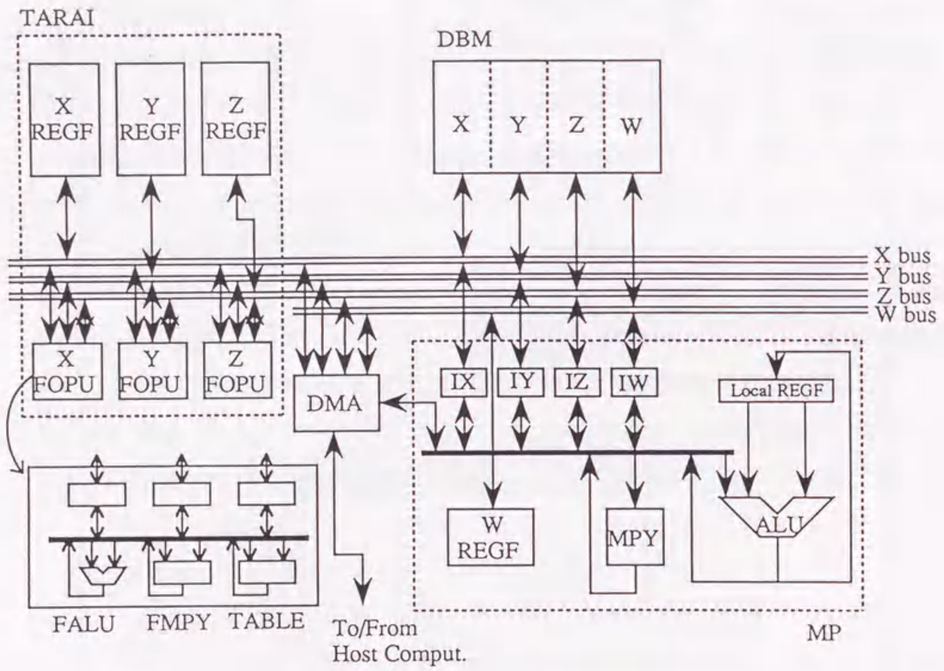


図４－５　PEの詳細構成

（ｃ）マイクロプログラムのフィールド構成

図４－６にマイクロプログラムのフィールド構成を示す。図は機能別にフィールドを示している。マイクロプログラムは一語１０２ビット構成であり、４ＫＷの容量を持つ。主なフィールドの意味を以下に説明する。

OPECODEフィールドには３個の演算器に対する命令を書く。したがって、演算器間で異なる命令を実行することができる。各演算器は２入力１出力構成である。

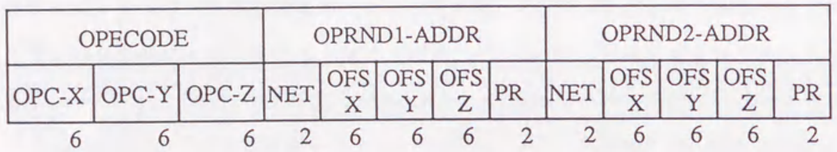
OPRNT1-ADDRとOPRND2-ADDRフィールドには演算器のそれぞれの入力データを与えるアドレスを書く。それぞれのフィールドは５つのサブフィールドから構成される。NETにはネットワーク制御コード、ofs x,ofs y,ofs zにはページ内アドレスオフセットを与える。PRは、ページ番号の選択を行う。PRは２ビット構成であり、その値が0～2の時はページ0～2を意味する。値が3の時はページレジスタを意味する。すなわち、ページレジスタの値とofsフィールドの値の和がアドレスとなる。従って、ページ3～63をアクセスするときは、あらかじめページレジスタにページ番号をセットしておく必要がある。ページレジスタへの値のセットはＭＰから行う。OPRND3-ADDRフィールドには演算結果の格納アドレスを書く。サブフィールドは２つであり、ofsにはページ内アドレスオフセット（それぞれのレジスタファイルに共通）、PRにはページ選択コード（上記と同様）を与える。

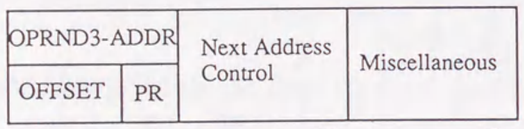
Next Address Controlフィールドはシークエンサの制御フィールドである。

（ｄ）時分割バス

ハードウェア量を節約するために、ＴＡＲＡＩのネットワークとＤＢＭ－レジスタファイル間のデータ転送路を１組のバスで実現する。そのため時分割バスを採用する。

図４－７に時分割バス（X-bus～Z-bus,W-bus）のタイムスロットをしめす。バスの基本クロックは１２５ナノ秒である。ＴＡＲＡＩは、６クロック（７５０ナノ秒）を単位として、浮動小数点演算を行なう。すなわち、Ｔ１、Ｔ２で演算データを取り込み、Ｔ３～Ｔ５で演算を実行し、Ｔ６で結果をレジスタファイルに格納する。したがって、ＴＡＲＡＩは最高４ＭＦＬＯＰＳの性能となる。ＦＯＰＵが演算を実行している間、バスは開放される。その期間、バスはＴＡＲＡＩのレジスタファイル（Ｘ～Ｚ）とＭＰのレジスタ（ＩＸ～ＩＺ）との間のデータ転送（Ｔ３、Ｔ４）およびＤＢＭとレジスタファイルとの間のデータ転送（Ｔ５）に使用する。これにより、ＴＡＲＡＩ演算とデータ転送の並列実行ができる。





(注)
上下２段のフィールドは
　上段：フィールド名
　下段：サブフィールド名
数字はフィールドの長さ

図４－６　TARAIマイクロプログラムのフィールド構成

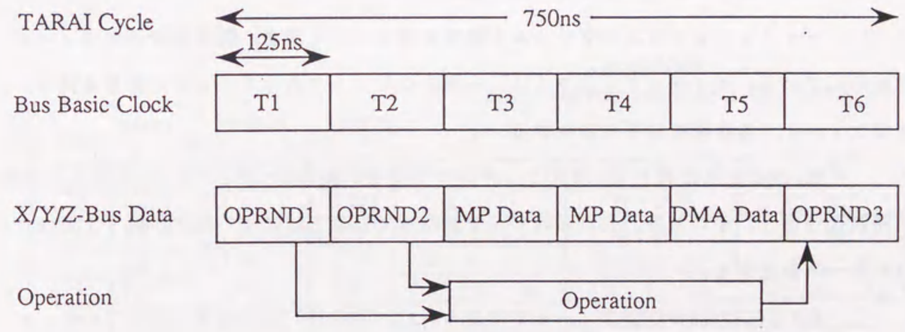


図４－７　バスのタイムチャート

（ｅ）マイクロプログラムの実行例

図４－８に交点計算の一部のマイクロプログラム実行例を示す。図からわかるようにこの例ではＴＡＲＡＩの３個の演算器が全部稼働することがわかる。

（２）ＭＰユニット

（ａ）構成の概要

ＭＰは、固定小数点演算を主体とするので、３２ビットＡＬＵと３２ビット乗算器を中心とした構成としている。乗算器はメモリアドレスの高速計算に用いる。レジスタファイル（WREGF）は、１６ＫＷ（３２ビット／Ｗ）である。基本クロックは１２５ナノ秒（８ＭＩＰＳ）であり、バスの基本クロックと同期している。ＭＰは、４個のレジスタ（ＩＷ～ＩＺ）を持ち、これらを経由してレジスタファイル（WREGF～ZREGF）にアクセスする。

（ｂ）マイクロプログラムのフィールド構成

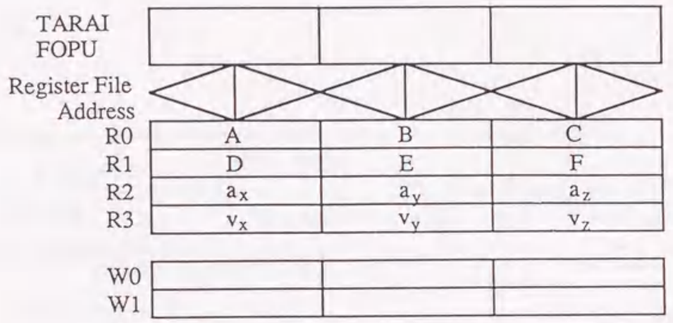
マイクロプログラムのフィールド構成を図４－９に示す。図は機能別にフィールドを示している。マイクロプログラムは、一語９０ビットであり１６ＫＷの容量を持つ。主なフィールドの意味を以下に説明する。

TW,DWはそれぞれ１ビットのフィールドであり、TARAIユニットとDMA転送の実行終了待制御を行う。すなわち、これらのビットがたっていると該当する動作が終了するまでＭＰは一時停止する。

ALU CTLはALUの制御フィールドである。ALUはAMD2901相当品を使用している。REG ADRには図４－５のLocal REGFのアドレスを与える。A,Bのサブフィールドには ALUの左右の入力アドレスを指定する。EXT SRCにはMPY,IW～IZ,ページレジスタなどの外部資源を指定する。

Next Address Controlはシーケンサの制御フィールドである。

TARAI Controlは２つのサブフィールドからなる。TEはＴＡＲＡＩ起動ビットである。このビットがたっていると、Entry Addressに指定されたアドレスからＴＡＲＡＩのマイクロプログラムの実行が開始される。ＴＡＲＡＩを起動したあと、ＭＰは自身のマイク



Traced Data
(Contents of Register Files at Operand 3)

Stp	OPC	OPRND 1	NET	OPRND 2	NET	OPRND 3	X	Y	Z
1	×	R0	↑	R2	↑	W0	$A \cdot a_x$	$B \cdot a_y$	$C \cdot a_z$
2	×	W0	↑	R2	↑	W3	$A \cdot a_x^2$	$B \cdot a_y^2$	$C \cdot a_z^2$
3	×	R2	↖	R1	↑	W1	$D \cdot a_y$	$E \cdot a_z$	$F \cdot a_x$
4	×	R2	↗	W1	↑	W4	$D \cdot a_y \cdot a_z$	$E \cdot a_z \cdot a_x$	$F \cdot a_x \cdot a_y$
5	+	W4	↑	W4	↑	W4	$2D \cdot a_y \cdot a_z$	$2E \cdot a_z \cdot a_x$	$2F \cdot a_x \cdot a_y$
6	×	W0	↑	R3	↑	W5	$A \cdot a_x \cdot v_x$	$B \cdot a_y \cdot v_y$	$C \cdot a_z \cdot v_z$
7	×	R3	↗	W1	↑	W6	$D \cdot a_y \cdot v_z$	$E \cdot a_z \cdot v_x$	$F \cdot a_x \cdot v_y$
8	×	R3	↖	R1	↑	W2	$D \cdot v_y$	$E \cdot v_z$	$F \cdot v_x$
9	×	R2	↗	W2	↑	W7	$D \cdot a_z \cdot v_y$	$E \cdot a_x \cdot v_z$	$F \cdot a_y \cdot v_x$
10	×	R0	↑	R3	↑	W8	$A \cdot v_x$	$B \cdot v_y$	$C \cdot v_z$
11	×	W8	↑	R3	↑	W8	$A \cdot v_x^2$	$B \cdot v_y^2$	$C \cdot v_z^2$

図４－８　マイクロプログラム実行例

TW	DW	ALU CTL	REG ADR		EXT SRC	Next Address Control
			A	B		
1	1	13	5	5	6	31

TARAI Control		DMA Control		Register File Read/Write Control	Miscellaneous
TE	Entry Address	DE	DMA CMD		
1	13	1	3	8	2

(注)
上下２段のフィールドは
上段：フィールド名
下段：サブフィールド名
数字はフィールドの長さ

図４－９　ＭＰマイクロプログラムのフィールド構成

ロプログラムを継続して実行できるので、ＭＰとＴＡＲＡＩの並列動作が可能となる。

DMA Controlは２つのサブフィールドからなる。DEはDMA起動ビットである。このビットがたっていると、DMA CMDで指定されたDMA転送が開始される。ＤＭＡ転送を起動したあと、ＭＰは自身のマイクロプログラムを継続して実行できるので、ＭＰとＤＭＡ転送の並列動作が可能となる。

Register File Read/Write ControlはＭＰからレジスタファイル（WREGF～ZREGF）をアクセスするための制御フィールドである。

この他に、図には示していないが、３２ビットの定数をマイクロ命令の中で使用できる。マイクロ命令の長さを短くするため定数フィールドはフィールドエンコードされている[萩原,1977]。

（３）ＤＭＡユニット

ＴＡＲＡＩ／ＭＰのレジスタファイルとＤＢＭ間のＤＭＡ転送、ホストプロセッサとＤＢＭおよびレジスタファイル間のＤＭＡ転送を行なうハードウェアユニットである。図４－５のＤＭＡと記した箱が相当する。

（４）ＤＢＭユニット

ＤＢＭには画像生成のための各種データ－－物体の定義データ、テキスチャマッピング用データ[Blinn,1976]、バンプマッピング用データ[Blinn,1978]等－－を格納する。これらのデータは相当な量になるので、ＳＩＧＨＴでは１ＭＷ（３２ビット／Ｗ）４バンク（Ｗ～Ｚ）のメモリを用意する（ＳＩＧＨＴは、高速処理を目的とするので、ホストプロセッサとＰＥ間のデータ転送を極力避ける方針をとる）。バンクＸ～ＺにはＴＡＲＡＩで使用するデータ、バンクＷにはＭＰで使用するデータを格納する。

４．３．３　関数テーブルの構成

光線追跡法では、平方根、正弦、余弦、等の関数を多用する。これらの関数計算を高速化するために、ＳＩＧＨＴでは、テーブルを利用した近似計算を行なう。以下に、

その手法を示す。

（1）ＳＩＮおよびＣＯＳ関数

いずれも同様に計算できるので、以下ではSIN 関数について述べる。

０≦Ｘ≦２πなるＸに対してSIN(X)を計算する。０～２πをｎ等分し、各等分点XiにおけるSIN(Xi) の値をテーブルで与えておく。区間内の任意の点におけるSIN(X)の値は補間して求める。一次式および二次式による補間法を示す。

（ａ）一次式による補間

補間式をｙ＝ａ×ｘ＋ｂとする。隣り合う２点Xi,Xj（ Xj>Xi ）の関数値をそれぞれFi, Fjとすると、補間式の係数ａ，ｂは次式で与えられる。

$$a = \frac{Fj}{Xj} - \frac{Fi}{Xi} \qquad (4-4)$$

$$b = \frac{Xj \times Fi}{Xj} - \frac{Xi \times Fj}{Xi} \qquad (4-5)$$

（ｂ）二次式による補間

補間式をｙ＝ａ×ｘ×ｘ＋ｂ×ｘ＋ｃとする。相隣り合う３点 Xi,Xj,Xk（ Xk>Xj>Xi ）の関数値をそれぞれ Fi, Fj, Fk とすると、補間式の係数 a, b, cは次式で与えられる。

$$a = \frac{1}{2 \times \Delta \times \Delta} \times A \qquad (4-6)$$

$$b = \frac{1}{2 \times \Delta \times \Delta} \times B \qquad (4-7)$$

$$c = \frac{1}{2 \times \Delta \times \Delta} \times C \qquad (4-8)$$

ここで、

$$\Delta = Xk - Xj = Xj - Xi$$
$$A = Fk - 2 \times Fj + Fi$$
$$B = 2 \times Xi \times A + \Delta \times (Fi - Fk)$$
$$C = Xj^2 \times A + 2 \times \Delta^2 Fi + \Delta \times Xj \times (Fi - Fk)$$
$$= Xj \times B - Xj^2 \times A + 2 \times \Delta^2 \times Fj$$

である。

一次式および二次式で補間した場合、２４ビットの精度（単精度浮動小数点演算の精度）を得るために必要なテーブルの大きさを表４－１に示す。表では、ＲＯＭでテーブルを作成することを考慮して、テーブルの大きさは、２のベキ乗としている。

表４－１　２４ビット精度を保証するために必要なテーブルの大きさ

関　数	SIN/COS	平方根
区　間	$0 \leq x \leq \pi/2$	$1/4 \leq x \leq 1$
１次補間	2048	4096
２次補間	256	512

ＳＩＧＨＴでは一次式による補間を採用する。その理由は、

①計算誤差の累積が少ない。

②計算ステップ数が小さい。

③テーブルサイズが大きくなるが、メモリの集積技術の進歩により十分吸収できる。

である。

（２）平方根

平方根の高速計算を行う方法として、補間法とニュートン法が考えられる。

（ａ）補間法

浮動小数点データを、$y = x \times 2^{2n}$ （ $1/4 \leq x < 1$、$-\infty < n < \infty$ ）と表記するとき、ｙの平方根は、

$$SQRT(y) = SQRT(x) \times 2^{n} \qquad (4-9)$$

と書ける。したがって、$1/4 \leq x < 1$の区間の等分点に対するＳＱＲＴ（ｘ）の値をテーブルで与えておけば、任意の点に対する平方根は、上記ＳＩＮ等と同様の手法で計算できる。

一次式および二次式で補間した場合、２４ビットの精度（単精度浮動小数点演算の精度）を得るために必要なテーブルの大きさを表４－１に示す。ＳＩＧＨＴでは、上記と同様の理由で一次式による補間を採用する。

（ｂ）ニュートン法

ｘの平方根を求めたいとき、ニュートン法では次の反復式を計算する。

$$a(n+1) = 0.5 \times a(n) \times (3 - a(n) \times a(n) \times x) \qquad (4-10)$$
$$n = 1, 2, \ldots$$

これより、$a = \lim_{n \to \infty} a(n)$とすると、ＳＱＲＴ（ｘ）はａｘで与えられる[山内,1972]。初期値ａ（１）は、ＳＱＲＴ（ｘ）に近い値を与える。通常は、ｘをインデックスとするテーブルを用意し、これを引くことにより与える[Weitek,1984]。図４－１０にそのハードウェアイメージを示す。単精度浮動小数点演算のとき、図４－１０の構成で、式（４－１０）の反復回数は２である。一方、反復回数を１に抑えるには、ＲＯＭ＃２の大きさを、８Ｋ×１２とする（入力：指数部１ビット、仮数部１２ビット；出力：仮数部１２ビット）。ＳＩＧＨＴでは、後者を採用し高速化を図る。

ＳＩＧＨＴでは、これら両手法を提供する。両手法の特徴は、補間法では、１つの平方根を３個の演算器を使って並列計算して求めることであり、ニュートン法では、３つの平方根が同時計算できることである。演算ステップ数は補間法がニュートン法より少ないから、１つの平方根が必要なときは補間法を使うのが得策である。

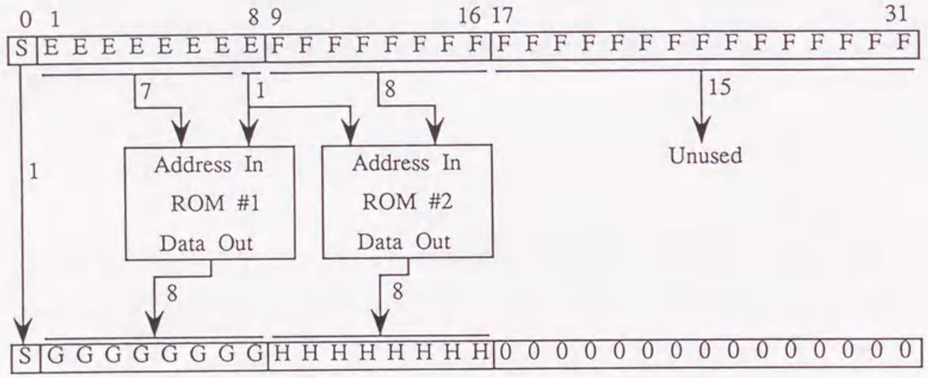


図４－１０　ニュートン法による平方根計算の初期値を与えるテーブル構成
(浮動小数点データ)

４．４　ＳＩＧＨＴ上の光線追跡アルゴリズム

この節では、ＰＥの各ユニットを並行に動作させる光線追跡アルゴリズムについて述べる。ここで述べるアルゴリズムは、ＳＩＧＨＴ用にシンタックスを拡張したＣ言語（ＳＩＧＨＴ／Ｃ）を用いて記述する（ＳＩＧＨＴ／Ｃについては４．６節を参照されたい）。

４．４．１　データ構造

ＳＩＧＨＴでは物体記述に柔軟性の高いＣＳＧ（Constructive Solid Geometry）モデルを採用する。ＣＳＧでは、基本形状（プリミティブ）間の論理演算により物体を定義する。データ構造は木構造（ＣＳＧ木）となる。光線追跡計算の高速化のために、ＣＳＧ木の中間ノードには、必要に応じてバウンディングボリューム（ＢＶ）が設定される。物体定義の一例は図２－６に示されている。ＣＳＧ木のノードのデータ構造を図４－１１に示す。図４－１１で、予約語MP,TARAIはターゲットユニットを指示する。また、vectorは３次元ベクトルを表すデータタイプである。図４－１１に示すように、ノードのデータ構造は、ＭＰで処理する部分（木構造を表すデータおよび属性データ）とＴＡＲＡＩで処理する部分（形状の方程式、位置データ等）から成り、それぞれ対応するＤＢＭに格納される。構造体node_MPのメンバdbm_addrには、構造体node_MPとnode_TARAIのＤＢＭ内格納アドレスが入る。

ＣＳＧ木をサーチする際、ＭＰの構造体データnode_MPは頻繁にアクセスするのでできる限りレジスタファイルに常駐させる戦略をとる。一方、ＴＡＲＡＩの構造体データnode_TARAIは後述のようにパイプライン処理できるので必要の都度ＤＢＭから読みだす。この理由からＳＩＧＨＴでは、ＭＰのレジスタファイルは容量を大きくし、ＴＡＲＡＩのレジスタファイルは容量を小さくしている。

```
typedef MP node_MP struct {   /*ＭＰの構造体*/
    int         x_flag;     /*交点フラグ*/
    int         children;   /*子供の数*/
    int         node_type;  /*AND/OR ノード*/
    int         leaf;       /*葉／中間ノード*/
    int         attr;       /*葉ノードのとき表面属性、*/
                            /*中間ノードのときBV設定フラグ*/
    int         shape;      /*BVまたはプリミティブの形状*/
    int         reg_addr;   /*レジスタファイルのロードアドレス*/
                            /*ＭＰによってセットされる*/
    int         dbm_addr;   /*ＤＢＭの格納アドレス*/
    node        *child;     /*子ノードへのポインタ*/
    node        *brother;   /*弟ノードへのポインタ*/
};
```

(a)ＣＳＧ木のノードのデータ構造（ＭＰユニット）

```
typedef TARAI node_TARAI struct {
    vector      coef1;      /*形状方程式の係数*/
    vector      coef2;      /*形状方程式の係数*/
    vector      position;   /*位置座標*/
    vector      rotation;   /*回転角*/
    vector      color;      /*(R,G,B)*/
    vector      misc1;      /*（減衰率、屈折率、）*/
}
```

(b)ＣＳＧ木のノードのデータ構造（ＴＡＲＡＩユニット）

図４－１１　ＣによるＣＳＧ木のノードのデータ構造の記述

４．４．２　ＰＥ上の光線追跡アルゴリズム

ＳＩＧＨＴのＰＥ上の光線追跡アルゴリズムは、各ユニット（ＴＡＲＡＩ、ＭＰ、ＤＢＭ）をできる限り並行動作させるよう工夫している。図４－１２に概略アルゴリズムを示す。これはＭＰで実行される。図において、反射光と屈折光の追跡を行う関数の本体は、またこの図のような構成になる。

図４－１２の関数search_CSGはＣＳＧ木を深さ方向優先でサーチし、視点に最も近い物体を見つける関数（交差判定関数）である。この関数の本体を図４－１３(a)に示す。その処理は、まず、関数intersectionにより、着目ノード(ptr)の全ての子ノードに対応するＢＶあるいはプリミティブとの交点計算を行い、交点の有無を交点フラグ（図４－１１参照）に記録する。たとえば、ルートノードに対しては、図４－１３(b)の①の矢印の順に処理する。ついで、サーチ順に各ノードの交点フラグをチェックする。フラグが立っていれば、そのノードのすべての子ノードに対応するＢＶ／プリミティブとの交点計算を行い、交点の有無を記録する。例えば、図４－１３(b)においてａのノードの交点フラグが立っていれば、その子供を②の矢印の順に処理する。フラグが立ってなければ、そのノードの子孫に対する交点計算は不要となる。このような手順で交点計算を進める。この過程で、ＢＶが設定されてない中間ノードは、交点ありとして処理を進める。また、フラグの立っている葉ノード（プリミティブ）は、そのプリミティブを視点に最も近いプリミティブの候補とする。この候補とすでに選ばれた候補を比較し、視点に近い候補を残す。このようにして、視点に最も近い候補が見つけられる。

図４－１３(a)の関数intersectionの本体では、node_TARAIのデータをＤＢＭからＴＡＲＡＩのレジスタファイルへ転送し、交点計算を行う。この処理は、図４－１４に示すようなパイプライン処理である。すなわち、ＴＡＲＡＩが交点計算をしている間に、ＭＰは次の交点計算の対象となるデータのＤＢＭ上のアドレスを計算し、ＤＭＡ転送によりＤＢＭから転送して準備する。図４－１４のＤＭＡとＴＡＲＡＩの欄で実線部分はそのユニットが走行していることを示す。このようなパイプライン処理によって、ユニットを並行動作させ、交点計算の高速化を図る。

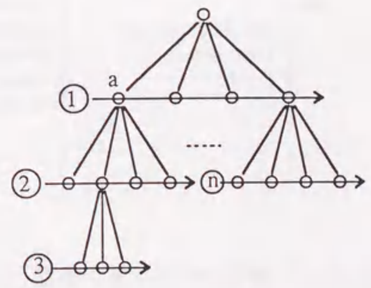
```
ray_trace(i,j) { /*画素(i,j)に対する光線追跡,MPのプログラム*/
  node_MP  *primitive;
  vector   *luminance, *luminance1, *r;

  ray_equation(i,j);     /*光線の方程式を決定する*/
  primitive= search_CSG(ROOT_NODE); /*CSG木をサーチして視点に最も近い*/
                                    /*プリミティブを見つける*/
  if(primitive==NULL) {
    output(back_ground_color);     /*プリミティブがなければ背景光を出力*/
  } else {
    switch(primitive->attr) { /*交差物体の表面属性が*/
    case DIFFUSE:  /*拡散反射面*/
        luminance= shading(primitive);  /*輝度計算*/
        break;
    case MIRROR:  /*鏡面*/
        luminance= trace_reflected_ray();  /*反射光を追跡して輝度を*/
        break;                             /*求める*/
    case TRANSPARENT:  /*透明体*/
        luminance= trace_reflected_ray();  /*反射光を追跡して輝度を求める*/
        luminance1= trace_refracted_ray(); /*透過光を追跡して輝度を求める*/
        luminace= addition(luminance,luminance1,r);/*反射光と透過光の*/
        break;                             /*輝度の重み付き加算、rは反射係数*/
    }
    output(luminance);  /*輝度値を出力*/
  }
}
```

図４－１２　ＳＩＧＨＴ上の光線追跡アルゴリズムの概略


```
search_CSG(ptr) { /*ＣＳＧ木のサーチ,ＭＰのプログラム*/
  intersection(ptr); /*子ノードに対する交点計算*/
  ptr_brother= ptr->child;
  while(ptr_brother!=NULL) { /*長兄から末弟の順に処理*/
    if(ptr_brother->x_flag) { /*交点ありなら*/
      if(ptr_brother->leaf!=LEAF) { /*葉ノードでなければ*/
        primitive= search_CSG(ptr_brother); /*子孫をサーチ*/
      } else { /*葉ノードなら*/
        primitive= ptr_brother;  /*視点に最も近いプリミティブの候補*/
      }
    }
    select_closer_primitive(); /*すでに選ばれたプリミティブと*/
        /*上で選んだプリミティブを比較し視点に近い方を選択し記憶する*/
    ptr_brother= ptr_brother->brother;
  }
  return(closest_primitive); /*上で記憶したプリミティブを返す*/
}
```

(a)アルゴリズムの概略



(b)処理の流れ

図４－１３　ＣＳＧ木のサーチ

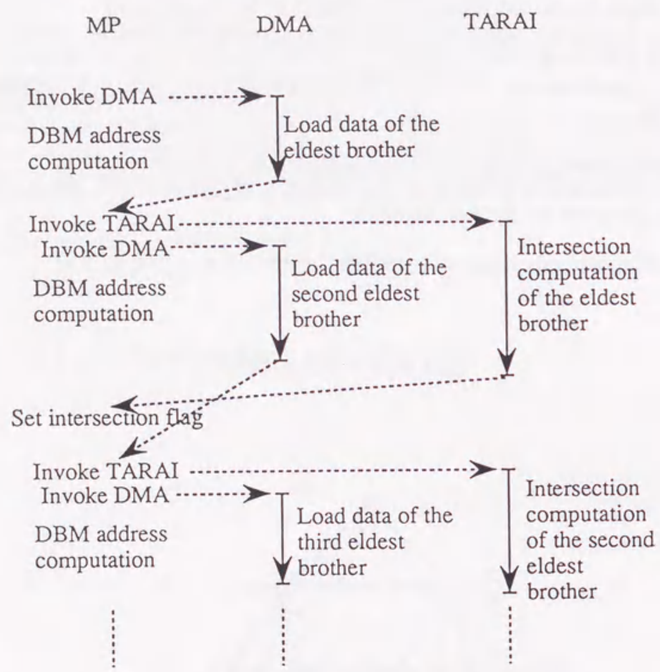


図４－１４ 交点計算のパイプライン処理

４．４．３ マルチプロセッサ上の光線追跡アルゴリズム

ＳＩＧＨＴでは、画像を生成する領域をいくつかの小領域に分割し、各小領域の処理をＰＥに割り付ける。各ＰＥは４．４．２節で述べたアルゴリズムを実行する。小領域の割付は静的とする。すなわち、各小領域の受持ちＰＥをあらかじめ定めておく。光線追跡法は、画素毎に独立に計算できるから、ＰＥ間の通信は基本的に不要である。したがって、同期に伴うオーバヘッドはない。負荷のアンバランスは小領域の計算時間の大きさの違いによって生じる。このアンバランスを低減するには、小領域をできるだけ小さくし、１つのＰＥに多数の小領域を割り当てるのがよい。もちろん各ＰＥにほぼ同数の小領域を割り当てる。こうすることにより、個々の小領域の処理時間はばらついても全体の処理時間は大数の法則により一定値に近づくであろう。その結果ＰＥ間の処理時間のばらつきは少なくなる。

４．５　ＳＩＧＨＴの性能解析・評価

この節では、ＳＩＧＨＴのＰＥアーキテクチャの有効性を検証するため、以下の点に関して性能解析・評価を行う。

（１）ＴＡＲＡＩの演算並列度
（２）各ユニットの並行動作
（３）ＴＡＲＡＩ起動のオーバヘッド
（４）ＶＡＸ１１／７８０との性能比較

上記（１）～（３）の検証は、マイクロプログラムレベルでＳＩＧＨＴを模擬するシミュレータ上で行った（付録４－２参照）。また、（４）の検証は、ＳＩＧＨＴ試作機およびＶＡＸ上で行った。

４．５．１　実験方法

（１）試作機上での実験

（ＳＩＧＨＴ試作機）写真４－１に試作したＳＩＧＨＴプロトタイプ機（１ＰＥ構成）の前景を示す。架には、上から順に、フレームバッファ、ＰＥ、デバグ用マイクロプロセッサ、電源が収納されている。ＰＥは１１枚のボードで構成される。そのうちの主なボードの写真を写真４－２～写真４－７に示す。

（実験方法）４．４．２節で述べたアルゴリズムをマイクロプログラムで記述し、試作機上で走行させ、実行時間を測定する。同様にＶＡＸ１１／７８０（浮動小数点演算器付き）上で光線追跡プログラムを走行させ実行時間を測定する。尚、ＶＡＸ上のプログラムはＦＯＲＴＲＡＮで記述している。

（実験項目）商用機（ＶＡＸ）との処理時間比較

（物体定義データ）格子状に並べた球。表面属性として、拡散反射、鏡面反射。球の数は１６、２５、３６。写真４－８、４－９に生成例を示す。生成画像の画素数は512×512である。

（２）シミュレータによる実験

（実験方法）４．４．２節で述べたアルゴリズムをマイクロプログラムで記述し、それをシミュレータ上で走行させる。シミュレータの統計情報収集機能を用いて各種統計データを収集する。統計データは、図４－１２の関数本体で、output関数を除いた部分について収集する。

（実験項目）ＳＩＧＨＴの基本特性を明らかにするために、以下の項目について実験する。

①ＴＡＲＡＩ演算器の演算並列度：プリミティブの形状、表面属性、ＣＳＧ木の構造をパラメータとして、ＴＡＲＡＩの演算並列度を調べる。
②ユニットの並行動作：プリミティブの形状、ＣＳＧ木の構造をパラメータとして、ユニットの並行動作を調べる。

（実験に用いた物体定義データ）以下の物体定義データを使用する。

①ＣＳＧ木：　図４－１５に示す５種類のＣＳＧ木を使用する。図中●の中間ノードにはＢＶを設定する。各ノードはすべてＯＲノードである。
②プリミティブの形状：　球、楕円体（一般形）、直方体
③プリミティブの表面属性：　拡散反射、鏡面反射、透明

シミュレーション実験は、画像の大きさを16×16画素にして行った。一例として、図４－１５のflat16のCSGデータを用いて、試作機上で生成すると写真４－８に示した画像が得られる。

写真４－１　ＳＩＧＨＴプロトタイプ機前景

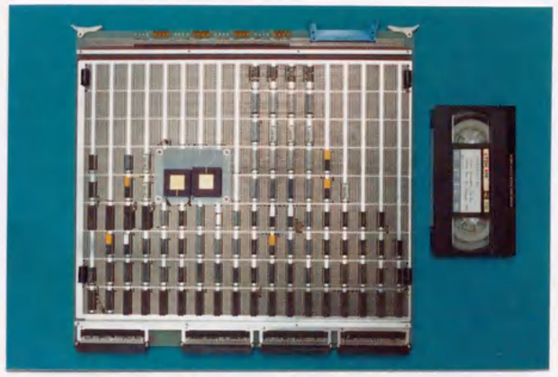


写真４－２　ＭＰの演算部

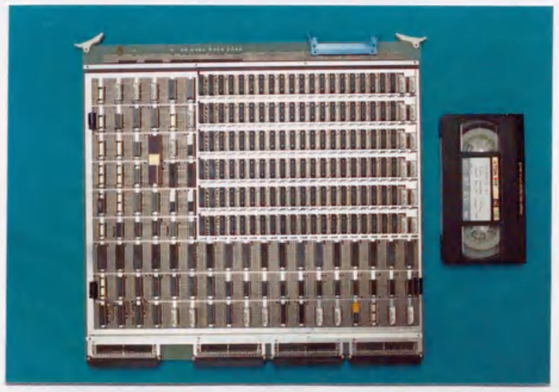


写真４－３　ＭＰのシーケンサ部

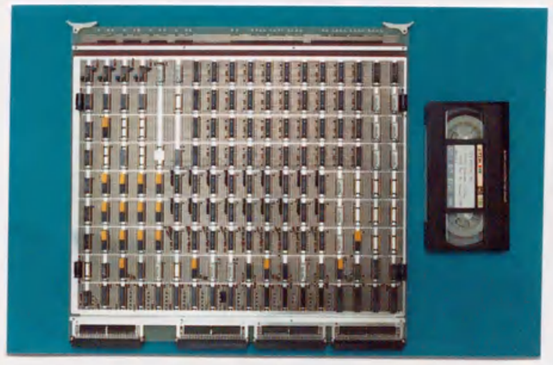


写真４－４　レジスタファイル部（ＭＰおよびＴＡＲＡＩ）

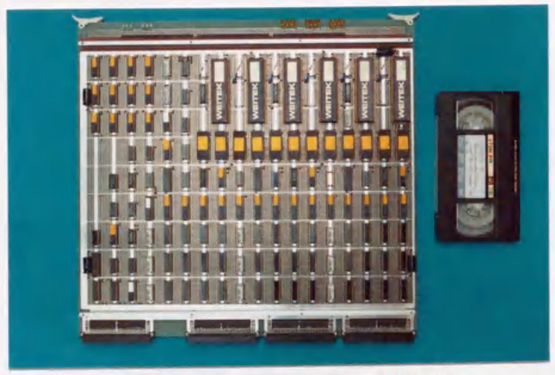


写真４－５　ＴＡＲＡＩの演算部

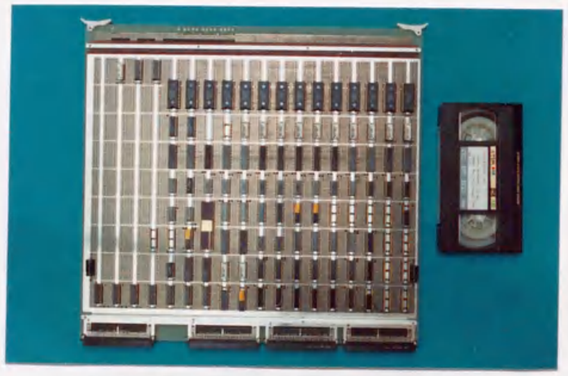


写真４－６　ＴＡＲＡＩのシーケンサ部

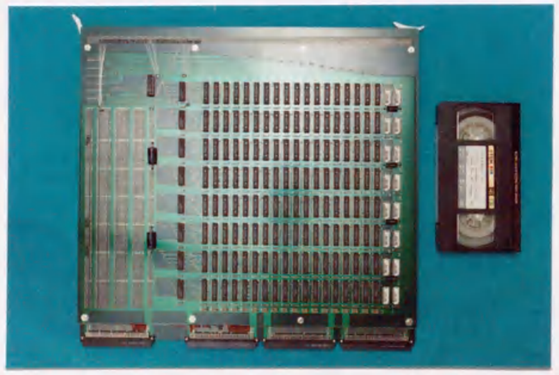


写真４－７　ＤＢＭ　（１バンク分）

写真４－８

写真４－９

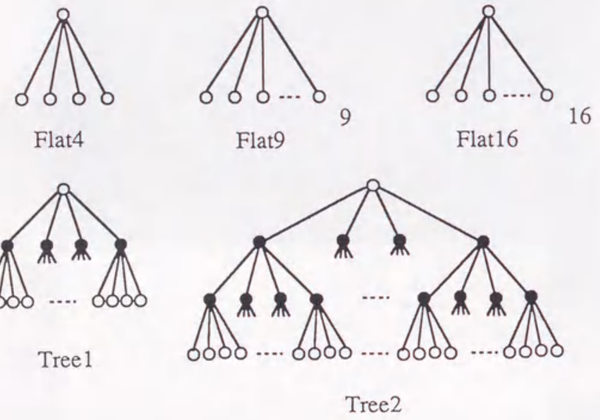


図４－１５ 実験に使用したCSG木

４．５．２ 実験結果と考察

（１）ＴＡＲＡＩの演算並列度

（ａ）演算並列度

ＴＡＲＡＩ動作時において、プログラムの各ステップでＴＡＲＡＩの演算器(FOPU)が何個稼働したかを調べた。結果を表４－２に示す。表４－２は次式で定義する演算並列度を示す（脚注）。

$$\text{演算並列度} = \frac{\sum_{k=1}^{3} k\text{-FOPUの有効実行ステップ数}}{\text{全実行ステップ数}} \qquad (4-11)$$

上式で有効実行ステップ数は、ＮＯＰ命令（No operation命令）以外の命令を実行したステップ数である。演算並列度の最大値は３である。

実験の結果、表４－２に示すように、2.20～2.62と高い演算並列度が得られた。また、演算並列度は、プリミティブの形状に依存し、表面属性やＣＳＧ木の構造にはほとんど依存しないことがわかる。

（ｂ）演算並列度３で実行可能なステップ数の割合

演算器の稼働内訳を、一例について図４－１６に示す。この例では、３個の演算器が稼働したステップ数は全ＴＡＲＡＩ実行ステップ数の85.0%である。２個および１個の演算器が稼働したステップ数はそれぞれ3.1%、1.0%である。また、全体の10.9%は演算器が稼働しなかった。この部分は、ＪＵＭＰ命令などの非数値演算を実行した部分である。このように、３個の演算器が稼働する割合が高いことがわかる。他の場合も同様の傾向を示す。この結果は、光線追跡法では３次元ベクトル演算の割合が高いことを示しており、ＴＡＲＡＩの有効性が実証された。

（脚注）ここで定義する演算並列度は５章で議論する演算並列度と同じものである。

表４－２　演算並列度

CSG木	球			楕円体			直方体		
	拡散	鏡面	透明	拡散	鏡面	透明	拡散	鏡面	透明
Flat4	2.24	2.28	2.22	2.36	2.40	2.34	2.52	2.61	2.54
Flat9	2.24	2.26	2.22	2.40	2.42	2.39	2.56	2.62	2.57
Flat16	2.24	2.25	2.23	2.42	2.43	2.40	2.58	2.62	2.59
Tree1	2.23	2.25	2.22	2.31	2.32	2.28	2.56	2.62	2.55
Tree2	2.21	2.23	2.20	2.28	2.28	2.26	2.58	2.61	2.56

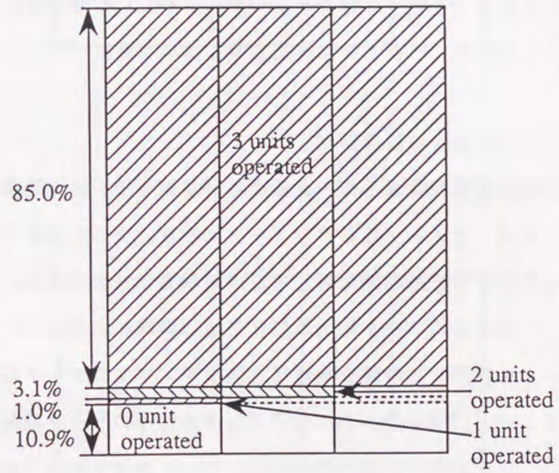


図４－１６　演算器稼働個数

（２）ユニットの並行動作

ＴＡＲＡＩ、ＭＰ、ＤＭＡ各ユニットの動的実行ステップ数を計測し、並行動作を解析した。

（ａ）ＴＡＲＡＩとＭＰの並行動作

ＴＡＲＡＩとＭＰが並行動作する時間の全実行時間に対する割合は14%～27%と小さかった（表４－３）。この並行動作による光線追跡計算の高速化は１～２割程度である。この割合は、主として、子ノード数に依存して変化し、他のパラメータによる変化は少ない。これは、ＴＡＲＡＩとＭＰが並行動作する部分が、図４－１４におけるＴＡＲＡＩの交点計算とＭＰのＤＢＭアドレス計算であることによる。この並行動作を採用するか否かは性能と計算機コストのトレードオフで決定することになる。

（ｂ）ＴＡＲＡＩとＤＭＡの並行動作

この並行動作は交点計算とデータ転送のパイプライン処理効率にかかわる（図４－１４）。表４－４は、子ノード数をパラメータにとり、交点計算とデータ転送時間が並行動作した時間の全データ転送時間に対する割合を示したものである。楕円体と直方体では、１つの子ノードあたりのデータ転送時間が交点計算時間より短い。したがって、最初の子ノードのデータを転送する時間を除けば、他の子ノードのデータ転送は交点計算と完全に並行動作する。一方、球では、１つの子ノードあたりのデータ転送時間が交点計算時間より大きくなり、ＴＡＲＡＩに遊び時間が生じる。このようになった理由は、プロトタイプ機では、ハードウェア量を抑えるため、ＤＭＡ転送１回あたりのデータ転送語数を１６語に固定しているからである。１個のプリミティブのデータ量は１６語よりも小さいので、データ転送語数を可変にすることにより球の場合もデータ転送が交点計算と完全に並行動作するように改良できる。この結果、交点計算とデータ転送のパイプライン処理が効率よく動作することが示された。このことから、レジスタファイルとＤＢＭの２階層メモリ構成により、大容量高速メモリを構成できる。

表４－３　ＴＡＲＡＩとＭＰの並行動作時間の
全実行時間にたいする割合(%)

子ノードの数	4	9	１６
球	16.1	22.4	26.6
楕円体	14.1	19.2	21.7
直方体	13.6	17.8	20.4

表４－４　交点計算に隠れるデータ転送時間の割合（%）

子ノードの数	4	9	１６
球	64.4	75.0	80.6
楕円体	75.0	88.9	93.8
直方体	75.0	88.9	93.8

（3）ＴＡＲＡＩ演算器の起動オーバヘッド

図４－７に示すような時分割バスを採用したため、ＭＰがＴＡＲＡＩを起動する際にオーバヘッドがある。ＭＰは任意の時刻にＴＡＲＡＩを起動できるが、その時刻がＴＡＲＡＩサイクルの途中であると、ＴＡＲＡＩはその次のサイクルから動作を開始する。したがって、ＴＡＲＡＩを起動した時刻から、そのＴＡＲＡＩサイクルの最後までがオーバヘッドとなる。起動１回あたりのＴＡＲＡＩ実行ステップ数が大きければ、このオーバヘッドは無視できるが、小さい場合には性能の劣化につながる。

起動１回あたりのＴＡＲＡＩ平均実行ステップ数の測定結果を表４－５に示す。この表から、ステップ数は、物体形状とＣＳＧ木の構造に依存して変わり、表面属性にはあまり依存しないことがわかる。これは、光線追跡計算の主要部が交点計算であり、交点計算の回数はＣＳＧ木構造に依存することから妥当な結果である。この結果から、ＴＡＲＡＩ起動のオーバヘッドの（起動１回あたりの）実行時間に対する割合を算出する。ＴＡＲＡＩの起動時刻はランダムとすると、オーバヘッドは平均３クロックとなる。一方、ＴＡＲＡＩの１実行ステップは６クロックであるから、その割合は、

$$\frac{3}{\text{ＴＡＲＡＩ平均実行ステップ数} \times 6} \qquad (4-12)$$

となる。この値は、球の場合で2.5%～3.0%、楕円体の場合で1.8%～2.3%、直方体の場合で1.6%～2.4%となる。

このオーバヘッドは、小さな値ではあるが、無視できる程に小さなものではない。このような値になったのはＴＡＲＡＩ起動１回あたりの実行ステップ数が小さいためである。このオーバヘッドを減らすには、ＴＡＲＡＩの実行サイクル数を減らす工夫をしなければならない。

表４－５　ＴＡＲＡＩ起動１回あたりの平均実行ステップ数

	flat4	flat9	flat16	tree1	tree2
球	16.4～18.5	19.9～18.4	17.4～18.2	19.0～19.7	17.9～19.1
楕円体	22.0～24.7	24.9～26.6	26.6～27.5	21.7～22.0	21.6～22.0
直方体	21.0～25.5	24.1～28.6	27.3～31.1	24.7～27.2	22.9～26.0

注）・物体形状とＣＳＧ木の構造の組合せで表示する。
・各欄の値は３つの表面属性に対してそれぞれの平均実行ステップ数を求め、その最大値と最小値を示す。

（４）ＶＡＸとの速度比較

ＰＥの並列処理性能を前節までに詳しく解析し、光線追跡法の演算レベル並列処理が提案したＰＥ構成で効率よくできることを示した。本節では、ＳＩＧＨＴプロトタイプ機の実性能を浮動小数点演算器の付いたＶＡＸ１１／７８０と比較して示す。一般に、計算機間の性能比較は、使用している素子技術、ソフトウェア環境などの違いにより、客観的に行なうことが難しい。ここでは、それぞれの計算機上で、光線追跡プログラムを走行させ、計算時間を実測して、それを単純に比較する。

４．５．１節に示した方法で、ＳＩＧＨＴ試作機とＶＡＸ双方で光線追跡プログラムを走行させ、処理時間を測定した。結果を表４－６に示す。表からＳＩＧＨＴの１ＰＥの性能はＶＡＸの約１０倍であることがわかる。

ＳＩＧＨＴとＶＡＸで使用している素子技術はほぼ同等である。また、ＳＩＧＨＴの並列処理技術はＴＡＲＡＩ演算器とユニット間の並行動作である。解析の結果、ＭＰとＴＡＲＡＩの並行動作による高速化効果は２０％前後であり、また、データ転送と演算の並行動作はＶＡＸでも同様の処理を行っていることから、ＳＩＧＨＴの方式的優位性は主としてＴＡＲＡＩによる並列演算からくる。そして、その値は３倍程度である。しかし

表４－６　処理時間比

（ＳＩＧＨＴ１ＰＥ／ＶＡＸ）

		球の数		
		１６	２５	３６
表面属性	拡散反射	9.67	10.0	10.3
	鏡面反射	9.35	9.81	10.2

ながら、実測値は１０倍となった。その理由を以下に考察する。

①マシンサイクルの違い：ＳＩＧＨＴのマシンサイクルは１２５ナノ秒であり、ＶＡＸのそれは２００ナノ秒である。この差異はＭＰに現れる。ＴＡＲＡＩの１個の演算器とＶＡＸの浮動小数点演算器の実行速度は同程度である。

②命令の実行レベルの違い：ＳＩＧＨＴはマイクロプログラムレベルの処理であるのに対し、ＶＡＸは機械語レベルの処理である。ＶＡＸの基本的な機械語命令は２～３マイクロ命令で解釈実行される。この差異はＭＰに現れる。ＴＡＲＡＩとＶＡＸの浮動小数点演算器は同程度の命令セットとなっている。

③言語の違い：ＳＩＧＨＴはマイクロプログラムでプログラムを記述しているのに対し、ＶＡＸはＦＯＲＴＲＡＮで記述している。ＶＡＸ　ＦＯＲＴＲＡＮのコンパイラのコード効率は、アセンブリ言語によるコードの２倍程度と言われている。

４．６　ＳＩＧＨＴ用高水準言語
ＳＩＧＨＴ／Ｃ

この節では、ＳＩＧＨＴのソフトウェア開発を促進するために導入した、ＳＩＧＨＴのマイクロコードをターゲットとする、Ｃ言語（ＳＩＧＨＴ／Ｃ）について、その設計思想を述べる。

４．６．１　設計思想

ＳＩＧＨＴアーキテクチャに適合したコードを生成するため、次のような設計思想の下にＳＩＧＨＴ／Ｃコンパイラの開発を進める。

・原則としてＣ言語の標準仕様[Kernighan,1978]に従うが、コンパイラの負担を少なくするためＳＩＧＨＴアーキテクチャに依存するシンタックスの導入を許すこととする。

・Functionの割当属性：FunctionのＭＰ／ＴＡＲＡＩユニットへの割当を指定するため、予約語ＭＰ、ＴＡＲＡＩを導入する。

例

```
MP func_a(...)      /*MPのfunction*/
TARAI func_b(...)   /*TARAI のfunction*/
```

割当属性の省略時解釈はＭＰユニットとする。

main function はＭＰユニットに割り当てる。

・データ構造：ＴＡＲＡＩユニットのリソース（３個の演算器）割付処理の簡単化を図るため以下のデータ構造を導入する。

vector　（三次元ベクトル）

matrix　（３×３マトリクス）

vectorはすべて行ベクトルとして扱う。

vector要素へのアクセスは、一次元配列と同じ記法とする。

例

```
v[0]    ; vector vのx要素
v[1]    ; vector vのy要素
v[2]    ; vector vのz要素
```

添字は３を法として計算する。また、添字はコンパイル時に値が確定しなければならない。

matrix要素へのアクセスは、二次元配列と同じ記法とする。

・記憶域の割付：ＭＰユニットは３レベル（高速レジスタ、レジスタファイル、ＤＢＭ）、ＴＡＲＡＩユニットは２レベル（レジスタファイル、ＤＢＭ）の記憶階層を持つ。各記憶階層は次のように使用する。

ＭＰユニット：

高速レジスタ：register/auto 変数の割付。

レジスタファイル：static/auto/register変数の割付およびstack area。

ＤＢＭ：heap area または、ユーザ管理エリア。

ＴＡＲＡＩユニット：

レジスタファイル：static/auto/register変数の割付およびstack area。

ＤＢＭ：heap area または、ユーザ管理エリア。

ＤＢＭはmalloc,mfree等のメモリ管理ルーチンによる管理を基本とする。しかし、ＤＢＭ－ＴＡＲＡＩ／ＭＰ間のデータ転送とＴＡＲＡＩ／ＭＰ演算の並列実行機能を有効に活用するためユーザーがＤＢＭを陽に管理することを許す。

・単精度浮動小数点データの内部演算は単精度演算とする。

・TARAI functionにおける整数演算は単精度浮動小数点演算(IEEE format)で行なう。従って、整数の演算精度は23bitとなる。

・同期メカニズム、ＰＥ間通信ユーティリティをサポートする。

４．６．２　プログラム例

以下に交点計算のプログラム例を示す。光線（パラメータ表示でｒ＝ａｔ＋ｖ）と物体（２次曲面を例とするとｒＭ'ｒ＝１）の交点は、

$$aM'at^2+2aM'vt+vM'v=1$$

を解くことにより与えられる。よって、プログラムは以下のように記述できる。

```
TARAI vector cross()  /* ２次方程式の係数を計算*/
{
  vector coef;        /*coef=(aM'a,2aM'v,vM'v-1)*/
  vector    a;        /*光線の方向余弦*/
  vector    v;        /*視点の位置ベクトル*/
  matrix    m;        /*曲面の係数マトリクス*/
  coef[0]= inner( a, prodmv( m, a ) );
  coef[1]= 2.0*( inner( a, prodmv( m, v ) ) );
  coef[2]= inner( v, prodmv( m, v ) ) - 1.0;
  return(coef);
}
```

上記プログラムにおいて、inner,prodmvはそれぞれ、内積、マトリクスーベクトル積を計算する関数である。

このようにベクトル演算はコンパクトに記述することができるので、コンパイラはＴＡＲＡＩリソースを有効に活用した効率の良いコードを生成することが可能となる。また、２次方程式の係数をベクトルデータとして扱うことにより、メモリエリアの有効利用が図れる。

４．６．３　処理系

上記設計思想に基づきＳＩＧＨＴ／Ｃの処理系（コンパイラ）を作成した。この処理系には現在のところオプティマイザをインプリメントしていない。処理系はＣ言語で記述している。規模は約１２０００行である。

いくつかのプログラムに対してこの処理系を適用した結果、出力コードのステップ数は人間が直接書き下したコードのステップ数の約１０倍であった。このコードに最適化ルールを適用した結果では、コード効率を約３倍にまで引き下げることができた。オプティマイザのインプリメントは今後の課題である。

４．７　まとめ

光線追跡法の並列処理をねらった計算機ＳＩＧＨＴのアーキテクチャと性能評価結果について述べた。

まず、光線追跡法に内在する並列性を検討し、画素レベルおよび演算レベルの２レベル並列処理ができることを示した。ＳＩＧＨＴは、この２レベル並列処理を実現したアーキテクチャである。ＳＩＧＨＴはマルチプロセッサシステムであり、そのＰＥは４つのユニット（ＴＡＲＡＩ、ＭＰ、ＤＭＡ、ＤＢＭ）から構成される。このうち、ＴＡＲＡＩがＰＥの核ユニットあり、３次元ベクトル演算を並列処理する。また、ＳＩＧＨＴではＰＥの各ユニットを並行動作させ、①アルゴリズムのフローの制御と交点計算等の演算の機能分散処理、②データ転送と演算のパイプライン処理、を行って、並列処理を徹底した。

次に、マイクロプログラムレベルでＰＥを模擬するシミュレータを用いて、ＰＥの性能解析を行った。ＰＥ内の並列動作の解析をした結果、以下のことが示された。

・ＴＡＲＡＩは、１実行ステップあたり2.20～2.62個の演算器が動作した。すなわち、光線追跡計算の大部分は３次元ベクトル演算であり、ＴＡＲＡＩはそれを効率よく実行するアーキテクチャである。

・ＴＡＲＡＩのレジスタファイルとＤＢＭの２階層メモリ構成の採用により大容量高速メモリが構成できる。

また、試作機上で画像生成実験を行い、実行時間を測定した。その結果、

・典型的な商用計算機ＶＡＸ１１／７８０（浮動小数点演算器付き）と比較して、ＳＩＧＨＴは１ＰＥで約１０倍高速に画像を生成できる、

ことが示された。

これらのことから、ＳＩＧＨＴのＰＥは光線追跡法の高速処理を実現する適切なアーキテクチャであると結論される。

現在、１６ＰＥ構成のマルチプロセッサシステムが稼働している。このシステム上で画像生成実験を行った結果、ＰＥ数に比例して性能が向上することが示された[Yoshida,1991]。今後、実用に供する画像の生成実験を行い、ＳＩＧＨＴシステムの有効性を検証していく。

４章付録　ＳＩＧＨＴ開発環境

この付録では、ＳＩＧＨＴ試作機を開発するための主なサポートソフトウェアの概要を説明する。

付録４－１　マイクロプログラム開発環境

（１）　概要

ＳＩＧＨＴのマイクロプログラムを開発するためのツールとして、汎用のマイクロプログラムアセンブラＭＡＳＭを作成した。ＭＡＳＭは、ＡＭＤ社の汎用マイクロプログラムアセンブラＡＭＤＡＳＭをベースにし、それに幾つかの拡張を加えたものである。

ＭＡＳＭは、パスカルで記述されており、ＶＡＸのＶＭＳ環境下で実行される。ソフトウェア規模は、パスカル・ステートメント数で約９０００行である。

ＭＡＳＭは、次の３つのソフトウェアから成る。

１）ＭＡＳＭ１：ハードウェア定義。具体的には、マイクロプログラムのフィールド構成、ニーモニック等を定義する。

２）ＭＡＳＭ２：マイクロプログラムのアセンブル。

３）ＭＡＳＭ３：ＭＡＳＭ２の出力コードをインテル社の１６進形式コードに変換する。

（２）　ＭＡＳＭ１の構成

ＭＡＳＭ１は、マイクロプログラムの定義ファイル（フィールド定義、ニーモニック定義、等）を解析し、内部形式定義ファイルをつくる。

（ａ）基本機能（ＡＭＤＡＳＭの機能。詳細は、[AMD]を参照されたい）

ア）定数定義

NAME: EQU CONST

例.

```
EIGHT: EQU    8
R2:    EQU    B#010            ; (B#は２進数を表わす)
```

イ）フィールド定義

NAME: DEF FIELD1,FIELD2, ... ,FIELDn

例.

ADD:　DEF　3V,B#10110,5X,B#0011,4X,B#010

マイクロ命令ADD の定義。３ビットの変数フィールド、５ビットの定数フィールド、５ビットのDON'T CAREフィールド、４ビットの定数フィールド、４ビットのDON'T CAREフィールド、３ビットの定数フィールドから成る。ビット長は２４である。（最初に、WORD 24 なる宣言がなされていなければならない。）

ADD:　DEF　SHFTRT,B#0011,4X,R2

サブフィールド定義と定数定義を使って、上の定義を書き直した例。

ウ）サブフィールド定義

NAME: SUB FIELD1,FIELD2, ... ,FIELDn

例.

SHFTRT: SUB　3V,B#10110,5X

（ｂ）拡張機能（ＭＡＳＭ１で追加した機能）

ア）フィールドのデフォルト値の定義（注）

NAME: DFV FIELD1,FIELD2, ... FIELDn

FIELD1からFIELDnはすべて定数値でなければならない。

アセンブル時にDON'T CAREフィールドがあれば、ここで定義した値が埋め込まれる。

イ）フィールド・エンコード・ビット位置の定義

FEN POS1,POS2, ... POSn

－－

（注）実際にマイクロプログラムを記述してみると、この機能があると便利なことが多い。

フィールドがエンコードされている場合に使う。指定されたビット位置はフィールドのエンコード情報を持つ。このビット位置のデフォルト指定は許されない。このビット位置の値を見ることにより、複数のデフォルトフィールド定義文（上記　ア））のうちから、該当するものが選択できる。

使用例

図付４－１にＳＩＧＨＴのＭＰのハードウェア定義（一部）を示す。

（３）　ＭＡＳＭ２の構成

ＭＡＳＭ２は、マイクロプログラム・ソースファイルのアセンブルを行なう。

（ａ）基本機能（ＡＭＤＡＳＭの機能。詳細は、[AMD]を参照されたい。）

ア）ステートメント

[label:] control-word　argument

または、

[label:] definition-word　argumentの並び

・control-wordはORG,EQU等。

・defintion-wordはＭＡＳＭ１で定義したフィールド定義名。

・第二のフォーマットでargumentの数はdefinition-wordを定義する変数フィールドの数と一致しなければならない。

イ）アーギュメント

アーギュメントは、定数、式、定数名、ラベルのいずれかである。

```
; This file is a definiton file of SIGHT MP micro code.
TITLE   SIGHT MP Micro Code Definition File.  ( Ver 01.01 )
WORD 96                         ;Micro Word Size is 89 OUT OF 96
;Field configuration.
;    95   94   93       84      82   80     79   74 69   63     59
; ---------------------------------------------------------------------
; | DBG | TW | DW | MP OPC | SHIFT | Cn | STLE | A | B | EXT | JUMP |
; |     |    |    |        |       |    |      |   |   | SRC | COND |
; ---------------------------------------------------------------------
;     1    1    1       9       2    2      1   5   5    6      4
;
;       55    48   47   46     30      17    16   14     10     8    7    6
; ---------------------------------------------------------------------
;   | ADDR | SEQ | TE | DE | NEXT | TARAI |   DMAC   | ADDR | REG | EN | RW |
;   | MOD  | CTL |    |    | ADDR | INST  | DIR | RD | INCR | NO. |    |    |
; ---------------------------------------------------------------------
;        4     4    1    1     16      13     1    2      4     2    1    1

; ***************************************************
; Micro Code Definition
; ***************************************************
; ALU function            (15,14,13)
ADD:        EQU   3Q#0              ;ADD    R+S
SUBR:       EQU   3Q#1              ;SUB    S-R
SUBS:       EQU   3Q#2              ;SUB    R-S
OR:         EQU   3Q#3              ;OR     R or S
AND:        EQU   3Q#4              ;AND    R and S
NOTRS:      EQU   3Q#5              ;AND    /R and S        ( /R = not R )
EXOR:       EQU   3Q#6              ;EXOR   R exor S
EXNOR:      EQU   3Q#7              ;EXNOR  /(R exor S)     ( / = not )

; ALU Source              (12,11,10)
                                    ; ALU source
                                    ;  R side  S side
AB:         EQU   3Q#1              ;    A       B
ZB:         EQU   3Q#3              ;    0       B
ZA:         EQU   3Q#4              ;    0       A
DA:         EQU   3Q#5              ;    D       A
DZ:         EQU   3Q#7              ;    D       0
```

この間各種マイクロコードの定義

```
;
; ***************
; Field Definition
; ***************
;Default value
;               DBG,TW,DW  ALUC    SHT,CN,STLE  A-ADR      B-ADR      EXT-SRC
DEFAULT:  DFV   3B#000, 9Q#101, 5B#00000,    5B#00000, 5B#00000, 6B#000000,
;               JC     AM
/               4H#0,  4H#A,
;               SEQC     TE,DE  N.A      TARAI    DMAC     REG-ADR-CTL
/               7H#00:,  2B#00, 16H#0%, 13H#0%, 3B#000, 8H#00,       6H#00:
;MP ALU Operation with shift control
MPS:        DEF   3X, 3V:Q#1, 3V:Q#0, 3V:Q#1, 2V:B#00, 3X, 5V%H#0, 5V%H#0,
/               6V:H#00, 63X
;MP ALU Operation without shift control
MP:         DEF   3X, 3Q#3, 3V:Q#0, 3V:Q#1, 2B#00, 3X, 5V%H#0, 5V%H#0,
/               6V:H#00, 63X
```

この間各種フィールド定義

```
        END
```

図　付４－１　ＭＰのハードウェア定義

ウ）式

symbol operator symbol operator ...

operatorは+,-,*,/のいずれか。

式の評価は左から右へ順番に行われる。（オペレータの優先順位はない。）

symbolは評価の結果が定数にならなければならない。

（ｂ）拡張機能（ＭＡＳＭ２で追加した機能）

ア）％ｉｎｃｌｕｄｅ文

シンタックス

％ｉｎｃｌｕｄｅ　’Ｆｉｌｅｎａｍｅ／ｌｉｓｔ’

％ｉｎｃｌｕｄｅ　’Ｆｉｌｅｎａｍｅ／ｎｏｌｉｓｔ’

％ｉｎｃｌｕｄｅ　’Ｆｉｌｅｎａｍｅ’　　；ｎｏｌｉｓｔ指定

インクルード文の追加により、ファイル管理がしやすくなる。

イ）マクロ機能

・シンタックス

マクロ定義　（下線部キーワード）

macro macro-name %p1, %p2, ... , %pn

　　macro body

mend

マクロ呼び出し

macro-name p1, p2, ... , pn

マクロ呼び出し時に、実パラメータと仮パラメータの数が一致しないとき、

実パラメータ＞仮パラメータ　→　余分なパラメータを無視する。

実パラメータ＜仮パラメータ　→　エラーメッセージを出す。

のように処理される。

マクロ機能の制約は以下の通りである。

・マクロのネスト（マクロ本体の中で他のマクロが使用できること）ができる。

・マクロの再帰的使用（マクロ本体の中で自分自身を使用すること）はできない。

・ローカルラベル（マクロ本体の中で使うラベル）はパラメータとして引き渡せない。

・ローカルラベルは#で始まらねばならない。

・マクロ名は_で始まらねばならない。

ウ）アセンブラソースプログラムで実数が使える。

・シンタックス

F#実数

ここで、実数は３２ビットIEEE format。実数のフィールドは３２ビットの幅を有すること。

付録４－２　ＳＩＧＨＴシミュレータ

４．５節の性能解析で使用したシミュレータは以下の特徴を有する。

①ＳＩＧＨＴをマイクロプログラムレベルでシミュレートする。

②デバッガが入っているので本シミュレータ上でマイクロプログラムのデバグができる。

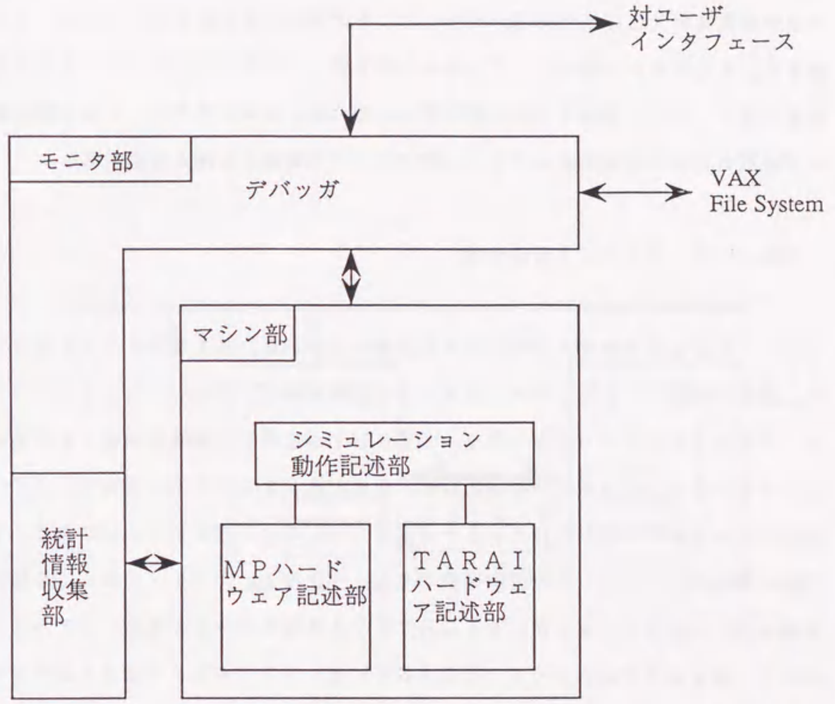
③統計情報収集ルーチンが入っているので、ＳＩＧＨＴの性能解析のためのデータが収集できる。

このシミュレータは、図付４－２に示すようにデバガを中心とするモニタ部とマシン部（ＳＩＧＨＴの動作記述部）から成る。モニタ部には、

・ＭＡＳＭでアセンブルしたマイクロプログラムの実行ファイルをロードする機能。

・マシン動作開始コマンド

・実行トレース機能



図付４－２　ＳＩＧＨＴシミュレータの構成

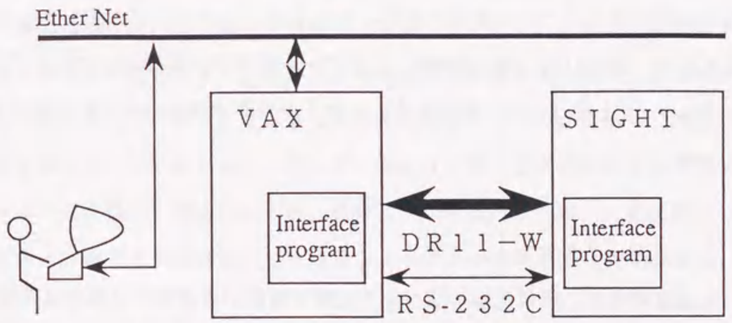
等がある。これらの機能を使って、プログラムをマシン部にロード／実行し、デバグコマンドによりバグ取りを行う。

このシミュレータは、試作機開発の初期段階で実際にマイクロプログラムを開発するために使用された。このシミュレータでバグ取りの終了したマイクロプログラムを試作機のデバグに使用できた。新しい計算機を開発する時にはこのようなシミュレータが極めて有効であることが改めて示された。

もう一つ重要なことは、このような構成にすることによりシミュレータ自身のデバグが容易に行えることである。すなわち、まずモニタ部を記述し、ついでマシン部を記述する。そしてマシン部のデバグはモニタ部を使って行うのである。モニタ部の論理的複雑度は浅く、マシン部のそれは比較的深い。それ故、モニタ部のデバグは容易に終わり、バグの取れたモニタ部を使ってマシン部のデバグが効率よく行えるのである。

付録４－３　ＳＩＧＨＴ動作環境

ＳＩＧＨＴはＶＡＸをホスト計算機としている。ＳＩＧＨＴのマイクロプログラム、画像の定義データ等はすべてＶＡＸ上で開発される。従って、これらのプログラムやデータをＳＩＧＨＴへロードしたり、ＳＩＧＨＴが生成した画像を保管するためにデータをＶＡＸに吸い上げる機能が必要となる。そのため、ＳＩＧＨＴの制御を含めてこれらの機能をＶＡＸから一括して行えることが望ましい。このようにすることにより、ＶＡＸの任意の端末からＳＩＧＨＴを遠隔操作できるようになる。図付４－３にＳＩＧＨＴの動作環境を示す。図示したように、ＳＩＧＨＴとＶＡＸはＲＳ－２３２Ｃシリアルラインと ＤＲ１１－ＷによるＤＭＡラインで接続されている。シリアルラインはＳＩＧＨＴの制御のために使用する。ＤＭＡラインは、プログラムやデータの転送のために使用する。また、このような環境でＳＩＧＨＴを動作させるためＶＡＸとＳＩＧＨＴそれぞれに接続用のソフトウェアを開発した。



図付４－３　ＳＩＧＨＴ動作環境

第５章　３次元ベクトル演算の並列実行に関する理論的基礎付け

５．１　まえがき

前章において、光線追跡法の主要な演算が３次元ベクトル演算であることを指摘し、その並列処理を実現するメカニズムを内蔵した計算機ＳＩＧＨＴについて詳しく述べた。本章では、ＳＩＧＨＴアーキテクチャの有効性に理論的裏付けを与えると共にその応用範囲の拡大をも狙って、３次元ベクトル演算および３×３マトリクス演算の並列処理に関して集合論的観点から考察を行う。

３次元ベクトル、３×３マトリクス演算をベースとして記述できる問題は、自然界が３次元空間であることから極めて多い。たとえば、コンピュータグラフィックスにおける光線追跡法　[Whitted,1980]の他に、ロボットのアーム制御[Fu,1989]、プラズマシミュレーション　[上村,1978]、等がある。これらの分野では次のように処理の高速化が求められている。

（１）コンピュータグラフィックス

光線追跡法は、幾何光学に基づいて光線の振舞いを忠実に計算する手法であり、極めて写実的な画像が生成できる。そのため、アニメーションのみならず、ＣＡＤ／ＣＡＭシステム[Thalmann,1987]や建築物の景観のシミュレーション[日経,1987]等への応用がなされている。また、光線追跡法は、音場の解析や、電波障害解析でも利用されている。しかしながら、これらの問題では膨大な計算量を要求することが多く、その高速演算が望まれている。例えば、光線追跡法によるＣＧでは１枚の画像を生成するのにミニコンピュータを使って数時間～数十時間を要する。会話型の画像生成環境を考えると、実時間で画像を生成する超高速計算機が必要となる。

（２）ロボティックス

ロボットのアーム制御では、数ミリ秒毎に外部環境をセンスしアーム位置を制御

する必要があり、実時間処理の要求がきびしい。ロボットのアーム制御は３次元ベクトル演算で記述されている。

（３）プラズマシミュレーション

粒子モデルによるプラズマシミュレーションでは、２０万粒子のシミュレーション１回に大型計算機で３０時間程度を要しており[阿部,1981]、より多数の粒子をより短時間でシミュレーションをしたいという要求がある。この計算の主要部は粒子の運動方程式を解くところにあり、それは３次元ベクトル演算として記述できる。

このような背景から、つぎのような興味ある問題が生まれる。すなわち、

「３次元ベクトル、３×３マトリクスから構成される演算はどの程度並列に実行できるのか、またそれを効率よく実行する機構はどのようなものか。」

という問題である。

本章では集合論的観点からこの問題を考察する。まず、３次元定数ベクトル、３×３定数マトリクスの集合を考え、その上に基本演算を定義する。更に３次元ベクトル変数を導入してこの集合を拡張する（拡張した集合をＧとする。）。次いで、集合Ｇがどのような性質を持つかを明らかにする。そして、集合Ｇに属する式の並列実行を検討し、その並列実行に必要な機能を明らかにする。最後に、並列実行した場合と逐次実行した場合との演算ステップ数の比で演算並列度を定義し、ベクトル演算の並列度を論じる。

５．２　諸定義

この節では、３次元ベクトル演算、３×３マトリクス演算を議論するために必要な基本事項の定義を行う。

５．２．１　Skewed配置マトリクス

マトリクスのSkewed配置はIlliac-IV[加藤,1976]の研究で提案された。その性質に関しては[Lawrie,1975]で詳しく論じられている。この章ではSkewed配置されたマトリクスを前提とした議論を進める。

この章で使うSkewed配置マトリクスは次のような形である。以下で、m_{ij}はマトリクス要素であり、m_iはベクトルで(m_{i1}, m_{i2}, m_{i3})を意味する。また、⊖、⊖は後述するシフト演算である。

通常配置	skewed配置
$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{pmatrix}$	$\begin{pmatrix} m_{11} & m_{12} & m_{13} \\ m_{23} & m_{21} & m_{22} \\ m_{32} & m_{33} & m_{31} \end{pmatrix}$
あるいは	あるいは
$\begin{pmatrix} m_1 \\ m_2 \\ m_3 \end{pmatrix}$	$\begin{pmatrix} m_1 \\ \ominus m_2 \\ \ominus m_3 \end{pmatrix}$

５．２．２　基本演算

３次元実ベクトル、３×３実マトリクスの全体からなる集合をＦとする。ここで、ベクトルは横ベクトルとする。以下では、Ｆのベクトル要素をａ、ｂ等と書き、マトリクス要素をＭ、Ｎ等と書く。

まず、Ｆ上に基本演算を定義する。

（１）　基本ベクトル演算（ベクトルーベクトル演算）

①ベクトル要素間の二項演算　⊙

$$a \odot b = (a1 \odot b1,\ a2 \odot b2,\ a3 \odot b3)$$

⊙は＋、－、×、／のいずれかである。ただし、割算では、ｂはstrictly-nonzero、すなわち、$(b1 \cdot b2 \cdot b3 \neq 0)$、でなければならない。

②ベクトル要素の単項演算　－

$$-a = (-a1, -a2, -a3)$$

③シフト演算　⊖、⊖

$$\ominus a = (a3, a1, a2)$$

$$\ominus a = (a2, a3, a1)$$

以下の議論でわかるように、シフト演算はベクトル演算において中心的役割を果たす。

（２）ベクトルーマトリクス演算

積　ａＭ　を考える。

（３）マトリクスーベクトル演算

積　Ｍａ　を考える。　この場合積は、厳密には、'（Ｍ'ａ）と書くべきであるが、煩雑になるので以下ではＭａのように記述する。

（４）基本マトリクス演算（マトリクスーマトリクス演算）

①マトリクス要素間の二項演算　⊙

$$M \odot N = \begin{pmatrix} m_1 \odot n_1 \\ m_2 \odot n_2 \\ m_3 \odot n_3 \end{pmatrix}$$

通常配置

$$M \odot N = \begin{pmatrix} m_1 \odot n_1 \\ (\ominus m_2) \odot (\ominus n_2) \\ (\ominus m_3) \odot (\ominus n_3) \end{pmatrix}$$

skewed配置

⊙は＋、－、×、／のいずれかである。割算では、除数はstrictly-nonzeroでなければならない。

②マトリクス要素の単項演算　－

$$-M = \begin{pmatrix} -m_1 \\ -m_2 \\ -m_3 \end{pmatrix}$$

通常配置

$$-M = \begin{pmatrix} -m_1 \\ -(\ominus m_2) \\ -(\oplus m_3) \end{pmatrix}$$

skewed配置

③シフト演算　⊖、⊕、①、①

$$\ominus M = \begin{pmatrix} \ominus m_1 \\ \ominus m_2 \\ \ominus m_3 \end{pmatrix} \qquad ①M = \begin{pmatrix} m_2 \\ m_3 \\ m_1 \end{pmatrix}$$

通常配置

$$\ominus M = \begin{pmatrix} \ominus m_1 \\ \ominus(\ominus m_2) \\ \ominus(\oplus m_3) \end{pmatrix} \qquad ①M = \begin{pmatrix} \ominus m_2 \\ \oplus m_3 \\ m_1 \end{pmatrix}$$

skewed配置

である。⊕、①も同様。

④対角要素の抽出diag、diag^r、diag^l

通常配置の場合

$\mathrm{diag}(M) = (m_{11}, m_{22}, m_{33})$

$\mathrm{diag}^r(M) = (m_{31}, m_{12}, m_{23})$

$\mathrm{diag}^l(M) = (m_{21}, m_{32}, m_{13})$

ここで、m_{ij}はマトリクス要素

skewed配置の場合

$\mathrm{diag}(M) = (m_{11}, m_{21}, m_{31})$

$\mathrm{diag}^r(M) = (m_{32}, m_{12}, m_{22})$

$\mathrm{diag}^l(M) = (m_{23}, m_{33}, m_{13})$

ここで、m_{ij}はマトリクス要素

（5）マトリクスーマトリクス積

通常のマトリクス積を考える。

5．2．3　ベクトル変数

ベクトル変数xを導入し、集合Fを拡張する。（以下では、x＝(x1、x2、x3)あるいはx＝(x、y、z)と記す。）集合G_0（＝F）の要素とxに上記の基本演算を行って作られる式をG_0に付加した集合を$G_1[x]$と書く（脚注）。すなわち、

$$G_1[x] = G_0 \cup \{e \odot x : e \in G_0,\ \odot は基本演算\}$$

と書く。次いで、G_nを

$$G_n[x] = G_{n-1}[x] \cup \{e_1 \odot e_2 : e_1, e_2 \in G_{n-1}[x],\ \odot は上記演算\}$$

とし、その推移的閉包G［x］を次のように定義する。

$$G[x] = \bigcup_n G_n[x]$$

G［x］はベクトルxを変数とし、定数ベクトル、定数マトリクスを係数として作られ

（脚注）意味のある演算だけを考える。たとえば、a＋xは意味があるが、M＋xは意味を持たない。

るすべての多項式およびマトリクスの集合である。例えば、$(x^3, y^2, z^2) \in G[x]$ であるが、$(x^3, y^2, z) \notin G[x]$ である。

5．2．4　微分演算子

G［x］上に微分演算子を導入する。すなわち、$E(x) = (E_x(x), E_y(x), E_z(x)) \in G[x]$ に対して、微分演算子を、

$$\frac{dE(x)}{dx} = \left(\frac{\partial E_x(x)}{\partial x}, \frac{\partial E_y(x)}{\partial y}, \frac{\partial E_z(x)}{\partial z}\right)$$

のように定義する。（これを$E'(x)$とも記す。）

5．3　諸性質

5．3．1　基本演算が持つ性質

前節のように基本演算を定義すると、集合Fに関して次の性質が成り立つ。

［性質１］

Fは基本演算に関して閉じている。

［証明］

Fのつくりかたから明かである。

同様に、微分演算を除いた基本演算に対して、

［性質２］

Gは基本演算に関して閉じている。

がいえる。

5．3．2　微分演算が持つ性質

この節では微分演算がG［x］で閉じていること、すなわち、G［x］の式を微分した結果の式はまたG［x］の式であることを以下に示す。最初に、ベクトルのシフト演算に関していくつかの性質を述べておく。

［性質３］⊖に関する二、三の公式（⊖についても同様の関係が成立する。）

$f(x)$、$g(x) \in G[x]$、$M \in G[x]$ のとき、

1．$\ominus(f(x) \odot g(x)) = (\ominus f(x)) \odot (\ominus g(x))$ ；⊙は二項演算

2．$\ominus(\odot f(x)) = \odot(\ominus f(x))$ ；⊙は単項演算

3．$\ominus\ominus f(x) = \ominus f(x)$

4．$\ominus(Mf(x)) = (\uparrow M)(\ominus f(x))$

$\ominus(f(x)M) = (\ominus f(x))(\downarrow M)$

［証明］

いずれも定義にもどって考えれば明かである。たとえば４．の上の式は、$M = (m_{ij})$、$f(x) = (f_i)$ とすると

$Mf(x) = (\Sigma m_{1i} f_i, \Sigma m_{2i} f_i, \Sigma m_{3i} f_i)$

$(\uparrow M)(\ominus f(x)) = (\Sigma m_{2i} f_i, \Sigma m_{3i} f_i, \Sigma m_{1i} f_i)$

より成立することがわかる。（マトリクスはskewed配置であることに注意）

［性質４］（シフト還元）

$\ominus f(x)$ なる形の式は中間のレベルにシフト演算を含まない式に還元できる。すなわち、還元した式ではシフト演算は$\ominus x$、$\ominus x$なる形でのみ現れる。

［証明］

付録５－１に示す。

シフト還元の例

$$\ominus(x^3 + Mx) = (\ominus x)^3 + (\uparrow M)(\ominus x)$$

次に、微分演算に関して二、三の公式を述べる。

［性質５］

$(f(x) \pm g(x))' = f'(x) \pm g'(x)$

$(f(x) \times g(x))' = f'(x) \times g(x) + f(x) \times g'(x)$

$(\ominus x)' = (\oplus x)' = 0$

［証明］定義にもどって考えれば明か。

性質４、性質５を使うと、次の性質が証明できる。

［性質６］

Ｇ［ｘ］の式を微分した結果の式は基本演算の組合せで記述できる。すなわち、Ｇ［ｘ］は、上記微分演算に関して閉じている。

［証明］

付録５－２に示す。

この性質により、微分した結果の式は他の基本演算と同様並列に計算できる。

光線追跡法では、曲面とその法線ベクトルが重要な役割を果たすが、それに関して性質６から次の系が導かれる。

［系］

曲面の式がＧ［ｘ］に属すなら、その法線ベクトルを表す式はＧ［ｘ］に属す。

５．４　基本演算の並列実行

前節で定義した基本演算の並列実行を考える。

５．４．１　マトリクス演算の基本ベクトル演算による表現

演算の実行という観点からは、マトリクスーベクトル積、マトリクスーマトリクス積を基本ベクトル演算、基本マトリクス演算を用いて記述しておくのが都合よい。

（１）マトリクスーベクトル積　Ｍａ

$$Z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} m_1 \times a \\ \oplus(\ominus m_2) \times a \\ \ominus(\ominus m_3) \times a \end{pmatrix}$$

とおく。（Ｚは通常の配置になっている。）そうすると、

$$Ma = \mathrm{diag}(Z) + \ominus \mathrm{diag}^r(Z) + \ominus \mathrm{diag}^l(Z)$$

と書ける。

（２）ベクトルーマトリクス積　ａＭ

$$Z = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = \begin{pmatrix} a \times \mathrm{diag}(M) \\ a \times \ominus \mathrm{diag}^r(M) \\ a \times \ominus \mathrm{diag}^l(M) \end{pmatrix}$$

とおく。（Ｚは通常の配置になっている。）そうすると、

$$aM = \mathrm{diag}(Z) + \ominus \mathrm{diag}^r(Z) + \ominus \mathrm{diag}^l(Z)$$

と書ける。

（３）マトリクスーマトリクス積　Ｌ＝ＭＮ

$$Z^{i}=\begin{pmatrix} m_{i}\times diag\,(N) \\ m_{i}\times \ominus diag^{r}\,(N) \\ m_{i}\times \ominus diag^{l}\,(N) \end{pmatrix} \qquad i=1,2,3$$

と置くと、

$$1_{1}=diag\,(Z^{1})+\ominus diag^{r}\,(Z^{1})+\ominus diag^{l}\,(Z^{1})$$
$$\ominus 1_{2}=\ominus diag\,(Z^{2})+diag^{r}\,(Z^{2})+\ominus diag^{l}\,(Z^{2})$$
$$\ominus 1_{3}=\ominus diag\,(Z^{3})+\ominus diag^{r}\,(Z^{3})+diag^{l}\,(Z^{3})$$

となる。

５．４．２　並列演算機構

マトリクス演算は前節のように基本ベクトル演算を用いて表現できる。従って、以下では基本ベクトル演算に対する並列実行機構を議論する。

まず、（２個以上の演算器を使うと仮定して）演算器を幾つ使うかが問題となるが、評価関数として、演算器の利用率を用いると３個使うときが最も良くなることは明かである。したがって、以下では演算器個数を３とする。

つぎに、これらの基本演算を行うためのメカニズムについて述べる。具体的には、３個の演算器（x-,y-,z-演算器と呼ぶ）と３個のメモリ（x-,y-,z-メモリと呼ぶ）を仮定し、その上にどのような機能があれば基本演算が逐次計算機で演算した場合の１／３の演算回数でできるかを検討する。尚、ベクトルデータは３個のメモリの同一アドレスに格納され、また、マトリクスデータは連続するアドレスに格納され、各行ベクトルは同一アドレスに格納されるものとする。

考えるべき演算は、二項演算、単項演算、シフト演算、対角要素の抽出演算だけである。二項演算と単項演算は対応する演算器とメモリの間にデータの転送ラインがあれ

ば実行できる。シフト演算は(x-,y-,z-)メモリから(y-,z-,x-)演算器への転送ラインと(x-,y-,z-)メモリから（z-,x-,y-)演算器への転送ラインがあれば実行できる。対角要素の抽出演算は、３通りのアドレスオフセット機能（具体的には、(+0,+1,+2)、(+2,+0,+1)、(+1,+2,+0)の３通り）があれば実行できる。したがって、これらの機能を具備した演算装置をつくればよい。一例として、⊖ａ×⊖ａ（＝($a_y*a_z, a_z*a_x, a_x*a_y$)）の実行を図５－１に示す。

ここで注意すべきことは、「データがメモリから演算器に転送され、演算を行って、結果がメモリに格納される。」というシーケンスで１回の演算が完了することを考えると、四則演算が伴うときシフト操作と対角要素抽出操作は四則演算に付随した操作となることである。例えば、⊖ａ×⊖ａ計算は１回の演算で実行できる。これは、演算の回数を考えるとき重要となる。

Ｇ［ｘ］の式Ｅに対し、Ｅを逐次計算機で計算した場合の演算回数をＮｓ（Ｅ）とし、上記の演算装置で計算した場合の演算回数をＮｐ（Ｅ）とする。また、これらの比、

$$r=Ns\,(E)\,/\,Np\,(E)$$

を演算並列度とよぶ。このように定義するとｒは、

$$1\leqq r\leqq 3$$

の範囲の値をとる。

５．２．２節の各基本演算に対して、それを逐次的に計算した場合の演算回数を数えて比較すれば、次の結果を得る（表５－１参照）。

［性質７］

上記の各基本演算に対してｒの値は３である。

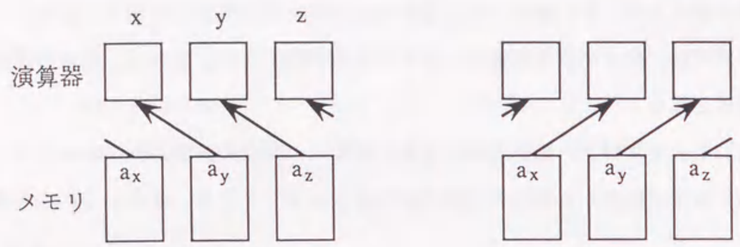


図５－１　ベクトルの並列演算

$\ominus a \times \ominus a = (a_y a_z, a_z a_x, a_x a_y)$ の計算

（１）データ (a_x, a_y, a_z) を左上の演算器に転送する。

（２）データ (a_x, a_y, a_z) を右上の演算器に転送する。

（３）乗算を行う。

（４）結果を真下のメモリに転送する。

表５－１　　演算並列度

演算			Ｎｐ	Ｎｓ	ｒ
基本演算	ベクトル	二項演算	1	3	3
		単項演算	1	3	3
		シフト演算	1	3	3
	ベクトルー マトリクス積		5	１５	3
	マトリクスー ベクトル積		5	１５	3
	マトリクス	二項演算	3	9	3
		単項演算	3	9	3
		シフト演算	3	9	3
		対角要素抽出	1	3	3
	マトリクスー マトリクス積		１５	４５	3
拡張演算	総和		2	2	1
	連乗		2	2	1
	内積		3	5	１．６７
	外積		3	9	3
	転置行列		3	9	3
	行列式		7	１７	２．４３
	逆行列		１９	５３	２．７９
	二次形式		9	２３	２．５６

５．５　拡張演算の並列実行

本節では、ベクトル演算やマトリクス演算で使用される内積、外積、逆行列等の種々の演算が５．２節で定義した基本演算の組合せで記述できることを示し、その演算並列度を検討する。（線形代数[佐竹,1971]で使用される主要な演算について検討する。）

まず、以下の演算に有用な２つの演算を定義する。

ベクトル要素の総和：　$\dot{\Sigma}x = x + \ominus x + \ominus x$

$= (\Sigma x_1, \Sigma x_1, \Sigma x_1)$

ベクトル要素の連乗：　$\dot{\Pi}x = x \times \ominus x \times \ominus x$

$= (\Pi x_1, \Pi x_1, \Pi x_1)$

これらの演算は、本質的に逐次演算であり、並列演算はできない。これらの演算の演算回数は２である。また、ｒの値は１である。

（１）ベクトル演算

①内積：ここでは内積を次のように定義する。

$$(a, b) = \dot{\Sigma}(a \times b)$$

内積演算の結果はスカラーであるが、ここでは、ベクトルとして扱う。そうすることによる計算上の不都合は何もない。

②外積：

$$[a, b] = \ominus a \times \ominus b - \ominus a \times \ominus b$$

③回転（ｒｏｔ）：回転は次のように記述できる。

$$rot(E) = \ominus\frac{d(\ominus E)}{dx} - \ominus\frac{d(\ominus E)}{dx}$$

従って、Ｅ∈Ｇ［ｘ］なら性質６によりｒｏｔ（Ｅ）∈Ｇ［ｘ］である。

④発散（ｄｉｖ）：発散は次のように定義する。

$$div(E) = \dot{\Sigma}\frac{dE}{dx}$$

演算結果は、ベクトルと考える。回転と同様Ｅ∈Ｇ［ｘ］ならｄｉｖ（Ｅ）∈Ｇ［ｘ］である。

（２）マトリクス演算　（skewed配置行列）

①転置行列：転置行列は次のように記述できる。

$$'M = \begin{pmatrix} diag(M) \\ diag^{r}(M) \\ diag^{l}(M) \end{pmatrix}$$

②行列式：行列式の値は次のように記述できる。（ただし、演算結果はベクトル値と考えることにする。）

$$det(M) = \dot{\Sigma}(m_1 \times \ominus(\ominus m_2) \times \ominus(\ominus m_3) - m_1 \times \ominus(\ominus m_2) \times \ominus(\ominus m_3))$$

③逆行列：逆行列は、以下のように定義される。

$$M^{-1}=\frac{1}{\Delta}\begin{pmatrix}\Delta_{11} & \Delta_{21} & \Delta_{31}\\ \Delta_{12} & \Delta_{22} & \Delta_{32}\\ \Delta_{13} & \Delta_{23} & \Delta_{33}\end{pmatrix}$$

ここで、$\Delta=det(M)$、Δ_{ij}は(i,j)余因子[佐竹,1971]である。

ここでは、skewed配置を前提としているから、上式は次のようになる。

$$M^{-1}=\frac{1}{\Delta}\begin{pmatrix}\Delta_{11} & \Delta_{21} & \Delta_{31}\\ \Delta_{32} & \Delta_{12} & \Delta_{22}\\ \Delta_{23} & \Delta_{33} & \Delta_{13}\end{pmatrix}$$

そうすると、余因子行列の各行は、次のように記述できる。

$$(\Delta_{11}\ \Delta_{21}\ \Delta_{31})=\ominus diag^{r}(M)\times\ominus diag^{l}(M)-diag^{r}(M)\times diag^{l}(M)$$

$$(\Delta_{32}\ \Delta_{12}\ \Delta_{22})=\ominus diag(M)\times\ominus diag^{l}(M)-diag(M)\times diag^{l}(M)$$

$$(\Delta_{23}\ \Delta_{33}\ \Delta_{13})=\ominus diag(M)\times\ominus diag^{r}(M)-diag(M)\times diag^{r}(M)$$

尚、3×3行列の場合、直接計算のほうが掃き出し法で計算するより演算回数は少ない。

（3）二次形式

二次形式は、（x、Mx）で与えられる。ここで、xはベクトル、Mはマトリクスである。したがって、以下のように記述できる。

$$(x、Mx)=\Sigma(x\times Mx)$$

以上の演算の演算並列度を表5－1に示す。前節および本節の議論からわかるよ

うに、ここで検討しているベクトル演算において並列性を阻害する要因は、「演算の本質的逐次性」だけである。したがって、次の性質が成り立つ。

［性質8］与えられた式が上記の要因を含まないならば、すなわち、Σ、Πを含まなければ、与式は演算並列度3で計算できる。

５．６　まとめ

本章ではグラフィックス、ロボティックス、等種々の分野で基本演算となる３次元ベクトル、３×３マトリクス演算の並列実行について考察した。まず、３次元ベクトル、３×３マトリクスから構成される式の集合を考え、その性質を調べた。そして、線形代数に現れる基本的なベクトル演算がこの集合に含まれることを示した。したがって、この集合は３次元ベクトル演算にあらわれるほとんど全ての式を含む。

ついで、この集合に属す式の並列演算機構を考え、その上で計算したときの演算並列度について検討した。残念ながらこの集合には本質的に逐次的な演算を必要とする式を含む。そのような式に対しては、並列演算器の演算並列度は低下する。しかし、それを含まない式に対しては、並列演算器は演算並列度３で式の計算を行える。

ＳＩＧＨＴのTARAI演算器は本章で述べた並列演算機構を実現したものである。３章で述べたように、TARAI演算器により光線追跡法の並列処理効率が著しく向上している。

今後の課題として、種々の応用分野において、プログラムのどれくらいの部分がベクトル演算で記述できるか明かにする必要がある。また、ｎ次元への拡張も検討する必要がある。

５章付録　証明

（付録５－１）　性質４の証明

⊖について示す。（⊖についても同様）

⊖ｋ（ｘ）∈Ｇ［ｘ］なる式を考えたとき、ｋ（ｘ）は次のいずれかの形をしていなければならない。（ここでｋ（ｘ）はベクトルであることに注意。）

①ｆ（ｘ）⊙ｇ（ｘ）；⊙は二項演算
　　；ｆ（ｘ），ｇ（ｘ）∈Ｇ［ｘ］
②⊙ｆ（ｘ）　　；⊙は単項演算
　　；ｆ（ｘ）∈Ｇ［ｘ］
③⊖ｆ（ｘ），⊖ｆ（ｘ）；ｆ（ｘ）∈Ｇ［ｘ］
④Ｍｆ（ｘ）　　；Ｍは定数マトリクス
　ｆ（ｘ）Ｍ　　；ｆ（ｘ）∈Ｇ［ｘ］
⑤ａ，ｘ　　；定数／変数ベクトル

上記のうち①～④にシフト演算を適用すると、性質３により部分式のシフト演算に還元することができる。また、⑤の場合はシフト還元された形になっている。この還元手続きを繰り返し行うことにより⊖ｋ（ｘ）は中間レベルにシフト演算を含まない式に還元できることをいう。そのためには、有限回の繰り返しで還元手続きが終了することを言えばよい。

いま、式に含まれる基本演算の数をＮｏｐ（・）で表す。

Ｎｏｐ（⊖（ｆ（ｘ）⊙ｇ（ｘ））＞Ｎｏｐ（⊖ｆ(ｘ)）
Ｎｏｐ（⊖（ｆ（ｘ）⊙ｇ（ｘ））＞Ｎｏｐ（⊖ｇ(ｘ)）
Ｎｏｐ（⊖（⊙ｆ（ｘ））＞Ｎｏｐ（⊖ｆ（ｘ））
Ｎｏｐ（⊖（⊖ｆ（ｘ））＞Ｎｏｐ（⊖ｆ（ｘ））
Ｎｏｐ（⊖（⊖ｆ（ｘ））＞Ｎｏｐ（ｆ（ｘ））

$$Nop(\ominus(Mf(x))) = Nop(①M(\ominus f(x))) > Nop(\ominus f(x))$$

$$Nop(\ominus(f(x)M)) = Nop((\ominus f(x))①M) > Nop(\ominus f(x))$$

であるから、部分式の基本演算の数は元の式の基本演算の数より必ず小さい。

$Nop(\ominus k(x))$ は有限であるから、シフト演算の入った部分式はいつかは、$\ominus x$、$\ominus a$ 等の形になる。［証明終］

（付録５－２）　性質６の証明

$k(x) \in G[x]$ に対して、$k(x)$ をシフト還元する。それをあらためて $k(x)$ とする。そうすると、$G[x]$ の作り方から $k(x)$ は次のいずれかの形にかける。

①$f(x) \odot g(x)$ ；⊙は二項演算
　；$f(x)$，$g(x) \in G[x]$

②$\odot f(x)$ ；⊙は単項演算
　；$f(x) \in G[x]$

③$\ominus x$，$\ominus x$

④$Mf(x)$ ；Mは定数マトリクス

　$f(x)M$ ；$f(x) \in G[x]$

⑤a，x ；定数／変数ベクトル

各場合につきその微分が$G[x]$に入るか否かを調べる。

①$f(x) \odot g(x)$：⊙は＋、－、×、／のいずれかである。$f'(x)$、$g'(x) \in G[x]$ ならば性質５により $(f(x) \odot g(x))' \in G[x]$

②$\odot f(x)$：$f'(x) \in G[x]$ ならば $(\odot f(x))' \in G[x]$

③、⑤は明か。

④$(Mf(x))' =$

$$\left(\frac{\partial}{\partial x}(m_{11}f_1+m_{12}f_2+m_{13}f_3),\ \frac{\partial}{\partial y}(m_{21}f_1+m_{22}f_2+m_{23}f_3),\ \frac{\partial}{\partial z}(m_{31}f_1+m_{32}f_2+m_{33}f_3)\right)$$

ここで、

$$V = \begin{pmatrix} (f(x))' \\ (\ominus f(x))' \\ (\ominus f(x))' \end{pmatrix}$$

$$W = \begin{pmatrix} diag(V) \times diag(M) \\ \ominus diag^r(V) \times diag^r(M) \\ \ominus diag^l(V) \times diag^l(M) \end{pmatrix}$$

と置くと上式は、

$$diag(W) + \ominus diag^r(W) + \ominus diag^l(W)$$

とかける。

従って、$(f(x))'$、$(\ominus f(x))'$、$(\ominus f(x))' \in G[x]$ なら $(Mf(x))' \in G[x]$ である。$(f(x)M)'$ も同様である。

以上から $f'(x)$、$g'(x)$、$(\ominus f(x))'$、$(\ominus f(x))'$ について、すなわち、部分式について上記の①～⑤のチェックを行うことにより、元の $k'(x)$ が $G[x]$ にはいるかどうかがわかる。（ただし、$\ominus f(x)$、$\ominus f(x)$ についてはシフト還元を行った式をチェックする。）よって、残るは有限回で部分式のチェックが終了することをいえばよい。

①、②、④について、シフト演算を除く基本演算の数を Nop（・）とすると、

$$Nop(k(x)) > Nop(f(x))$$

$$Nop(k(x)) > Nop(g(x))$$
$$Nop(k(x)) > Nop(\ominus f(x))$$
$$Nop(k(x)) > Nop(\ominus f(x))$$

である。なぜなら、シフト還元により、シフト演算以外の基本演算の数は変わらないからである。また、シフト還元を行った式では、シフト演算は$\ominus x$、$\oplus x$の形であらわれる。この式はこれ以上分解の必要がないので、部分式の分解においてシフト演算の数をカウントする必要はない。

よって、部分式は有限回で③、⑤の形に帰着する。故に、$k'(x) \in G[x]$がいえる。［証明終］

第6章　結論

本論文では、光線追跡法の高速処理に向けてアルゴリズムおよびコンピュータアーキテクチャの両面から、以下の研究を行った。

（1）動的部分木を用いた光線追跡法の高速化
（2）光線追跡法向計算機（ＳＩＧＨＴ）の開発と性能評価
（3）３次元ベクトル演算の並列実行に関する理論的基礎付け

本論文で得られた主な結果を以下に示す。

（1）動的部分木を用いた光線追跡法の高速化

・光線追跡法の高速化の本質は、画面上の各画素に対して、その画素に投影される物体を効率よく求めることにあるとの観点から、木構造化された物体定義データを用いた光線追跡法の高速化を検討した。画面の小領域に投影される物体数は少数であることから、これらの物体からなる木構造（部分木）を動的に作り、小領域の画像生成はこの木を利用して行う手法を提案した。これは、部分木を作るコストが少々大きくても、小領域内の画素数がある程度大きければ、部分木を用いた光線追跡による計算時間削減効果により、トータルの計算時間は小さくなるとの考えによる。

・木構造データを用いる場合は、効率の良い枝刈手法が要求される。本論文で用いた木構造データでは、木の各ノードにその子孫の物体をすべて含むバウンディング・ボリューム（ＢＶ）を設定している。光線追跡時に、あるノードのＢＶと光線が交差したら、その子ノードを視点からの距離の近い順の並べ代える。そして、視点に近い子ノードから深さ方向優先で探索する。このようにすることで、子ノードを探索して得られた物体の視点からの距離が、次の子ノード（兄弟ノード）のＢＶの視点からの距離より近ければ、兄弟ノードの探索は不要である。このような、ノードのソーティングによる枝刈手法が適用できる。

・小領域内がある１つの物体の投影像だけで占められる（領域内の一様性）なら、その

領域はその物体に対するレンダリングをすればよい。この場合、（部分）木の探索はまったく不要である。本論文では、領域内の一様性をテストする簡単な手法を提案した。

・本論文で提案した高速光線追跡アルゴリズムは、部分木、ソーティング、一様性テストを併用したアルゴリズムである。このアルゴリズムをインプリメントして、高速化効果を調べた。その結果、部分木による高速化効果が極めて高く、その有効性が示された。

・実験、および、アルゴリズムの解析により、本論文のアルゴリズムは、従来より高速といわれているArvoのアルゴリズム[Arvo,1987]より高速であることを示した。

（２）光線追跡計算向計算機（ＳＩＧＨＴ）の開発と性能評価

・光線追跡法に内在する並列性を抽出した。光線追跡法は、表示画面の画素毎の追跡計算が並列に実行でき（画素レベル並列処理）、さらに追跡計算自身が並列実行できる（演算レベル並列処理）ことを示した。後者は、３次元空間のベクトル演算の並列実行という問題に一般化してとらえることができる。

・これらの並列処理を実現する計算機アーキテクチャＳＩＧＨＴを提案した。ＳＩＧＨＴは、画素レベル並列処理をマルチプロセッサ構成で、演算レベル並列処理をＴＡＲＡＩと名付けた並列演算ユニットで実現する。ＴＡＲＡＩは要素プロセッサ内に組み込まれた核ユニットであり、３次元ベクトル演算を効率よく並列実行する。要素プロセッサは、この他に、ＭＰユニット、ＤＢＭユニットを含む。これらはＴＡＲＡＩと共に並行動作して、光線追跡法の徹底的な並列処理を実現する。

・ＳＩＧＨＴ上の光線追跡アルゴリズムを提案した。要素プロセッサ内各ユニットが並列に動作するよう工夫したアルゴリズムである。

・ＳＩＧＨＴの要素プロセッサの性能を解析した。ＴＡＲＡＩユニットの演算並列度は、2.20～2.62であることが示された。このことは、光線追跡計算において３次元ベクトル演算の割合が極めて高いことを示しており、ＴＡＲＡＩの有効性が示されたものと考える。また、ＴＡＲＡＩの演算とＤＢＭからＴＡＲＡＩへのデータ転送のパイプライン処理の実現により、等価的に大規模高速メモリが構成できることが示された。典型的商用計算機ＶＡＸ１１／７８０との比較の結果、要素プロセッサはＶＡＸの約１０倍の性能を持つことが示された。

（３）３次元ベクトル演算の並列実行に関する理論的基礎付け

・ＴＡＲＡＩの演算並列度に関して理論的基礎付けを与える目的で、集合論的観点から３次元ベクトル演算の並列実行に関する考察を行った。

・３次元定数ベクトル、３×３定数マトリクスからなる集合を考え、その上に基本演算を定義する。この集合に３次元ベクトル変数を導入し、集合を拡張する。この拡張した集合は基本演算に関して閉じており、基本演算はＴＡＲＡＩの演算機構により演算並列度３で実行できることを示した。

・上記拡張集合上に微分演算を導入し、微分演算が閉じていることを示した。

・また、拡張集合上に拡張演算を導入し、その演算並列度を検討した。拡張演算は本質的に逐次な演算を含むものがある。本質的に逐次な演算だけが演算並列度を低下させる要因であり、その部分を除けば拡張演算は演算並列度３で実行できることを示した。

・ここで述べた式の並列演算に関する集合論的検討は、並列演算器の性能や、適応性に関する見通しを極めて明るくするものである。ここで展開した方法論は、この種の問題に広く応用できよう。

光線追跡法により生成された画像の品質は、他の手法を凌駕するものであり、計算時間の問題が克服されれば広く利用されることは疑う余地がない。その高速処理のために提案したＳＩＧＨＴは、光線追跡法に内在する並列性を徹底的に引き出すものであり、その並列処理に最適なアーキテクチャを有しているものと考えている。

本論文で残された主な課題として、次のものがある。

（１）に関しては、２次光線に対する提案アルゴリズムの有効性の評価。

（２）に関しては、ＳＩＧＨＴを実用に供するために必要な開発環境等の整備。

（３）に関しては、ｎ次元への拡張（特に、演算器構成法に関する検討）。

本論文が、コンピュータグラフィックス分野の発展にいささかでも貢献できれば筆者にとって望外の喜びである。

謝辞

本論文をまとめるにあたり、名古屋大学工学部鳥脇純一郎教授には丁寧なご指導を賜りました。深く感謝いたします。また、名古屋大学工学部杉江昇教授、並びに横井茂樹助教授には適切なご助言を頂きました。厚くお礼申し上げます。さらに、学生時代より公私にわたってご指導いただいた名古屋大学福村晃夫名誉教授（現中京大学情報科学部学部長）に深く感謝いたします。

本研究は、ＮＴＴ基礎研究所およびヒューマンインタフェース研究所において、多くの方々のご指導を得て行われました。研究の機会を与えて頂くと共に、日頃からご指導頂いた元畔柳功芳基礎研究部長（現東京工科大学教授）、高野陸男知能ロボット研究部長に心から感謝いたします。また、基礎研究所においてご指導ご鞭撻いただいた塚本克治博士（現積水ハウス）、雨宮真人博士（現九州大学）、増田功博士（現セコムＩＳ研究所）、梅田三千雄博士（現大阪電気通信大学）、内藤誠一郎博士、並びにヒューマンインタフェース研究所においてご指導ご鞭撻いただいた安田浩博士、若菜忠博士（現明電舎）、末永康仁博士、滝川啓氏、酒井高志氏、木暮賢司博士に心より感謝いたします。

本研究は多くの方々との協同研究や討論を経ています。ＳＩＧＨＴのプロトタイプ機の完成は、知能ロボット研究部吉田雅治氏、高橋時市郎氏の努力に負うところ大きく、両氏に心より感謝いたします。特に吉田氏には、１２年にわたって職場を共にし、ハードウェアの基礎知識を教えていただきました。それは、アーキテクチャ検討のための強力な武器となりました。動的部分木を用いた光線追跡アルゴリズムでは、新谷幹夫博士、斉藤隆文博士との討論に負うところ大きく、両博士に深く感謝いたします。

また、日頃より公私にわたって討論をしていただいた多くの方々の存在なしに、本研究をまとめることはできませんでした。特に、小林幸雄博士、金子博博士、小杉信博士、武川直樹氏、尺長健博士にはお世話になりました。厚くお礼申し上げます。

末筆となりましたが、家庭を省みなかった夫を終始励まし、また本論文の草稿に目を通して適切なコメントをくれた妻英子に心より感謝いたします。

参考文献

[阿部,1981] 阿部,金田,寺島,津田,渡辺 ”プラズマ科学における情報処理”, 情報処理, Vol.22, No.1, pp.10-18, Jan. 1981

[秋本,1986] 秋本,間瀬 ”画素選択型光線追跡法”, 信学論(D), Vol.J69-D, No.12, pp.1943-1952, Dec. 1986

[Amanatides,1984] J.Amanatides "Ray Tracing with Cones", Comput. Graphics, Vol.18, No.3, pp.129-135, Jul. 1984

[Amanatides,1987] J.Amanatides "Realism in Computer Graphics: A Survey", IEEE CG&A, Vol.7, No.1, pp.44-56, Jan. 1987

[AMD] "AMSYS29/10 Micro Program Support Software User's Manual", AMD

[Appel,1968] A. Appel "Some techniques for shading machine renderings of solids" AFIPS 1968 SJCC, 32, pp.37-45, 1968

[Arvo,1987] J. Arvo, D. Kirk "Fast Ray Tracing by Ray Classification", Comput. Graphics, Vol.21, No.4, pp.55-64, Jul. 1987

[Blinn,1976] J. F. Blinn, M. E. Newell "Texture and Reflection in Computer Generated Images", CACM, Vol. 19, No. 10, pp.542-547, Oct. 1976

[Blinn,1978] J.F.Blinn "Simulation of Wrinkled Surface", Comput. Graphics, Vol.12, No.3, pp.286-292, Aug. 1978

[Clark,1976] J.H.Clark "Hierachical Geometric Models for Visible Surface Algorithms", CACM, Vol.19, No.10, pp.547-554, Oct. 1976

[Clark,1982] J.H.Clark "The Geometry Engine: A VLSI Geometry System for Graphics", Comput. Graphics, Vol.16, No.3, pp.127-133, Jul. 1982

[Cook,1984] R.L.Cook, T.Porter, L.Carpenter "Distributed Ray Tracing", Comput. Graphics, Vol.18, No.3, pp.137-145, Jul. 1984

[Cook,1986] R.L.Cook "Stochastic Sampling in Computer Graphics", ACM Trans. on Graphics, Vol.5, No.1, pp.51-72, Jan. 1986

[出口,1984] 出口,西村,吉村,河田,白川,大村 ”コンピュータグラフィックスシステムＬＩＮＫＳ－１における画像生成の高速化手法”, 情処学会論文誌, Vol.25, No.6, pp.944-952, Nov. 1984

[Dippe,1984] M.Dippe, J.Swensen "An Adaptive Subdivision Algorithm and Parallel Architecture for Realistic Image Synthesis", Comput. Graphics, Vol.18, No.3, pp.149-158, Jul. 1984

[Fu,1989] K.S.Fu, R.C.Gonzalez, C.S.G.Lee 著,本多,遠山,古屋,金子 訳 "ロボティクス", 日刊工業新聞社, 平成元年

[Fuchs,1989] H.Fuchs, J.Poulton, J.Eyles, T.Greer, J.Goldfeather, D.Ellsworth, S.Molnar, G.Turk, B.Tebbs, L.Israel "Pixel-Planes 5: A Heterogeneous Multiprocessor Graphics System Using Processor-Enhanced Memories", Comput. Graphics, Vol.23, No.3, pp.79-88, Jul. 1989

[Fujimoto,1986] A. Fujimoto, T. Tanaka, K. Iwata "ARTS: Accelerated Ray-tracing System", IEEE CG&A, Vol. 6, No. 4, pp.16-26, Apr. 1986

[Glassner,1984] A. S. Glassner "Space subdivision for fast ray tracing", IEEE CG&A, Vol. 4, No. 10, pp.15-22, Oct. 1984

[Glassner,1989] A. S. Glassner Ed. "An Introduction to Ray Tracing", Academic Press, 1989

[萩原,1977] 萩原 "マイクロプログラミング", 産業図書, 1977

[Hall,1983] R.A.Hall, D.P.Greenberg "A Testbed for Realistic Image Synthesis", IEEE CG&A, Vol.3, No.10, pp.10-20, Nov. 1983

[Hanrahan,1983] P. Hanrahan "Ray Tracing Algebraic Surfaces", Comput. Graphics, Vol. 17, No. 3, pp.83-90, Jul. 1983

[橋本,1987] 橋本,間瀬,秋本 "ボーダー・レイトレーシング法",信学技報, PRU86-110, Feb. 1987

[Heckbert,1984] P.S.Heckbert, P.Hanrahan "Beam Tracing Polygonal Object", Comput. Graphics, Vol. 18, No. 3, pp.119-127, Jul. 1984

[日高,1985] 日高,平井,中瀬,浅原,鷲島 "マルチコンピュータ画像生成システムＭＣ－１", 情処研資, CA58-5, 1985

[Hoshino,1986] T.Hoshino "An Invitation to the World of PAX", COMPUTER, Vol. 19, No. 5, pp.68-79, May 1986

[Kajiya,1982] J.T.Kajiya "Ray tracing parametric patches", Comput. Graphics, Vol. 16, No. 3, pp.245-254, Jul. 1982

[Kajiya,1983] J.T.Kajiya "New Techniques for Ray Tracing Procedurally Defined Objects", ACM Trans. on Graphics, Vol. 2, No. 3, pp.161-181, Jul. 1983

[上村,1978] 上村 "プラズマの輸送現象と計算機シミュレーション", 第１７回プラズマ若手夏の学校テキスト, pp.163-210, Aug. 1978

[加藤,1976] 加藤,苗村 "並列処理計算機", オーム社, 1976

[河合,1988] 河合, 山下, 大野, 他 "並列画像生成システムＬＩＮＫＳ－２のアーキテクチャ", 情処学会論文誌, Vol.29, No.8, pp.729-740, Oct. 1988

[Kay,1986] T.L.Kay, J.T.Kajiya "Ray Tracing Complex Scenes", Comput. Graphics, Vol.20, No.4, pp.269-278, 1986

[Kernighan,1978] B. W. Kernighan, D. M. Ritchie "The C Programming Language", Prentice-Hall, 1978

[久保田,1972] 久保田 "応用光学", 岩波全書２４５, 岩波書店, 1972

[Lawrie,1975] D.H.Lawrie "Access and Alignment of Data in an Array Processor", IEEE Trans. Comput., Vol.C-24, No. 12, pp.1145-1155, Dec. 1975

[Levinthal,1984] A.Levinthal, T.Porter "Chap - A SIMD Graphics Processor", Comput. Graphics, Vol.18, No. 3, pp.77-82, Jul. 1984

[松本,1983] 松本,村上 "Octreeデータ構造を用いたRay-Tracing法", 第２７回情報処理学会全国大会講演論文集, 2M-6, pp.1535-1536, 1983

[Mitsuya,1987] E.Mitsuya, Y.Tamamura, T.Akimoto "A Multiprocessor System for Three Dimensional Graphics", Proc. IEEE Conf. Comput. Design, pp.316-319, Oct. 1987

[中瀬,1985] 中瀬,日高,西村,宮崎,野口,鷲島 "高速浮動小数点演算機能を持つユニットコンピュータ・ＭＣのアーキテクチャ", 情処研資, CA58-6, 1985

[成瀬,1986a] 成瀬,吉田,高橋,金子 "グラフィックス計算機ＳＩＧＨＴの基本構想", 信学論(D), Vol.J69-D, No.3, pp.474-476, Mar. 1986

[成瀬,1986b] 成瀬,吉田,高橋 "ＣＧ計算機ＳＩＧＨＴの性能評価", 情報処理学会第３３回（昭和６１年後期）全国大会講演論文集, 1P-1, pp.1551-1552, 1986

[成瀬,1987] 成瀬,吉田,高橋 "ＣＧ計算機ＳＩＧＨＴの高水準言語ＳＩＧＨＴ／Ｃ", 昭和６２年度電子通信学会情報・システム部門全国大会講演論文集, 95, 1987

[Naruse,1987] T.Naruse,M.Yoshida,T.Takahashi,S.Naito "SIGHT - A Dedicated

Computer Graphics Machine", Computer Graphics Forum, Vol.6, No.4, pp.327-334, 1987

[成瀬,1988] 成瀬,吉田,高橋 "グラフィックス専用計算機ＳＩＧＨＴ－－光線追跡法の並列処理－－", 情処学会コンピュータアーキテクチャシンポジウム論文集, pp.217-225, May 1988

[成瀬,1989a] 成瀬,吉田,高橋 "ＳＩＧＨＴ上の３次元ベクトル演算に関する考察", 情処学会計算機アーキテクチャ研究会資料, CA77-7, 1989

[成瀬,1989b] 成瀬,國島,斎藤 "Bounding Volumeの構成法に関する考察", 信学技報, PRU89-65, Oct. 1989

[成瀬,1990a] 成瀬 "３次元ベクトル演算の並列実行に関する考察", 情報処理, Vol.31, No.5, pp.643-649, May 1990

[成瀬,1990b] 成瀬,斉藤,國島 "ｎ球包含問題の近似解法",信学論(D-II), Vol.J73-D-II, No.10, pp.1792-1795, Oct. 1990

[成瀬,1991a] 成瀬,吉田,高橋 "ＣＧ計算機ＳＩＧＨＴの性能解析", 信学論(D-I), Vol.J74-D-I, No.2, pp.166-174, Feb. 1991

[Naruse,1991] T.Naruse "Parallel Execution of 3 Dimensional Vector Operations", 情報処理学会欧文論文誌掲載予定

[成瀬,1991b] 成瀬,新谷,斉藤 "動的部分木を用いた光線追跡法",電子情報通信学会論文誌投稿中

[日経,1987] "今月の表紙", 日経ＣＧ, No.8, p45, 日経マグロウヒル, May 1987

[Nishimura,1984] H.Nishimura, H.Ohno, T.Kawata, I.Shirakawa, K.Omura "LINKS-1: A Parallel Pipelined Multimicrocomputer System for Image Creation", Proc. 10th Annu. Int'l Symp. on Comput. Arch., pp.387-394, 1984

[西村,1985a] 西村,平井,河合,河田,白川,大村 "分布関数による物体モデリングと画像生成の一手法", 信学論(D), Vol.J68-D, No.4, pp.718-725, Apr. 1985

[西村,1985b] 西村,出口,辰巳,河田,白川,大村 "コンピュータグラフィックスシステムＬＩＮＫＳ－１における並列処理の性能評価", 信学論(D), Vol.J68-D, No.4, pp.733-740, Apr. 1985

[Phong,1975] B.T.Phong "Illumination for Computer Generated Pictures", CACM, Vol.18, No.6, pp.311-317, 1975

[Potmesil,1989] M.Potmesil, E.M.Hoffert "The Pixel Machine: A Parallel Image Processor", Comput. Graphics, Vol.23, No.3, pp.69-78, Jul. 1989

[Preparata,1985] F.P.Preparata, M.I.Shamos "Computational Geometry An Introduction", Springer-Verlag, 1985

[Rappaport,1989] D.Rappaport "Computing the Furthest Site Voronoi Diagram for a Set of Discs", in Lecture Note in Computer Sciene 382 Algorithms and Data Structures, pp.57-66, Springer-Verlag, 1989

[Roth,1982] S.D.Roth "Ray Casting for Modelling Solids", CGIP, No.18, No.2, pp.109-114, Feb. 1982

[Samet,1984] H.Samet "The Quadtree and Related Hierarchical Data Structures", ACM Computing Survey, Vol.16, No.2, pp.187-260, Jun. 1984

[佐竹,1971] 佐竹 "行列と行列式", 裳華房, 1971

[Sato,1985] H.Sato, M.Ishii, K.Sato, M.Ikesaka, H.Ishihata, M.Kakimoto, K.Hirota, K.Inoue "Fast Image Generation of Constructive Solid Geometry Using a Cellular Array Processor", Computer Graphics, Vol. 19, No. 3, pp.95-102, Jul. 1985

[Sederberg,1984] T.W.Sederberg, D.C.Anderson "Ray Tracing of Steiner Patches", Comput. Graphics, Vol.18, No.3, pp.159-164, Jul. 1984

[Shinya,1987] M.Shinya, T.Takahashi, S.Naito "Principles and Application of Pencil Tracing", Comput. Graphics, Vol.21, No.3, pp.45-54, Jul. 1987

[新谷,1989] 新谷,高橋 "光束追跡の理論と誤差解析", 信学論(D-II), Vol. J72-D-II, No. 12, pp.2000-2011, Dec. 1989

[新谷,1990] 新谷 "レンダリング技術の研究動向", 信学技報, IE89-94, Feb. 1989

[正田,1985] 正田,水野,山下,西村,白川,大村 "画像生成用マルチコンピュータシステムＬＩＮＫＳ－２について", 信学技報, CAS84-204, Feb. 1985

[Sutherland,1974] I.E.Sutherland, et.al. "A Characterization of Ten Hidden-Surface Algorithms", Comput. Survey, Vol.6, No.1, pp.1-55, Mar. 1974

[Sweeney,1986] M.Sweeney and R.H.Bartels "Ray Tracing Free-form B-spline Surfaces", IEEE CG&A, Vol.6, No.2, p.41, Feb. 1986

[高木,1984] 高木,横井,鶴岡,木村,三宅 "色分散を考慮した光線追跡の一方法",情処学

会グラフィックスとＣＡＤシンポジウム論文集，pp.81-87，1984

[高橋,1985] 高橋,成瀬,吉田,内藤 ”グラフィックスプロセッサＳＩＧＨＴの基本構成――ファームウェア構成を中心として――”，信学技報，PRU85-53，Dec. 1985

[Takahashi,1987] T.Takahashi,M.Yoshida,T.Naruse "Architecture and Performance Evaluation of the Dedicated Graphics Computer : SIGHT", Proc. Compint'87 (MONTECH'87), pp.153-160, 1987

[Thalmann,1987] N. Magnenat-Thalmann, D. Thalmann "Image Synthesis Theory and Practise", p172, Springer-Verlag, 1987

[Weghorst,1984] H.Weghorst, G.Hooper, D.P.Greenberg "Improved Computational Methods for Ray Tracing", ACM Trans. on Graphics, Vol.3, No.1, pp.52-69, Jan. 1984

[Weitek,1984] "WEITEK High Speed Digital Arithmetic VLSI Data Book", 1984

[Whitted,1980] T.Whitted "An Improved Illumination Model for Shaded Display", Comm. ACM, Vol. 23, No. 6, pp.343-349, June, 1980

[山内,1972] 山内,宇野,一松 ”電子計算機のための数値計算法III”，培風館，1972

[山本,1983] 山本 ”The 3 Dimensional Computer Graphics パソコンによる３次元グラフィックスの実際”，ＣＱ出版社，1983

[安田,1984] 安田,横井,鳥脇,鶴岡,三宅 ”透明物体表示のための改良光線追跡法”，情処学会論文誌，Vol.25，No.6，pp.953-959，1984

[安田,1985] 安田,横井,鶴岡,三宅 ”透明物体表示のための改良光線追跡法(2)”，情処学会論文誌，Vol.26，No.4，pp.591-599，1985

[横井,1986] 横井,鳥脇 ”画像処理アルゴリズムの最新動向（１３） コンピュータグラフィックスにおける光線追跡法”，O plus E，No.79，pp.94-106，Jun 1986

[吉田,1985] 吉田,成瀬,高橋,内藤 ”グラフィックス計算機ＳＩＧＨＴの基本構成”，情処研資，CA60-4，Dec. 1985

[吉田,1988] 吉田,成瀬 ”コンピュータグラフィックス用プロセッサの動向”，情報処理，Vol.29，No.10，pp.1109-1114，Oct. 1988

[Yoshida,1991] M.Yoshida, T.Naruse, T.Takahashi "A Dedicated Graphics Processor SIGHT-2" in R.L.Grimsdale W.Straser(Eds.) Advances in Computer Graphics Hardware IV, pp.151-169, Springer-Verlag, 1991