画素並列法によるレイトレーシングの高速化

山森一人[1)] 園田光[2)] 吉原郁夫[3)] 相川勝[4)]

Perfomance Evaluation of Parallel Ray-tracing Algorithm using Pixel-base Parallelism

Kunihito YAMARIMORI Hikaru SONODA Ikuo YOSHIHARA Masaru AIKAWA

Abstract

Recently, computer graphics are used in various fields like movies, commercial films, etc. To generate high quality graphics, it takes much coumputing time for rendering process. So parallel computers such as cluster system are usually uesd to generate high quality graphics. In this paper, we discuss on the advantages and the disadvantages of there parallel ray-tracing algorithms; object-base parallelization, space-bace parallelization and pixel-base parallelization. Then we implement the parallel ray-tracing algorithm using pixel-base parallelization into a cluster system that has 44 processors, and evaluate the performance of the parallel ray-tracing algorithm.

Key Words:
Ray-tracing, Pixel-base parallelism, Computer graphics

1 はじめに

近年、コンピュータグラフィクス（CG）は映画、コマーシャル、プレゼンテーションなど、さまざまな分野で使用されており、その応用範囲は多岐にわたっている。また、光の屈折や反射を表現できるレイトレーシング、拡散光による複雑な相互反射を考慮し柔らかな光環境を描くラジオシティ等、様々な表現手法が存在する。しかし、複雑で写実的な作品を作成しようとすれば多くの時間がかかるのが問題となっている。

CG の製作は大きくモデリング処理とレンダリング処理に分けられるが、多大な演算処理能力を必要とするのはレンダリング処理であり、レンダリングの優劣によって画像のクオリティが左右され、クオリティの高い画像を求めればそれだけ演算量は多くなる。スーパーコンピュータのような演算処理能力の高い計算機を用いることができれば短時間にレンダリングを終わらせる事ができるが、スーパーコンピュータの利用は導入や維持のコストが高く現実的でない。そこで、近年高性能化が著しいパーソナルコンピュータ（PC）を複数台ネットワークで結合し、一種の並列計算機として用いる方法が注目されている。PC を利用した並列計算機を用いてレンダリングを並列に実行する事ができれば、処理時間を短縮する事ができコストも下げる事ができる。

並列計算機による並列レンダリングは処理時間短縮に非常に有効な手法であり、多くの研究が行われている。平田ら [1)] は、超並列計算機における最適な画像生成アルゴリズムについて、負荷分散、耐故障性、総合的な効率など様々な観点から検討し、画素並列法とオブジェクト並列法を組み合わせた並列アルゴリズムを提案している。また、水谷ら [2)] は、平田らの並列アルゴリズムに加え、新たに屈折光や反射光等の計算を並列化する手法についても検討し、これらを組み合わせた並列アルゴリズムを提案している。

本論文では、並列レイトレーシング法であるオ

[1)] 情報システム工学科准教授
[2)] 現在 (株) 川崎重工業
[3)] 情報システム工学科教授
[4)] 工学部教育研究支援技術センター技術職員

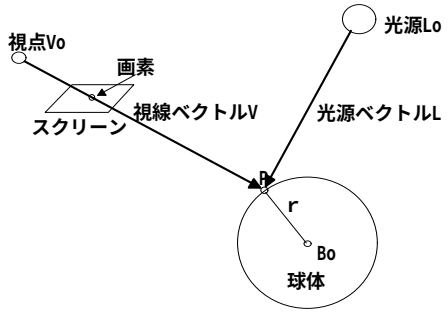

図. 1　球体の描画

ブジェクト並列法、空間並列法、画素並列法について検討を行い、その中で比較的通信負荷の小さい画素並列法を実際の並列計算機上に実装し、その処理性能について議論することを目的とする。

2　レイトレーシング

レイトレーシングは光線追跡法とも呼ばれ、光源からの光がオブジェクトに当たり反射光が目に到達することによってオブジェクトが知覚される実際とは逆に、視点から画素を通る視線を追跡して描画を行う手法である。つまり、全てのオブジェクトに対して視線とオブジェクトとの交点を調べ、そのうち、最も視点に近い交点を可視点とし、可視点の輝度を計算する。光源から発せられる光線を１次光線、反射、屈折後の光線を２次光線と呼ぶ。レイトレーシングにより影付け、鏡面反射による映り込み、透明オブジェクトによる光の屈折を表現することができる。

図 1 に示すような球体の描画について考える。まず、視点 V_0 とスクリーン上の画素との位置関係から、視線ベクトル $\boldsymbol{V}$ を求める。そして、視線が球体と交わる場合、光源との位置関係から交点 P の色を求めて画素の色を決定するのが基本的な流れである。

視線が球体と交わるか判定するには、まず、視線ベクトル $\boldsymbol{V}$ と交点 P との関係、および球体の半径 r の関係から (1) 式、(2) 式を求める。

$$\boldsymbol{V_0} + t \cdot \boldsymbol{V} = \boldsymbol{P}, \tag{1}$$

$$(\boldsymbol{P} - \boldsymbol{B_0}) \cdot (\boldsymbol{P} - \boldsymbol{B_0}) = r^2. \tag{2}$$

ここで t は視点から交点までの距離を表し,$\boldsymbol{P}$、$\boldsymbol{V_0}$、$\boldsymbol{B_0}$ は原点から各点までのベクトルを表す。

(1) 式を (2) 式に代入すると (3) 式が得られ、球体と視線が交わるかの判別式 d は (4) 式となる。

$$t^2 + 2t\cdot\boldsymbol{V}\cdot(\boldsymbol{V_0}-\boldsymbol{B_0}) + (\boldsymbol{V_0}-\boldsymbol{B_0})\cdot(\boldsymbol{V_0}-\boldsymbol{B_0}) - r^2 = 0, \tag{3}$$

$$d = \{\boldsymbol{V}\cdot(\boldsymbol{V_0}-\boldsymbol{B_0})\}^2 - (\boldsymbol{V_0}-\boldsymbol{B_0})\cdot(\boldsymbol{V_0}-\boldsymbol{B_0}) + r^2. \tag{4}$$

d が 0 以上の場合に球体と交わり、そのときの視点に最も近い交点 P の座標は (5) 式で求められる。

$$t = -\boldsymbol{V}\cdot(\boldsymbol{V_0} - \boldsymbol{B_0}) - \sqrt{d}. \tag{5}$$

視線が球体と交わる場合、交点 P と光源 L_0 との位置関係から (6) 式で光源ベクトル $\boldsymbol{L}$ を求め、球体の中心点 B_0 と交点 P との位置関係から (7) 式で法線ベクトル $\boldsymbol{N}$ を求める。

$$L = \frac{\boldsymbol{P} - \boldsymbol{L_0}}{|\boldsymbol{P} - \boldsymbol{L_0}|}, \tag{6}$$

$$N = \frac{\boldsymbol{P} - \boldsymbol{B_0}}{|\boldsymbol{P} - \boldsymbol{B_0}|}. \tag{7}$$

視線ベクトル $\boldsymbol{V}$、光線ベクトル $\boldsymbol{L}$、法線ベクトル $\boldsymbol{N}$ からオブジェクトの色を Phong の反射モデル [3] である (8) 式により計算する。

$$\begin{aligned} C &= (s\cdot k_d\cos\alpha + k_e)\cdot C_s + \\ &\quad s\cdot k_s\cos^n\beta\cdot C_w, \\ \cos\alpha &= -\boldsymbol{L}\cdot\boldsymbol{N}, \\ \cos\beta &= 2(\boldsymbol{L}\cdot\boldsymbol{N})(\boldsymbol{N}\cdot\boldsymbol{V}) - \boldsymbol{L}\cdot\boldsymbol{V}. \end{aligned} \tag{8}$$

ここで、C は描画に使用する色、C_s はオブジェクトの表面色、C_w は白色の成分、k_d は拡散反射係数、k_e は拡散反射係数、k_s は鏡面反射係数を表す。

レイトレーシングにおいて、処理の 95%以上がオブジェクトと光線との交点計算であるということが知られており [2]、交点計算を並列に行うことにより処理の高速化が実現できる。本論文においても、交点計算を並列に行うことで処理の高速化を図る。交点計算並列化の観点として、オブジェクト並列法、空間並列法、画素並列法の 3 つを次節で比較検討する。

2.1　並列レイトレーシング法

2.1.1　オブジェクト並列法

オブジェクト並列は、光線とオブジェクトとの交点計算処理はオブジェクト毎に独立しているという観点に基づく並列手法である。図 2 に表され

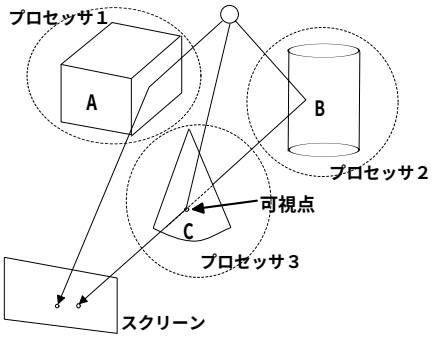

図. 2　オブジェクト並列法の例

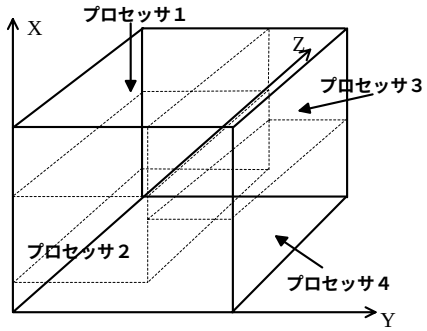

図. 4　空間並列法の例

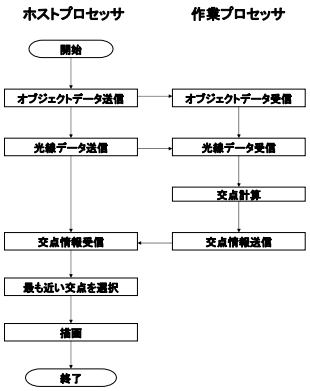

図. 3　オブジェクト並列法の処理の流れ

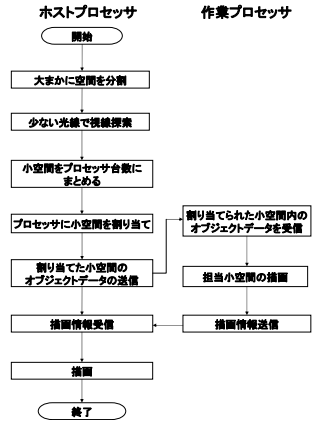

図. 5　空間並列法の処理の流れ

るように、オブジェクトの一つ一つまたは複数のオブジェクトをまとめたグループを各プロセッサに割り当てる。次に、光線ベクトル $\boldsymbol{L}$ を全てのプロセッサに配信し、それぞれのプロセッサが担当するオブジェクトとの交点計算を並列に行う。交点が求まると、それを全体を統括するプロセッサに送り、視点に最も近い交点を選択する。オブジェクト並列法の処理の流れを図 3 に示す。対象とするオブジェクトの大きさや形状が均一で、多数のオブジェクトが画面内に存在した場合、負荷を均等に分散できるためプロセッサ台数に比例した効率が得られる。一方、オブジェクト並列の問題として、通信量が多くプロセッサ間の通信がボトルネックになるため、全てのプロセッサを利用するだけのオブジェクトが存在したとしても十分な処理速度向上が得られないことあることが挙げられる。

2.1.2　空間並列法

空間並列法は図 4 のように、描画する空間を小空間に分け、その小空間に対してプロセッサを割り当てる方法である。分割した空間の中を光線が通り抜けたり、反射によって他のプロセッサの担当している空間に光線が届くという事が起きるため、各プロセッサ間で光線をメッセージとしてやりとりする。最終的には、全光線をホストプロセッサが回収して描画する。空間並列法では、オブジェクトが存在しない領域を割り当てられるプロセッサがありうるなど、プロセッサ間で著しい負荷の不均一が発生するおそれがある。そこで、安部ら [4] により次のような方法が提案されている。はじめに、オブジェクト空間をあらかじめ任意の大きさの小空間に分け、少ない光線でレンダリングを行って、どの小空間にどのくらい光線が入ったかをカウントし、オブジェクト空間全体での大まかな負荷を測定する。そして、負荷が同じになるよう小空間をプロセッサの台数分にまとめ、まとめた領域に対してプロセッサを割り当てる。空間並列法の処理の流れを図 5 に示す。空間並列法の利点として、上記の手法によりある程度の負荷の分散が可能となるため、負荷の偏りが少なくなるということが挙げられる。一方、問題点としては、プロセッサ間の光線メッセージの通信によるオーバーヘッドのために、プロセッサ台数を増やしても台数効果が表れにくいことが挙げられる。

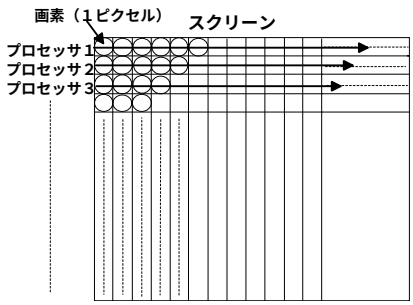

図. 6　画素並列法の例

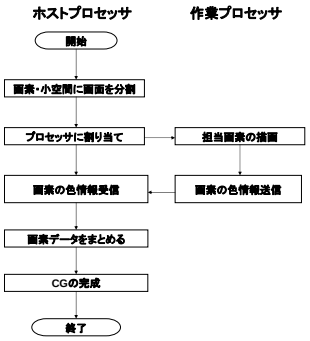

図. 7　画素並列法の処理の流れ

2.1.3　画素並列法

画素並列法は図 6 のように、画素単位で独立して光線の処理を行うことができるレイトレーシングの特徴を活かせる並列化手法である。各画素、または画面をいくつかに分割し、画素または小画面に対してプロセッサを割り当てる。オブジェクトデータは分割せず全てのプロセッサがそのコピーを持ち、屈折光、反射光等の処理も 1 次光線を割り当てられたプロセッサが担当する。担当した画素の処理を終えると、ホストプロセッサに担当画素の描画色を送り 1 枚の CG にまとめる。画素並列法の処理の流れを図 7 に示す。画素並列法では、全てのプロセッサが物体のデータを持っているため、プロセッサ間で光線データを交換する必要がない。そのため、他の手法よりも通信が少なく通信量によるボトルネックが起きにくいという利点がある。一方、問題点としては、全てのプロセッサがオブジェクトデータのコピーを持つことから、メモリ要求量が大きいことが挙げられる。

本研究ではギガビットイーサネットで接続されたクラスタ型並列計算機を利用することから、各プロセッサ間での通信が少なく使用するプロセッサ台数に比例して処理時間の短縮が期待できる画素並列法を用いることにした。

表. 1　並列処理実験環境

CPU	Intel Xeon 2.8GHz
Memory	2GB
OS	Cent OS5(x86-64)
プロセッサ数	22 ノード、プロセッサ 44 台
ネットワーク	ギガビットイーサネット接続
ホームディレクトリ	NFS による共有
CG の画像サイズ	横 640、縦 480(pixel)

3　実験と考察

3.1　実験環境

並列処理実験環境を表 1 に示す。並列化のための通信ライブラリとして MPICH2[5] を用いた。MPICH2 とはプロセス間の通信をメッセージのやりとりによって行うメッセージ通信ライブラリである。現在、並列プログラミングの規格として最も広く使われており、1 対 1 通信だけではなく、集合通信も一つの関数で実行できる等数多くの有用な機能を持ち、多くの並列計算機や LAN 環境で高い性能を実現できる [6]。

3.2　並列処理時間

処理対象とした CG（コンピュータグラフィクス）を図 8 と図 9 に示す。図 8 を CG1 とし、図 9 を CG2 とする。図 8 は、オブジェクトが中心に一つ配置されていて、並列処理を行った際にプロセッサにより負荷の偏りが発生する例として採用した。図 9 は、松や池、灯篭等が画面内に分散して配置されているため、並列処理を行った際に負荷が比較的均等に分散される例として採用した。本研究では 2 種類の複雑さの異なる画像を対象に、画像のクオリティを決める光線の屈折追跡回数を 20 回として画像生成を行った。画素の割り当ては、行方向に画面をプロセッサ数だけ分割し小画面を各プロセッサに割り当てる小画面分割方式とした。画像分割時に行数がプロセッサ台数で割り切れず余りが生じた場合には、プロセッサに 1 行づつ順々に担当行を増やすことにした。

図 8 に示した CG を 1 台のプロセッサでレンダリングしたところ、全処理時間は 4544 秒であった。その内訳は、初期設定に 0.48 秒、オブジェクトデータの登録に 5.13 秒、交点計算に 4538 秒となり、交点計算が処理時間の 99%を占めていることが分かった。

並列処理実験では、2 台ずつプロセッサ台数を増やし、2 台から 44 台までの処理時間を測定した。用いたプロセッサ台数を変化させた時の処理

図. 8　処理対象 CG(灯篭)

図. 9　処理対象 CG(池と松と灯篭)

時間を図 10 に示す。図 10 から分かる通り、処理時間はプロセッサ台数を増やす毎に短縮できている事がわかる。しかし、プロセッサ台数が増えるにつれて処理時間の向上は緩やかになっている。プロセッサ台数を変化させたときの、(9) 式で定義される速度向上度を図 11 に示す。

$$速度向上度 = \frac{逐次処理時間（秒）}{並列処理時間（秒）}. \qquad (9)$$

図 11 から、プロセッサ台数の増加につれ右上がりに速度向上度は大きくなるが、CG1 は CG2 に比べその増加は緩やかとなっている。

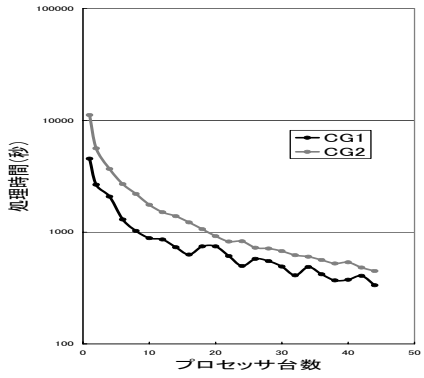

図. 10　プロセッサ台数に対する処理時間の変化

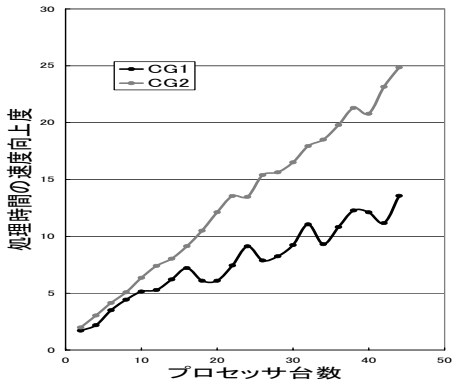

図. 11　プロセッサ台数の変化による処理時間の速度向上度の推移

3.3　考察

CG1 において 1 台のプロセッサを基準として 44 台のプロセッサを使用したときに最大 14 倍の処理速度向上となり、CG2 において最大 25 倍の処理速度向上となった。速度向上度でみると、1 台から 2 台にプロセッサを増やしたときが一番大きく向上しており、2 倍にかなり近い数値となっている。言い替えると、2 台目以降では使用したプロセッサ台数に見当った速度向上度が完全には得られていない。

台数に比例した速度向上となっていない理由としては負荷の不均一が発生しているためと考えられる。CG1 を 14 台、16 台、18 台で処理したときの各プロセッサの処理終了時間を図 12 に示す。図 12 より、各プロセッサが等しい数の画素数を割り当てられる 16 台のプロセッサ使用時には、全プロセッサがほぼ等しい処理時間となっている。一方、割り当てられる画素数がプロセッサにより異なる 14 台、18 台使用時にはプロセッサによって負荷の不均一が発生している事が分かる。これは、単純な担当画素数の違いに加え、割り当てられた領域が複雑な画像となっているか否かや、遅いプロセッサが生成した画像をホストプロセッサに集めるまでの待ち時間、ホストプロセッサが画像データをファイルに書き込む時間の影響などが考えられる。そこで、各プロセッサにおける通信時間と、ホストプロセッサでのファイル書き込み時間についても調査した。

CG1 を 16 台で処理したときの各プロセッサの通信時間を図 13 に示す。図 13 より、プロセッサによって担当する小画面のデータ送信に時間の差

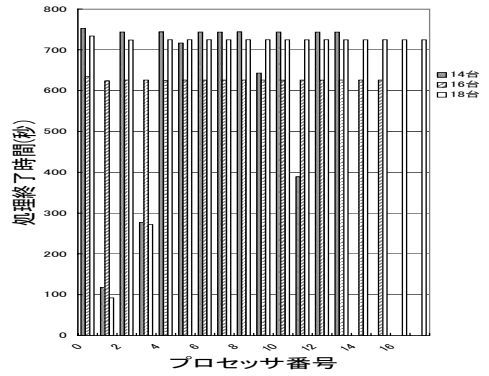

図. 12 並列処理時の各プロセッサのレイトレーシング時間の違い

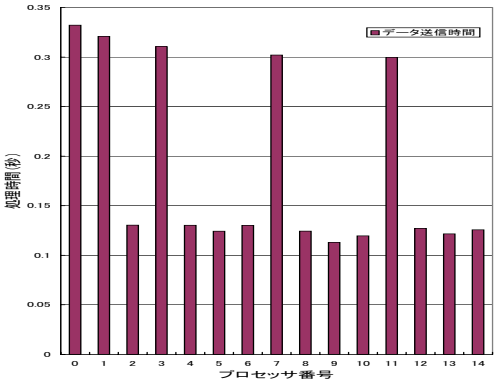

図. 13 各プロセッサのデータ送信時間

が発生していることがわかる。しかし、データ送信時間は一番時間のかかっているプロセッサにおいても 0.32 秒程度である。全処理時間の 630 秒の約 $\frac{1}{2000}$ の時間であるため、通信はボトルネックになっていないことがわかる。一方、全てのプロセッサが送信してきた小画面のデータをホストプロセッサがファイルに書き込む時間は 0.29 秒であった。ファイルへの書き込みにシステムがバッファリングを行っているため、書き込み時間の影響はほとんど無視できることが分かった。これらのことから、負荷不均衡が生じる原因はほとんどがプロセッサに割り当てられた画素数に因るものであることがわかった。これは、図 11 で示したように負荷分散が生じやすい CG1 において速度向上度が波打っており、余りの行がないようプロセッサを割り当てられている時に速度向上度が大きくなっていることからも間接的に説明できる。

4 おわりに

本論文では、コンピュータグラフィクスの主要な描画手法であるレイトレーシングでのレンダリング処理において、ほとんどの処理時間を占める視線とオブジェクトとの交点計算処理時間を短縮することを目的とし、画素並列法を用いてその並列処理時間を実際の並列計算機により計測した。最大 44 台のプロセッサを用いたとき、比較的単純な CG では単一プロセッサ私用時の 14 倍、比較的複雑な CG では 25 倍の処理速度向上が得られた。使用したプロセッサ台数に応じた速度向上が得られなかったのは、小画面分割方式を採用したことによる各プロセッサでの負荷不均一と考えられる。

今後の課題としては、画素並列法の分割手法を 1 画素毎に担当プロセッサを代えて負荷を均等にすることや、他の並列手法と組み合わせて処理時間のさらなる短縮を図ることがあげられる。

参考文献

[1] 平田貴光, 安部殻, 大野義夫, “レイトレーシングの並列化”, 情報処理学会研究報告, Vol. 1994, No. 41, pp. 25–32 (1994).

[2] 水谷政美, 中島正之, “超並列計算機を用いた並列 CG アルゴリズムに関する研究”, 情報処理学会研究報告, Vol. 1994, No. 41, pp. 17–24 (1994).

[3] 小笠原祐治, C++による簡単実習　3 次元 CG 入門 (第 2 版), 森北出版株式会社 (2004).

[4] 安部殻, 大野義夫, 西田友是, 近藤邦雄, 中島正之, “並列グラフィクスアルゴリズムのサーベイ”, 情報処理学会研究報告, Vol. 1994, No. 41, pp. 9–16 (1994).

[5] 千葉則茂, 土井章男, 3 次元 CGno 基礎と応用, サイエンス社 (2004).

[6] ウィリアム・クロップ, ユーイング・ラスク, ラジープ・タークル, 実践 MPI- 2 , ピアソン・エデュケーション (2002).