修 士 論 文

モンテカルロ光線追跡法を利用した並列レンダリングに関する研究

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

清水 昭尋

2006 年 3 月

修 士 論 文

モンテカルロ光線追跡法を利用した並列レンダリングに関する研究

指導教官　井口寧 助教授

審査委員主査　井口寧 助教授
審査委員　松澤照男 教授
審査委員　田中清史 助教授

北陸先端科学技術大学院大学
情報科学研究科情報システム学専攻

410062 清水 昭尋

提出年月: 2006 年 2 月

目 次

図 目 次

表 目 次

第1章　序論

1.1　研究の背景と目的

今日，コンピュータグラフィックスにおける画像合成技術は，映画や建築，ゲームやシミュレーションといった様々な応用分野で利用されている．その中でも，映画などに用いられている画像合成技術は，本物なのか CG なのかわからないほどの画像合成を可能にしている．このような写実的画像合成はコンピュータグラフィックスにおける一大分野となっており，様々な研究がおこなわれている．特に光線追跡法を基礎とした画像合成アルゴリズムは写実的画像合成に欠かせないものであり特に重要になっている．

現在，写実的画像合成において重要な鍵となるのは，モンテカルロ光線追跡法というモンテカルロ積分を用いたポイントサンプリングアルゴリズムである．モンテカルロ光線追跡法において品質の高い画像を得るためには，膨大な計算コストがかかることが知られており，これを高速に実行できるということは非常に有用である．高速化の方法としては，集積化や並列化といったものから，新しい洗練されたアルゴリズムの開発といったものが挙げられる．アルゴリズムの洗練とは，いかに少ない計算量で品質の良い画像を得られるかというもので，この分野では素晴らしい研究成果が上げられている．しかしながら，集積化，並列化といった分野では，従来の古典的光線追跡法におけるものがほとんで，モンテカルロ光線追跡法を考慮に入れたものはほとんどない．これは，古典的光線追跡法とモンテカルロ光線追跡法のアルゴリズムの共通点によるもので，こと並列化に関しては古典的光線追跡法と変わらない手法が適用可能である．しかしながら，モンテカルロ光線追跡法は古典的光線追跡に比べ大量の光線の処理が必要になるため，従来手法ではロードバランスを維持することができず効率的な並列化が難しいといった問題がある．

また並列化とは別の問題として，画像のサンプリング問題がある．モンテカルロ光線追跡法では，サンプリングが不十分であるとそれがノイズとして画像にあらわれる．そのため多くのサンプルを取る必要があり，その結果，計算コストが大きくなってしまうのである．このサンプリング処理は画像の各ピクセル毎におこなわれるが，任意のピクセルにおいてどれだけのサンプルを取ればピクセルが収束するのかがわからないために，ヒューリスティックにサンプル数を決定するのが現状である．このため，各ピクセルにおいて無駄なサンプリング処理が発生してしまう．

本研究では，モンテカルロ光線追跡法を視野にいれた，上記の二つの問題を解決するための分散平衡並列化アルゴリズムを提案する．これは，従来では各ピクセルに同数のサンプリングを処理をおこなっていたのを，解の収束の遅いピクセルの分散値に基づいて適応

的にサンプル数を変化させてやることにより，計算資源をそのピクセルに集中させようという手法である．これにより，無駄なサンプリングを省き，結果として同タイムスライスにおいて画像品質の高い並列処理がおこなえるシステムの構築を目的とする．そして，これを並列計算機上に実装し実験をおこない，画像の品質，レンダリング速度などを評価することにより，その有効性を示す．

1.2 本論文の構成

本論分の構成は次の通りである．

1 章では，研究の背景と目的について述べる．

2 章では，モンテカルロ光線追跡法を理解するためのレンダリングの基礎知識について概説する．

3 章では，レンダリングの並列化について，従来どのような目的でどのような並列化がおこなわれてきたのかを説明する．

4 章では，画像の品質に基づいたコスト評価を導入し，分散平衡並列化アルゴリズムを提案する．そして MPI ライブラリを用いた並列実装について解説する．

5 章では，提案したアルゴリズムの実行結果を示し，評価をおこなう．

6 章に本研究のまとめを述べる．

第2章　レンダリングの基礎

2.1　はじめに

レンダリングとは，与えられた幾何データ，光源データなどの数値から画像合成をおこなうことである．写実的画像合成のアルゴリズムとして最も広く認知されているものは，1980 年の光線追跡法 (ray tracing)[13] と 1984 年のラジオシティ法 (radiosity)[3] が挙げられる．これらのアルゴリズムが解く問題は，1986 年にレンダリング方程式 [5] として定式化され，光線追跡法もラジオシティ法もレンダリング方程式の限定的な解を求めることしかできないことがわかった．レンダリング方程式と同時に発表された経路追跡法 (path tracing) は，レンダリング方程式の完全な解法となるアルゴリズムである．

レンダリング方程式は積分方程式であり，その解法にはモンテカルロ積分が利用される．モンテカルロ積分を利用した光線追跡系のアルゴリズムはモンテカルロ光線追跡法と呼ばれ，経路追跡法もその中の一つになる．

この章では，大域照明の基本的な考え方を説明し，どのような方法で画像合成をおこなうのかを解説する．そして，レンダリングにおける基本方程式であるレンダリング方程式について説明し，実際に本研究で利用するモンテカルロ光線追跡法の概要を示す．

2.2　大域照明の理論

大域照明とは，物理学を基礎とした，シーン内に存在するすべての光エネルギーの伝播をシミュレーションすることである．反射や屈折といった光の挙動をシミュレーションし，目に入射する光のエネルギーを算出することにより像が出来上がる．光源から放出された光エネルギーは，オブジェクトのサーフェスにおいて複数回の反射を繰り返し，エネルギーを減衰させながら目へと到達する．

次節以降では，この光エネルギーの定義から解説し，大域照明がどのように定式化されているのかを述べる．そして，本研究で利用するモンテカルロ光線追跡法の一種である経路追跡法について説明する．

2.2.1　光のエネルギー

大域照明では，放射測定 (radiometry) で利用されるエネルギー単位を扱う．その中でも特に重要な単位は，放射束 (radiant flux)，放射照度 (irradiance)(放射照度は入射エネ

ルギーで出射エネルギーは放射発散度 (radiosity))，放射輝度 (radiance) の 3 種類である．特に放射輝度は，画像のピクセルそのものの値になる単位なので大切である．

放射束

放射束 Φ は光の放射エネルギー Q を時間微分することにより定義される．

$$\Phi = \frac{dQ}{dt} \tag{2.1}$$

単位はワットであり, 蛍光灯から太陽光に至るまで, すべての光の単位時間あたりのエネルギーを記述する．

放射照度 (放射発散度)

放射照度 E と放射発散度 B はサーフェス上の単位面積あたりの放射束 Φ として定義される．

$$E = \frac{d\Phi}{dA} \tag{2.2}$$

これはある単位面積あたりの領域に入射する放射束の値であって，領域の半球上のすべての方向から入射するエネルギーの総量である．放射照度は入射するエネルギーであるが，放射発散度は半球上に出射するエネルギーの総量で，この二つの値は同じである．

放射輝度

放射輝度 L はサーフェス上の投影単位面積あたり, 単位立体角あたりの放射束として定義される．

$$L = \frac{d^2\Phi}{d\omega dA\cos(\theta)} \tag{2.3}$$

投影単位面積は位置を，単位立体角は方向をあらわすので，任意の位置から任意の方向へ伝播するエネルギーを記述できる．放射輝度は照明アルゴリズムにおいて最も重要な単位で，光線追跡法など多くのアルゴリズムは放射輝度を伝播させる．

2.2.2 BRDF

光のシミュレーションをおこなうためには，前節で定義した光のエネルギーが物体との相互作用によりどのように散乱するのかがわかっていなければならない．この物体表面における光の散乱を局所照明 (local illumination) といい，さまざまな記述モデルが存在する．その中でも，入射した光は同じポイントにおいて出射されると仮定したモデルが BRDF(Bidirectional Reflectance Distribution Function)[8] であり，次のように定義される．

$$f_r(x, \Theta, \Psi) = \frac{dL}{dE} = \frac{dL}{L\cos(\theta)d\omega} \tag{2.4}$$

2.2.3 レンダリング方程式

大域照明はすべての光の伝搬を記述する．前節で光エネルギーの基本単位を定義したが，このエネルギーの伝搬を記述するものがレンダリング方程式である．レンダリング方程式はサーフェスで反射される放射輝度を与える．

$$L_o(x \to \Theta) = L_e(x \to \Theta) + L_r(x \to \Theta) \tag{2.5}$$

反射される放射輝度 L_o は，サーフェス自身の自己放射 (emittion)L_e と入射した光の反射量 L_r の和となる．これにより x から Θ 方向に反射する放射輝度 L_o が求められる．一般的なサーフェスでは自己放射 L_e は 0 となるので普通は L_r だけを考えればよい．この L_r は，入射する放射輝度 L_i と BRDFf_r，入射角 ψ の余弦項の積を半球 Ω 上で積分することにより与えられる．

$$L_r(x \to \Theta) = \int_\Omega f_r(x, \Theta, \Psi) L_i(x \leftarrow \Psi) \cos(\theta) d\omega_\Psi \tag{2.6}$$

上記の方程式は半球積分により放射輝度を求めているが，微小立体角 $d\omega$ と微小面素 dA との間の関係式 $d\omega = \frac{\cos(\theta')dA}{r(x,y)^2}$ を利用して面積分による方程式に変形することができる．θ' はポイント y の法線とベクトル $y \to x$ のなす角であり，$r(x, y)$ は x, y 間の距離である．

$$L_r(x \to \Theta) = \int_A f_r(x, \Theta, \Psi) L_i(x \leftarrow y) V(x, y) G(x, y) dA_y \tag{2.7}$$

ここで V(x, y) は可視関数でポイント x, y がお互いに可視であれば $V(x, y) = 1$，そうでなければ $V(x, y) = 0$ となる．G(x, y) は幾何学項で次のようになる．

$$G(x, y) = \frac{\cos(\theta)\cos(\theta')}{r(x, y)^2} \tag{2.8}$$

二種類の式は用途によって使い分けることができる．半球積分による式は全方向に渡って積分をおこなうので，すべての方向からの放射輝度を扱う場合に利用し (間接照明)，面積分による式はある特定領域の面からしか放射輝度が寄与しない場合に利用することができる (直接照明)．

2.3 モンテカルロ光線追跡法

モンテカルロ光線追跡法とは，レンダリング方程式をモンテカルロ積分により解く光線追跡系アルゴリズム全般を指す．このアルゴリズムは，写実的画像合成における最も重要なもので，多くの研究がおこなわれている．代表的なアルゴリズムとして，経路追跡法 [5]，双方向経路追跡法 [7][10]，メトロポリス光輸送 [11] といったものが存在するが，これらの手法に一貫して存在する思想は，如何に少ないサンプル数で解を近似できるか，というものである．これはモンテカルロ積分における被積分関数に対する知識をどれだけ活用

するかという，アルゴリズム的に非常に興味深いものであるが，本論文の対象範囲外なので割愛する．

本節では，モンテカルロ積分について簡単な説明をおこない，レンダリング方程式をどのように解くのかを述べる．そして，本研究で利用する経路追跡法についての直感的な解説をおこなう．

2.3.1 モンテカルロ積分

モンテカルロ積分は一様乱数列 ξ_i を用い関数 $f(x)$ の無作為なサンプリングをおこなうことにより積分値を推定する．

$$I = \int_a^b f(x)dx \approx (b-a)\frac{1}{N}\sum_{i=1}^{N} f(\xi_i) \tag{2.9}$$

このモンテカルロ積分の推定値は一般に $N = \infty$ で真の解に収束するが関数の膨大なサンプル値を必要とし収束が遅いため効率的ではない．またサンプル数 N が少ないことによる分散は, そのまま合成画像にノイズとしてあらわれてしまう．効率的解法のためにはモンテカルロ積分を正しく適用することが必須である．

モンテカルロ積分の収束が遅い一因は，被積分関数の性質を考慮しない一様サンプリングをおこなっているところにある．予め関数の重要な部分がわかっていればその部分にサンプルを集中させることにより推定値の分散を抑え解の収束を早めることが可能となる．この方法を重点的サンプリングといい，関数 $f(x)$ の性質を表す確率密度関数 $p(x)$ を与えることによりおこなう．

$$I = \int_a^b \frac{f(x)}{p(x)}p(x)dx = E\left[\frac{f(x)}{p(x)}\right] \approx \frac{1}{N}\sum_{i=1}^{N}\frac{f(\psi_i)}{p(\psi_i)} \tag{2.10}$$

ψ_i は $p(x)$ の密度により生成した乱数列であり, これにより $f(x)$ の重要な部分にサンプルを集中させることができるようになる．特に被積分関数を正規化した確率密度関数を与えることができれば，ゼロ分散の推定値を得ることができる．

2.3.2 レンダリング方程式の解法

レンダリング方程式の積分範囲である半球 Ω の確率密度 $p(\Psi)$ を与えることにより，次のモンテカルロ推定値を得る．

$$\begin{aligned} L_r(x \to \Theta) &= \int_\Omega f_r(x,\Theta,\Psi)L_i(x \leftarrow \Psi)\cos(\theta)d\omega_\Psi \\ &\approx \frac{1}{N}\sum_{i=1}^{N}\frac{f_r(x,\Theta,\Psi_i)L_i(x \leftarrow \Psi_i)\cos(\theta_i)}{p(\Psi_i)} \end{aligned} \tag{2.11}$$

積分項は入射する放射輝度 $L_i(x \leftarrow \Psi)$ と余弦重み付き BRDF $f_r(x, \Theta, \Psi)\cos(\theta)$ の二つの関数の内積であり，重点的サンプリングのための確率密度関数 $p(\Psi)$ はこれら二つの関数の性質により決定されるべきである．この推定値を効率的に求めることが昨今における大域照明研究の課題の一つであり，それは確率密度関数 $p(\Psi)$ をいかに決定するかという問題でもあるといえる．

積分項におけるもう一つの重要な点は，放射輝度 L_i が未知関数なところにある．大域照明系では，すべての物体は光を反射することにより放射輝度を放つ．しかしその放射輝度を求めるためには，その物体のポイントにおいて再びレンダリング方程式を解かなければならない．つまりレンダリング方程式は再帰的になるのである．入射する放射輝度 L_i は光源をサンプリングすることにより定まるが，光源がサンプリングされなければ再帰が深くなっていく．

以上の点から，レンダリング方程式のモンテカルロ的解法は，放射輝度を確率密度関数に沿って伝播させるモデルであるといえる．このモデルは光輸送と呼ばれ，現在では光線追跡法によるポイントサンプリングに基礎を置く経路追跡法などのアルゴリズムが適用される．

2.3.3 経路追跡法

本研究で利用する光輸送アルゴリズムは経路追跡法である．経路追跡法とは，レンダリング方程式の完全な解法となる最も単純なアルゴリズムである．前節までは，数式を使い理論的な説明をおこなってきたが，この節では直感的な解説をおこなう．

古典的光線追跡法を基礎とする光線追跡系のアルゴリズムは，視点から光線を追跡していく．これは光源から光線を追跡していっても，その光線のほとんどが目に入ってこないため計算が無駄になってしまうからである．視点から放出された光線はオブジェクトのサーフェスに接触し反射するが，古典的光線追跡法と経路追跡法においては，この反射処理が異なる．古典的光線追跡法では，その時点で光源に対して光線を向かわせる．これは光源からサーフェスに直接届く光量が最も支配的になると仮定したうえで，複数回反射して目に届く光を考慮していない．これに対し，経路追跡法ではサーフェスの接触地点からランダムな方向に光線を出射させる．これにより，すべての方向から入射する光が考慮され，より写実的な画像を生成することが可能になるのである．

以下に経路追跡法のプロセス (図 2.1) と擬似コード (**Procedure 1** Path Tracing) を示す．

1. 視点から一次光線を生成する．
2. 光線が交差する一番近いサーフェスを判定する．
3. サーフェスから BRDF に基づくランダムな方向に光線を生成する．
4. 光線が光源に到達するか，反射で減衰し消滅するまで 2，3 を繰り返す．

5. 光源に到達した時点で視点への放射輝度を計算する．

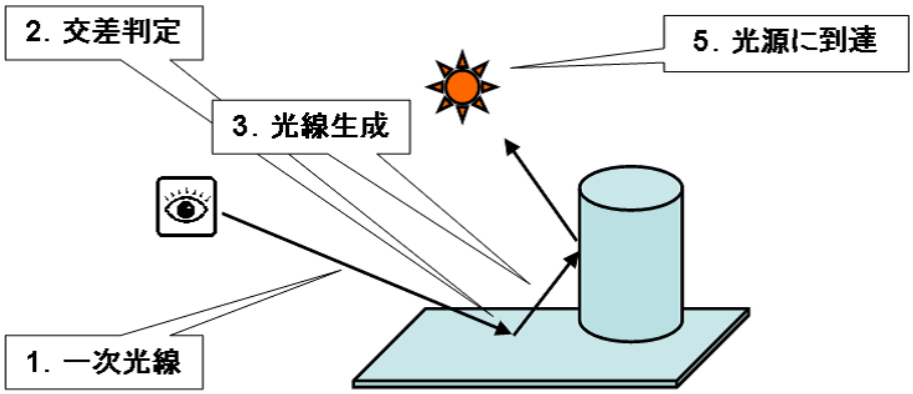

図 2.1: 経路追跡法

経路追跡法では 1〜5 の一連のプロセスが 1 サンプルとなる．可能性のある光線経路の一つを確率的に決定することにより，その視点方向に入射する放射輝度を推測するのである．可能性のある光線経路を多くサンプルすることにより正しい解へと収束していく．これが経路追跡法の計算コストが高い理由で，単純に光線経路をランダムに決定しているために正しい解へと収束させるためには，大量のサンプリングが必要になる．

サンプル数の取り方により画像がどのように変化していくのかを図 2.2 から図 2.4 に示す． サンプル数が少ないと画像が暗くなっているのが，これは光源へ到達する光線が少ないためである．このため経路追跡法で合成した画像は，サンプル数が少ないとノイズが顕著にあらわれる．

2.4 まとめ

本章では，レンダリングにおける基礎理論についてを説明した．

写実的画像合成においては，大域照明という物理シミュレーションによる光エネルギーの伝播を扱う．このエネルギーの伝播は，物体表面の反射率分布関数である BRDF の定義により反射され，この一連の伝播の流れはレンダリング方程式として定式化される．レンダリング方程式は積分方程式であり，これを解くにはモンテカルロ積分が利用される．レンダリング方程式のモンテカルロ的解法を与えるアルゴリズムはモンテカルロ光線追跡法と呼ばれ，現在の写実的画像合成における最重要となるアルゴリズムである．

本研究ではモンテカルロ光線追跡法の一種である経路追跡法を利用する．経路追跡法は実装が単純であり，並列化とも相性が良い．しかしながら，サンプリング処理に関して多くの問題を抱えている．第四章で，経路追跡法を用いた際のサンプリング問題を解決するアルゴリズムの提案をおこなう．

Procedure 1 Path Tracing

```
1: RenderImageUsingPathTrace
2: for all pixel do
3:    for all sample do
4:       generate ray
5:       color = color + Trace(ray)
6:    end for
7:    pixelcolor = color / numberofsamples
8: end for
9:
10: Trace(ray)
11: intersection test
12: get intersection data point and normal
13: color = Shade(point, normal)
14: return color
15:
16: Shade(point, normal)
17: if point == light source then
18:    color = lightradiance
19: else
20:    color = color + Trace(randomray)
21: end if
22: return color
```

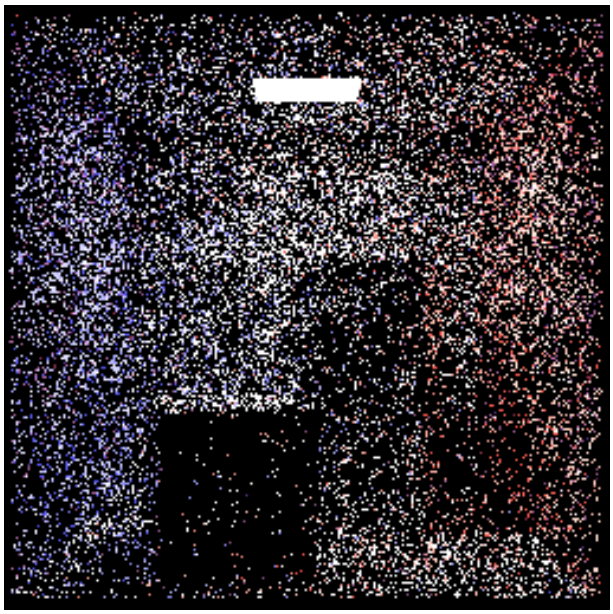

図 2.2: cornell-box:16samples/pixel

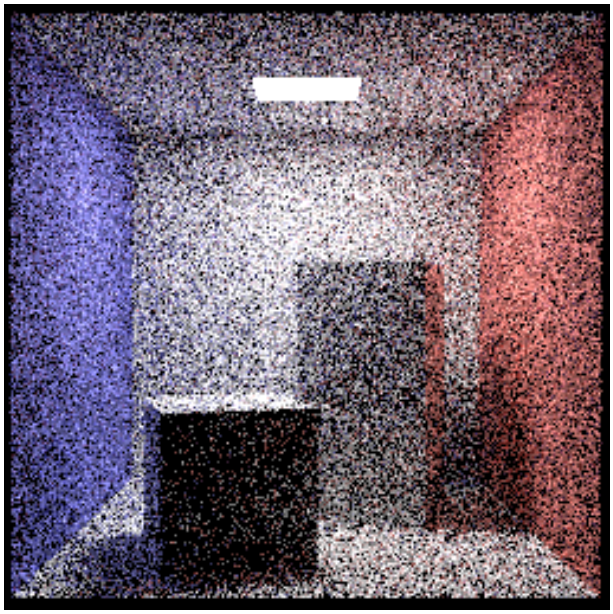

図 2.3: cornell-box:128samples/pixel

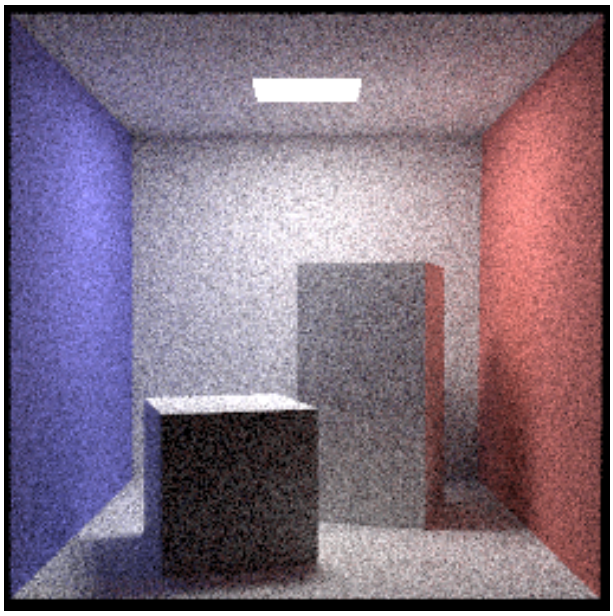

図 2.4: cornell-box:1024samples/pixel

第3章　並列レンダリング

3.1　はじめに

レンダリング，特に光線追跡法を用いたものは多大な計算コストがかかる．そのためレンダリングの高速化のための研究が数多くおこなわれており，その中の一つとして並列化が挙げられる．一口に並列化といっても，その方法は様々であり，解決すべき問題によって並列化のアプローチも変わってくる．

本章では，代表的な並列化手法の説明をおこない，その並列化の有効範囲を明らかにする．そしてモンテカルロ光線追跡法を利用した場合のレンダリングシステムの課題を挙げ，どのような問題点を解決するべきかを考察する．

3.2　並列化手法

従来の並列レンダリングにおける研究は，イメージ空間分割による並列化とオブジェクト空間分割による並列化の二種類に大別することができる．これは並列計算モデルとしての要求駆動モデルとデータ並列モデルに対応しており，すべての基礎となる並列化モデルである．要求駆動並列化 (図 3.1) は，それぞれのプロセッサに対して，要求に応じてタスクが割り当てられるモデルである．データ並列化は，タスクが対象とするデータをそれぞれのプロセッサに分割し，プロセッサ上では同じタスクを実行するモデルである．光線追跡法においても，この二種類の並列化手法はよく適している．

次節からにおいて，並列レンダリングにおける代表的な手法であるイメージ空間分割とオブジェクト空間分割について考察し，それぞれがどのような問題点を解決しようとし，またどのような問題を抱えているかを述べる．そして上記二つの手法を組み合わせたハイブリッド手法についても簡単に説明する．

3.2.1　イメージ空間分割手法

イメージ空間分割による手法では，画像のピクセルやスキャンラインといった単位で各プロセッサにタスクを割り当てることにより並列処理を実現する [14][12]．各プロセッサで割り当てられたイメージの部分空間のピクセル値を計算し，最終的に一つのイメージを得る．光線追跡法は各ピクセルにおいて独立した処理なので非常にスケーラブルな並列処

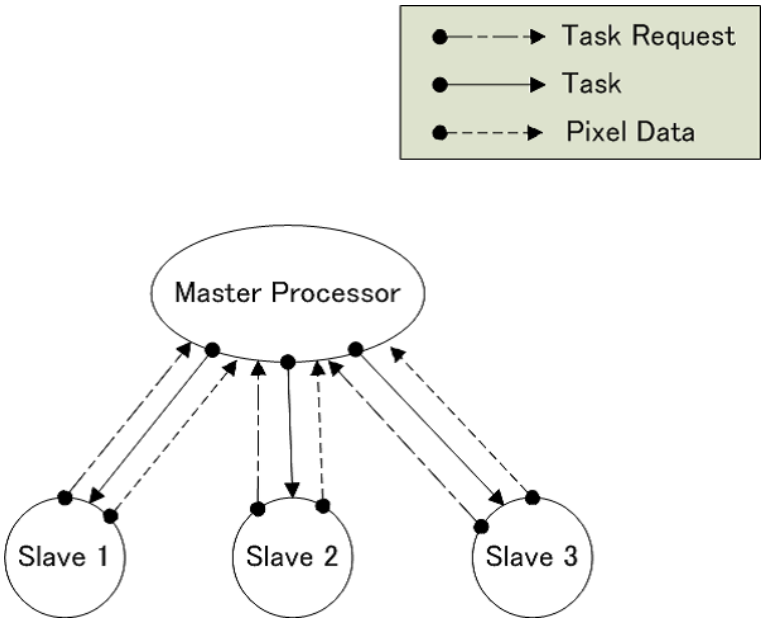

図 3.1: 要求駆動並列化

理をおこなうことができるが，モンテカルロ光線追跡法においてはピクセル間での計算コストに大きな違いがでてくるためプロセッサ間のロードバランスを維持することが難しい．

イメージ空間分割による研究では，どのように画像を分割しプロセッサに割り当てるかという部分に主眼がおかれる．静的，動的ロードバランシングをおこなうためには，各イメージ空間においての計算コストを推測しなければならない．部分イメージ空間においては，シーンの複雑性が大きく異なってくる可能性がある．

静的ロードバランシング

静的ロードバランシングは，あらかじめ決まったサイズの部分イメージの処理をタスクとし，それぞれのプロセッサに割り当てる方法である．図 3.2 のように各プロセッサ 1〜4 がそれぞれの部分イメージを処理する．この方法では，それぞれのタスクの複雑性の違いにより，ロードバランスが悪くなるのは明らかであるが，通信コストを最小限に抑えることができる．

動的ロードバランシング

動的ロードバランシングは，タスクの数がプロセッサ数よりも多くなった場合に，タスクを動的にプロセッサに割り当てる方法である．図 3.3 のように部分イメージサイズを均等に分割してタスクとするものから，部分イメージを計算コスト予測により適応的サイズに分割しタスクとするものまで，様々な方法が存在する．ロードバランスは良くなるが，通信コストが静的な場合よりも増加する．

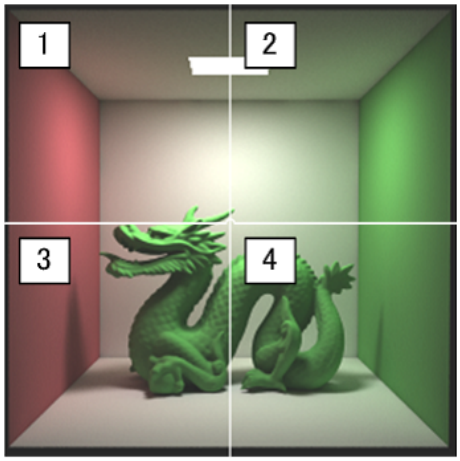

図 3.2: 静的ロードバランス

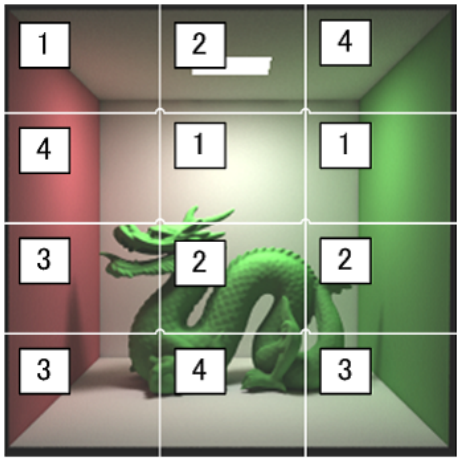

図 3.3: 動的ロードバランス

イメージ空間分割の従来研究では部分イメージへの分割の仕方に焦点が当てられている．イメージ空間における最小単位であるピクセルをタスクとすれば，良好なロードバランスが得られるが，通信コストが大きくなってしまう．

3.2.2 オブジェクト空間分割手法

オブジェクト空間分割による方法は，シーンデータを分割しそれぞれのプロセッサに割り当てることにより並列処理をおこなう (図 3.4) [15]．光線をすべてのプロセッサに与え，それぞれの分割されたシーンのポリゴンとの交差判定をおこなう．この方法は，主に単一メモリ内にシーンのポリゴンデータが収まりきらない場合の対処法として用いられるが，どのようにシーンを分割するかという複雑性の高い問題を抱えている．

オブジェクト空間分割による研究では，データの分散配置に関して，以下の三つのコスト指標に気をつける必要がある．

- どのプロセッサにおいても均等な計算コストになるようにする．
- メモリの要求サイズがどのプロセッサにおいても均等になるようにする．
- 通信コストが小さくなるようにする．

これらの指標を満たすことはとても大変である．通常，シーンデータは，交差判定の計算コスト削減のために BSP-木 (Binary Space Partition tree)[1] や八分木 (octree)[2] といった空間分割手法により階層化される．そのため，単純に同数のポリゴンをプロセッサに配分するといった方法では均等な分散配置は難しい．

また，光線交差判定は同じ光線を複製してそれぞれのプロセッサで処理するためすべてのプロセッサにおいて負荷分散が均等でなければ，処理が終わるまでの待ち時間が発生してしまう．この問題の解決法として Kilauea [6] が参考になる．

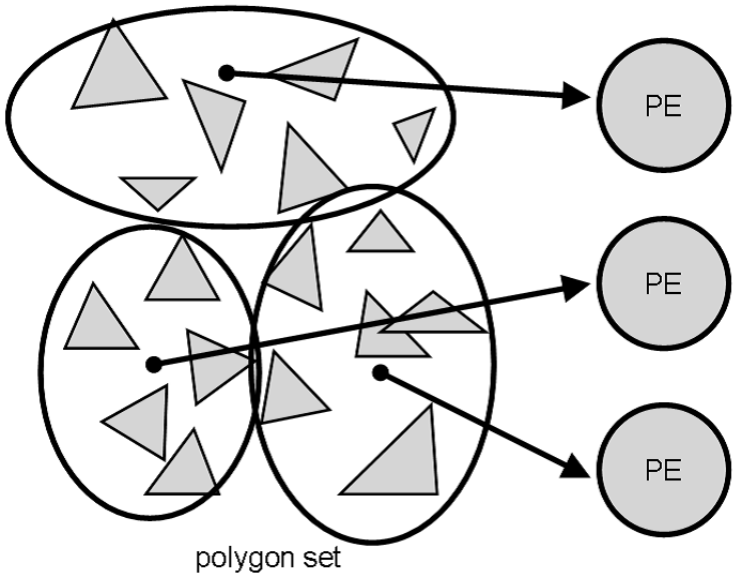

図 3.4: オブジェクト空間分割

3.2.3 ハイブリッド手法

上記の二つの手法を組み合わせたハイブリッドな手法の提案もおこなわれている．どのような考え方でハイブリッド化をおこなうのかを簡単に説明しておく．ハイブリッド手法の目的は，イメージ空間分割ではメモリに載り切らないような大きなシーンを，ロードバランスや通信コストといった面において効率的に処理できるようにすることにある．しかしながら，複雑性が最も高く，実装も容易ではない．ここでは Reinhard らによるハイブリッド手法 [9] を紹介しておく．

Reinhard らのハイブリッド手法は，光線の特性を利用してスケジューリングをおこなう．一次光線や影光線といったコヒーレンスの高い光線は要求駆動型で処理し，鏡面反射光線などの光線の独立性の高いものはデータ並列型で処理するといった具合である．詳しくは [9] を参照のこと．

3.3 従来型並列化の目的と問題点

従来型並列化における主な目的は以下の二点に焦点を当てることができる．

- 速度問題の解決．レンダリングは非常に計算コストの高い処理である．特にモンテカルロ光線追跡法においては，シーンのポリゴン数，光線のサンプル数といった要因で，計算コストが膨大に跳ね上がってしまう．そのため，並列化により処理速度を向上させることは非常に有用である．

- メモリ問題の解決．レンダリングにおいて，単一メモリ内にシーンのすべてのポリゴンが乗り切らない場合がある．このような場合に，分散メモリクラスタなどにすべてのポリゴンを分散して載せる方法がとられる．

イメージ空間分割では速度問題の解決，オブジェクト空間分割では速度，メモリ，双方の問題を解決することができる．イメージ空間分割手法においては，プロセッサそれぞれにシーンデータが複製されており，それぞれのプロセッサにおいて異なる光線を処理させることにより並列処理をおこなう．これは単一メモリ内にシーンのポリゴンすべてが載ることが前提となっている．これに対し，オブジェクト空間分割では光線を複製し各プロセッサに分散されているポリゴンとの交差判定処理を並列におこなう．負荷分散や待ち時間の問題も考えると，シーンデータが単一メモリ上にすべて展開できるなら，積極的にイメージ空間分割手法を利用した方が良い．

ここでモンテカルロ光線追跡方を利用した場合のイメージ空間分割並列化において，サンプリングという問題が顕著になる．イメージ空間での最小タスク単位はピクセルとなり，従来ではピクセル以上のタスク単位での並列処理が主流であった．モンテカルロ光線追跡法におけるサンプリング処理は各ピクセルにおいてそれぞれ独立に何度もおこなわれる処理であり，1024samples/pixel の場合だと，1 ピクセル辺り 1024 回ものサンプリング処理がおこなわれることになる．これだけのサンプル数を取ると，タスク単位が小さいとはいえ，光線交差判定処理にかかる時間は大きくなってしまい，各々のタスク間の仕事量にも大きな差が出てくることになる．それが結果としてロードバランスが悪くなる等の問題を引き起こしてしまう．

3.4 まとめ

この章では，並列レンダリング手法を概観し，今までどのような研究がおこなわれてきたかを述べた．現在の並列計算機システムでは単一プロセッサにおいても十分なメモリ量が提供されていることが多く，余程大きなシーンデータでない限りオブジェクト空間での分割をする必要性を感じることはない．

本研究では，イメージ空間分割を基礎としたサンプル空間による分割手法を提案する．サンプル空間とそれを利用した並列アルゴリズムについては第 4 章で解説する．

第4章　分散平衡並列化

4.1　はじめに

これまでのイメージ空間分割による並列化においては，画像の品質の評価が難しかった．これは，イメージ空間がピクセル，またはピクセルの集合を単位として扱っていたためである．この方法では，古典的な光線追跡法においては良好な結果を与えるが，モンテカルロ光線追跡法においては，サンプリング手法の影響によりピクセル間のロードバランスを維持することが難しい．これは，ピクセル一つをサンプルするのに多量の光線が必要となるが，シーン空間のポリゴン数のばらつきにより各々のピクセルでの計算量に大きな差が出てしまうからである．これは光線の数を増やせば増やすほど顕著になっていく．また各々のピクセル値が収束するためにはどれだけのサンプル数が必要かということも事前にわからないために，無駄なサンプリングがおこなわれてしまうということも挙げられる．

本章では，この問題を解決するために，画像の品質を視野にいれた，適応的サンプリングをおこなう分散平衡並列化手法を提案する．これは，各ピクセルの分散の評価を細かい層にわけておこなうことにより，ピクセルにおける無駄なサンプリングを排除しながら，そのタイムスライスにおける画像品質を一定に保つ手法である．これは，イメージ空間分割よりもさらに細かい並列粒度であるサンプル空間を対象に並列化をおこなうことにより実現する．この手法の実装・評価をおこない，提案手法の妥当性を示す．

4.2　イメージ空間の等価性

従来のレンダリングにおいては，画像のピクセルそれぞれについて同じ数だけのサンプリングをおこなうようになっていた．この方法では，ピクセル間の収束速度の違いにより画像全体の品質の均衡が保てない．特にイメージ空間分割並列化のように，イメージの部分空間をそのままスレイブに割り当ててしまうと，ピクセル間の収束の違いを判定することが難しくなる．逆にピクセル間の収束速度の違いを判定できれば，無駄なサンプリング処理を省くことが可能になり，画像の品質を保ったまま計算量を削減することができる．

この節では，画像の各ピクセルにおいて値がどのように変化していくのかを観察する．そして，この結果を踏まえ，次節で分散平衡アルゴリズムという，値の変動の大きいピクセルを集中的にサンプリングし，画像全体での収束状態を一定に保つようにする手法を提案する．これによりオーバーサンプリングによる無駄な計算を省き，アンダーサンプリングなピクセルに計算が集中するような効率のよいシステムを目指す．

4.2.1 ピクセルの収束

経路追跡法において，ピクセルがどのように収束していくのかを考える上で，照明についての簡単な知識が必要になる．通常，目に入ってくる光は，光源から直接届くもの，光源からサーフェスの反射を一回経て到達するもの，反射を複数回経て到達するものといったように分けることができる．これは，直接照明，間接照明といったように分けて考えることができ，特に直接照明のような目に強い刺激を与える照明が画像にとって重要な部分となる．

4.2.2 収束評価実験

照明の知識を踏まえた上で，cornell-box シーンの 4 つのポイントにおいてピクセル値が収束していく様子を捉えた (図 4.1)．観察したポイントはそれぞれ以下のようになっており，(125, 90) のポイントは直接照明と間接照明の部分，(30, 120) のポイントは直接照明と間接照明を含む青色の部分，(76, 30) のポイントは間接照明のみの比較的明るい部分，(75, 180) は間接照明のみの暗い部分である．これらのポイントにおける収束具合を RGB 成分で表示した図がそれぞれ，図 4.2，図 4.3，図 4.4，図 4.5 になる．それぞれの図の X 軸がサンプル数，Y 軸がピクセル輝度 (radiance) になる．そしてこの 4 つのピクセルをグレイスケール値で重ね合わせたのが図 4.6 である．

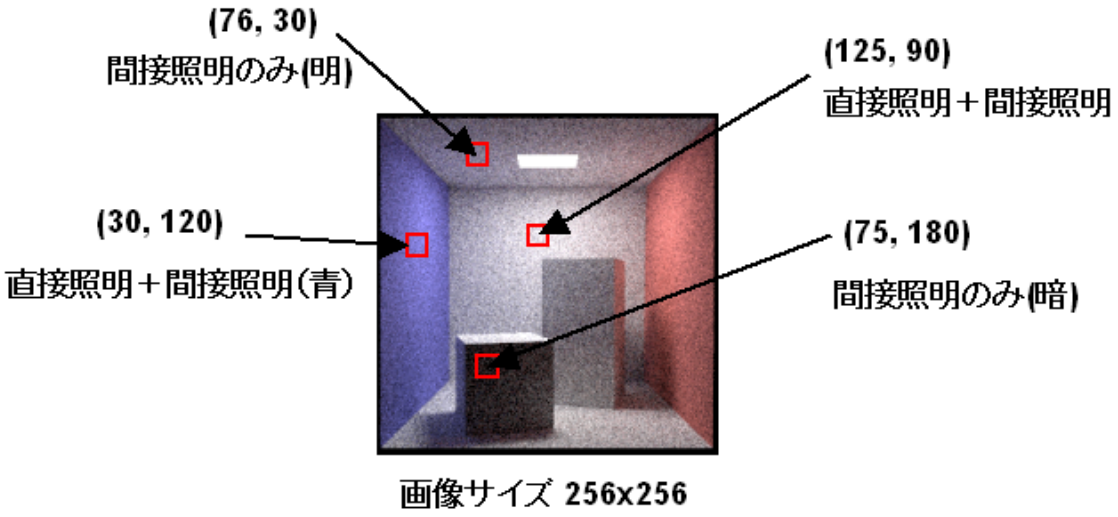

図 4.1: ピクセルの分散チェック

これらの図から，照明が強く影響する明るい部分においてはピクセル値の変化が激しく (図 4.2)，逆に暗い部分ではほとんど変化していないことがわかる (図 4.5)．特に図 4.6 を見てわかるとおり (125，90) のピクセルと (75，180) のピクセルにおいては明らかに収

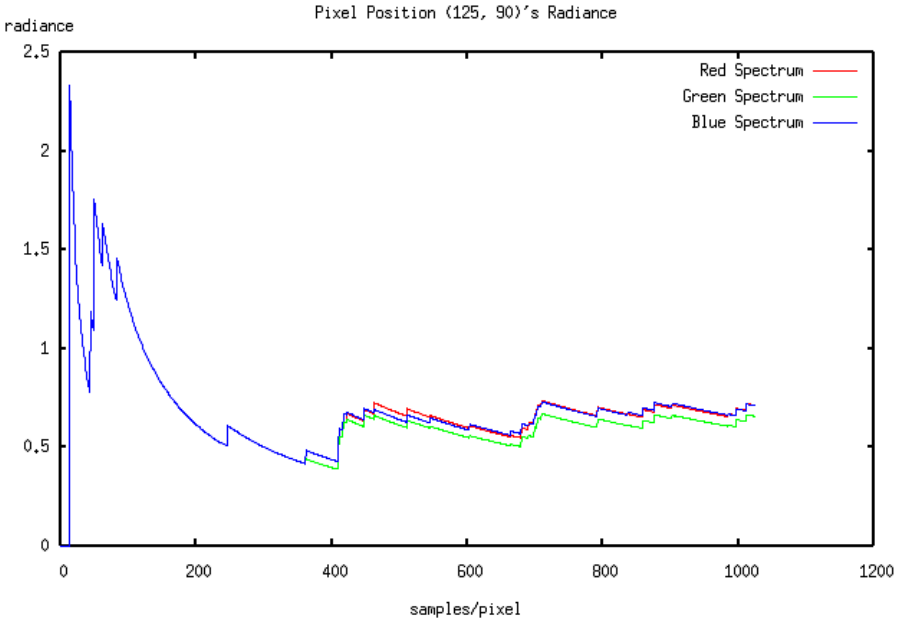

図 4.2: ピクセル (125, 90) における収束

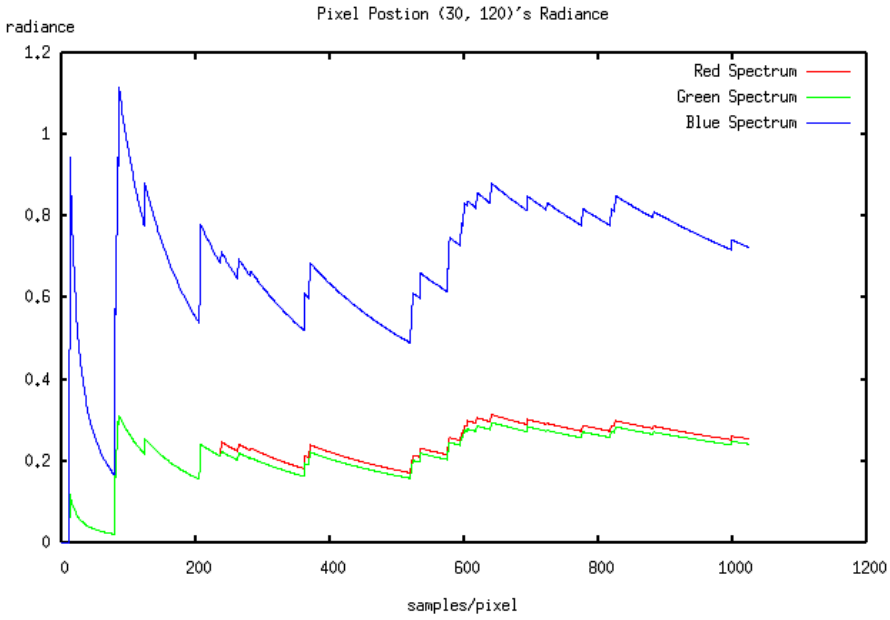

図 4.3: ピクセル (30, 120) における収束

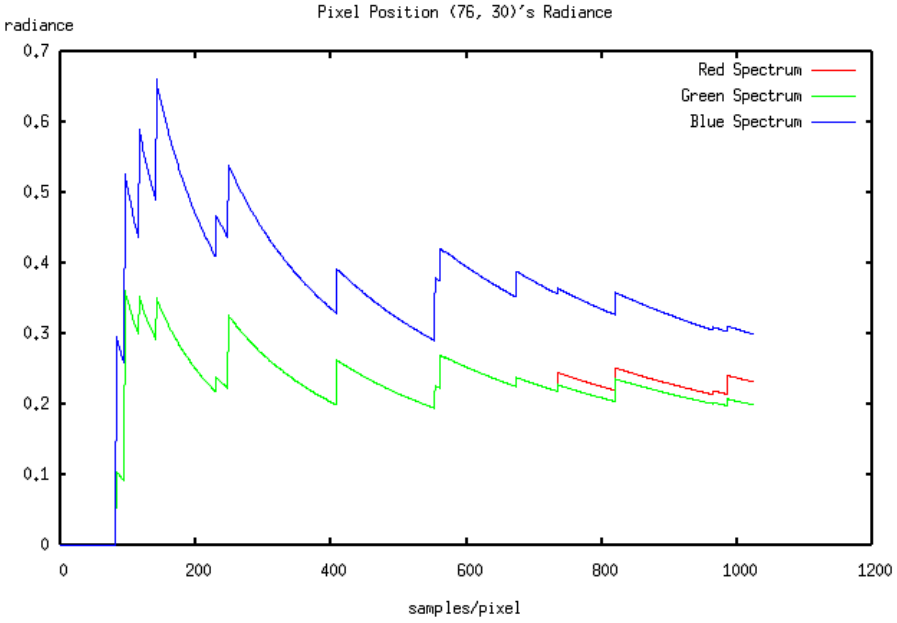

図 4.4: ピクセル (76, 30) における収束

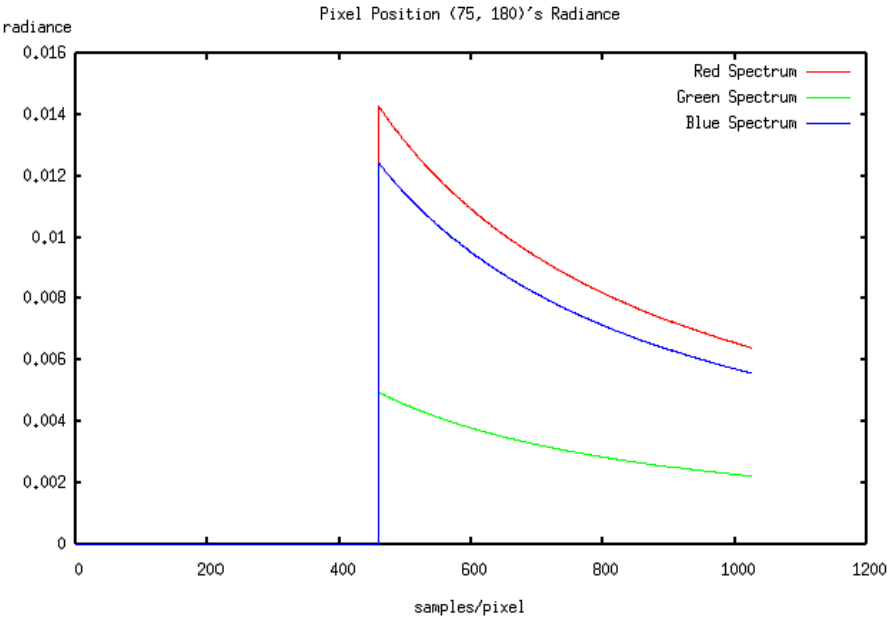

図 4.5: ピクセル (75, 180) における収束

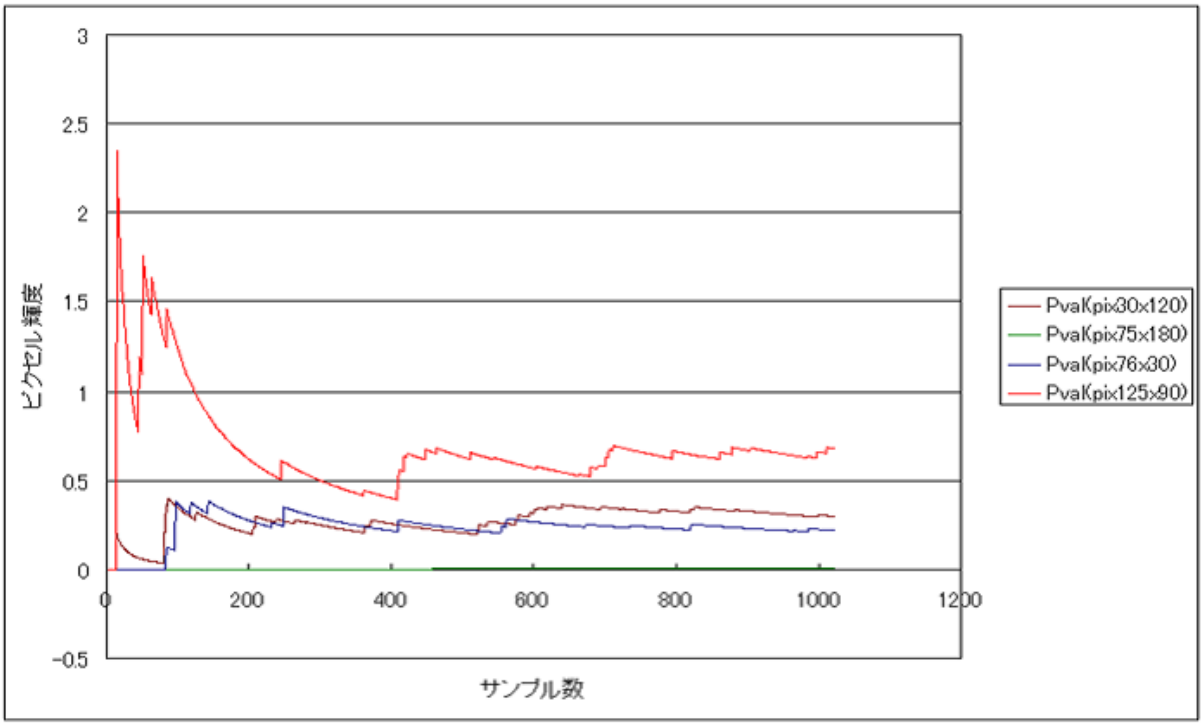

図 4.6: 4 点における収束差

束速度が違いすぎる．この二つのピクセルに同数のサンプリング処理をおこなうことは，計算資源の無駄になることは明らかである．明るい部分ほど分散が大きく解の収束が遅いため，暗い部分よりもサンプル数を多くとってやる必要がある．

4.3 分散平衡アルゴリズム

これまでにピクセルの収束速度に起因するサンプリング問題について見てきた．前節の実験により，画像の明るい部分ほど分散が大きく，暗い部分ほど分散が小さいために解の収束速度に差がでてくる．これでは，これらのピクセルに同数のサンプリングをおこなうことは明らかに計算資源の無駄であることがわかる．この節では，この問題を解決するために，ピクセルの分散値から解の収束性を判定し，収束速度が遅いピクセルに計算資源を集中させる，分散平衡並列化アルゴリズムを提案する．

分散平衡の基本となる考え方は無駄なサンプリング処理を省くという部分にある．経路追跡法による画像合成では，ピクセルそれぞれにおいてレンダリング方程式を解くことになるが，その解はモンテカルロ積分で求めるために多くのサンプル数を取る必要がある．しかし，それぞれのピクセル位置においてどの程度のサンプル数を取れば解が収束するのかということがわからない．そのため，解が十分に収束しているのにサンプリングをおこ

なってしまうことが起こる．その無駄に使われてしまう計算資源を解の収束が遅い部分に割り当てようというのがこのアルゴリズムである．

4.3.1 ピクセル分散

まず始めに，ピクセルの収束度合いをあらわす値として，ピクセル分散というものを以下のように定義する．

$$Pvar(x,y) = \sum_{i=1}^{N-1}(P_i - P_{ave})^2 \tag{4.1}$$

N はサンプル数であり，P_i は i 番目のサンプリングをおこなったときのピクセル値である．P_{ave} は N 回のサンプリングをおこなったピクセル値で，$P_N = P_{ave}$ である．

通常，ピクセル値はRGBの三成分であらわされる．ピクセル分散の値をそれぞれの成分ごとにおこなうこともできるが，ここではピクセルの輝度を用いて計算をおこなう．ピクセルの輝度とは，いわゆるグレイスケール値のことでRGB空間から以下の変換式を用いることで求められる．

$$P(x,y) = R * 0.3 + G * 0.59 + B * 0.11 \tag{4.2}$$

三成分での分散計算をおこなわない理由は，計算量の問題もあるが，我々の視覚に関しての問題によるものが大きい．個々のピクセル分散による影響は画像にノイズとしてあらわれ，直接視覚に訴えてくる．上記の変換式を見てわかるとおり，RGBそれぞれの係数の値が大幅に違っている．G成分の値が大きく，B成分は小さくなっているが，これは緑色のほうが目に強い刺激を与え，青色は他の二色ほど視覚に強い影響を与えないことを意味する．この理由により，分散計算にはピクセルの輝度値を採用した．

4.3.2 サンプル空間の粒度

ピクセル分散を計算する上で問題となるのが N のサンプル数である．任意のピクセルのサンプリング数が N の場合，ピクセル分散は N サンプルを取るまでに，このピクセル値がどれだけ変動したかを示す指標となる．しかしながら我々が知りたいのは，サンプリングがすべて終了したときの分散の指標ではなく，サンプリング途中での分散の指標である．そこで $1 \sim N$ までのサンプル空間を等間隔の層 (layer) として分割し，この間隔をサンプル粒度とする (図 4.7)．この図では，X-Y平面をイメージ空間とし，Z軸をサンプル空間としている．それぞれのピクセルについてサンプル数という次元が存在しているということである．このサンプル粒度を単位としてピクセル分散を計算することにより，ピクセル分散に基づいたサンプリングをおこなう．

ここで，サンプル粒度をどの程度にすればいいのかということが問題になる．特にサンプル粒度が小さすぎると，モンテカルロ法のランダムな性質からピクセル分散値が信頼性

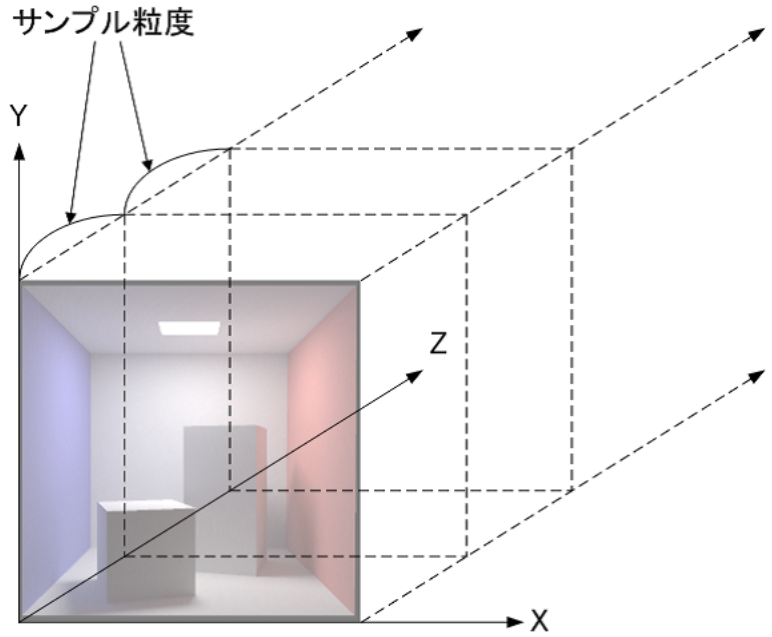

図 4.7: サンプル粒度

のないものになってしまう．また，並列計算機上での実装を視野に入れると，このサンプル粒度がそのままタスクサイズとなるので，これも考慮しなければいけない．

そこで，サンプル空間の粒度と実行速度の関係を調べるための実験をおこなった．並列化手法は後に述べる Master-Slave 法で，使用したシーンは cornell-box [16]，実験結果は表 4.1 にまとめた．

表 4.1: 異なるサンプル粒度による速度実験結果 [sec]

サンプル粒度	1	8	16	32	64
スレーブ数 1	3258.940	1871.504	1743.615	1706.665	1680.325
スレーブ数 8	881.433	248.838	225.968	214.938	210.048
スレーブ数 16	881.159	141.649	116.691	108.869	105.879
スレーブ数 32	881.092	115.280	70.421	57.022	53.661

サンプル粒度を 1 とすると，式 4.1 が必ず 0 となりピクセル分散の計算ができないので論外であるが，興味深いことにサンプル粒度 1 という細かいタスクサイズで Master-Slave 法を利用すると，極端にスケーラビリティが落ちることがわかる．スケーラビリティを保持するためには，タスクサイズ，すなわちサンプル粒度を大きくすればするほど良いのだが，それでは，この研究の目的である，ピクセル分散の平衡を保ちながら無駄なサンプリング計算を省くということが難しくなってしまう．またサンプル粒度が小さければ，その

層で計算されるピクセル分散値が信頼性の低いものになってしまい，計算するピクセルが局所解に陥ってしまう可能性がある．本研究では，4.2.2 節と本節の実験結果を踏まえ，サンプル粒度を 32 と設定した．

4.3.3 アルゴリズム

提案するアルゴリズムの目的は，ピクセル分散の値を判定することにより分散の変動が大きいピクセルのサンプリングを集中的におこなうことにある．図 4.8 はあるピクセルにおける解の収束の様子を示す．このサンプル軸をサンプル粒度で分割，それぞれを層とし，層単位でサンプリング計算をおこなう．

各層のピクセル分散は式 4.1 により計算する．図 4.8 で説明すると，各層における一番最後のサンプル値がその層における基準ピクセル値となる．層において基準ピクセル値と，それぞれのサンプル点における差の 2 乗を合計したものが，その層におけるピクセル分散となる．

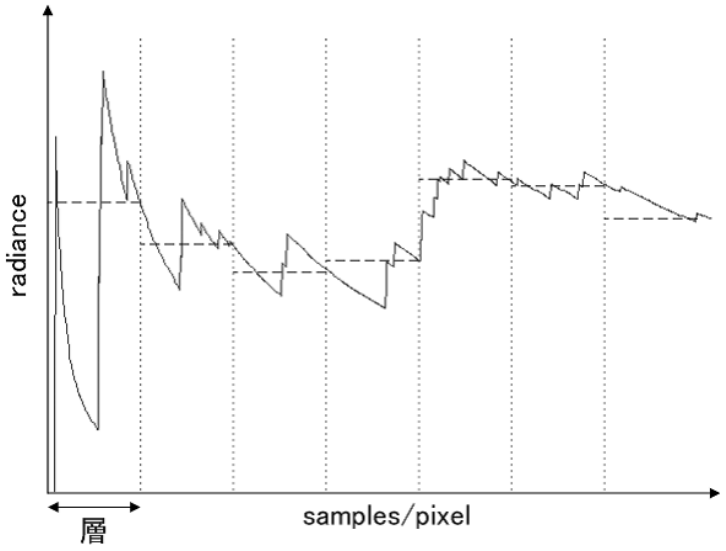

図 4.8: ピクセル分散の計算

任意の層においてピクセルの計算をスキップするかどうかは，画像における全ピクセルの分散値の平均を取って閾値とし判定する．

$$Pvar_{ave} = \sum_{i=1}^{allpixel} Pvar_i \tag{4.3}$$

任意のピクセルが閾値より大きい分散値であれば，より多くのサンプリングが必要と判断し計算をおこなう．閾値以下であれば，その層でのサンプリングをおこなわない．

ここでピクセル分散値の信頼性について考えなければならない．ピクセル分散は図 4.8 を見てわかるとおり，初期のサンプル層では値が大きく，後半になるにつれて小さくなっていく．これはサンプルを多く取れば取るほど解が安定していくことを示すが，初期の層においてこの分散値で収束性を判断すると，局所解に陥る危険性がある．そのため，閾値に初期のサンプル層計算においては分散平衡アルゴリズムは利用せず，図 4.9 のようなフラグ列を用いて，アルゴリズムを適用する層とそうでない層に分割した．このフラグ列は

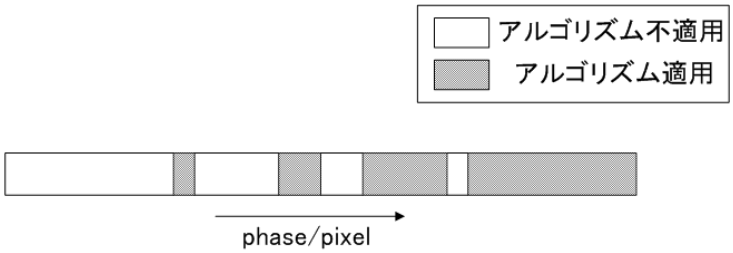

図 4.9: アルゴリズム適用層

ヒューリスティックに作成しているが，基本的な考え方は総サンプル数が少ないうちはピクセル間の分散値の比較をおこなわず全ピクセルで均等にサンプルを取り，サンプル数が十分多くなってきてから集中的にアルゴリズムを適用するというものである．

これらを踏まえて分散平衡アルゴリズムの擬似コードを **Procedure 2** Variance Balanced Algorithm に示す．3 行目のコードによりアルゴリズムを適用する層か判断し，4 行目でピクセルの分散値に基づいてピクセル計算をおこなうか決定する．13 行目が層における各ピクセルの分散値の計算処理で，すべてのピクセル走査が終わったあとに 17 行目で次の層で用いる分散閾値を計算する．

4.4 システムの実装

本研究では，提案アルゴリズムを並列計算機上に実装し，評価をおこなう．使用する並列計算機は分散メモリ型のもので，並列化手法は Master-Slave 法である．

4.4.1 並列計算機システム

本研究で使用する並列計算機は分散メモリ型並列計算機の Cray-XT3 である (図 4.10)．XT3 のシステム構成を表 4.2 示す．

Procedure 2 Variance Balanced Algorithm

```
1: for all layer do
2:   for all pixel do
3:     if flag[layer] { アルゴリズムを適用する層であるか } then
4:       if variance[pixel] < Pvar_ave then
5:         continue
6:       end if
7:     end if
8:     for all sample do
9:       color[pixel][sample] = TraceRay(pixel)
10:    end for
11:    P_ave = color[pixel][SampleMax]
12:    for all sample do
13:      variance[pixel]+ = (color[pixel][sample] − P_ave)^2
14:    end for
15:  end for
16:  for all pixel do
17:    Pvar_ave = variance[pixel]/allpixel
18:  end for
19: end for
```

図 4.10: Cray-XT3 の外観

表 4.2: Cray-XT3 のスペック

OS	Catamount LWK
プロセッサ	AMD Opteron150 2.4GHz × 180
メモリ	8GB × 180

4.4.2 Master-Slave 法

本研究では，並列化手法として Master-Slave 法を採用する．Master-Slave 法は非常に高いスケーラビリティを実現することができる手法である．プロセッサの割り当ては，マスターとなるプロセッサが一つ，残りはスレーブとなり，すべてのスレーブが絶え間なくマスターにタスクを要求する．マスターは要求のあったスレーブに対しタスクを割り当て，これをすべてのタスクが終了するまで繰り返す．この特性により，タスク間の仕事量のばらつきが大きい場合にとても有効な手法となる．あとは次のような特徴がある．

- タスクの仕事量がばらついている場合の他，スレーブとなるプロセッサの処理能力がばらついているような環境においても有効である．
- タスクに依存性があり，タスク間の情報のやりとりを必要とし通信をおこなうような並列化では Master-Slave 法を適用するのは難しい．
- 通常，マスターはタスクの割り当てのみをおこなうが，マスターの計算資源を利用してタスクを処理してもよい．

時間軸においてマスターとスレーブがどのように通信をおこなうかを図 4.11 に示す．マスターは要求のあったスレーブに対してタスクを投げる．図ではマスターはスレーブに 1，2，3 の順にタスクを渡し，スレーブは 3，1，2 の順でタスクを処理し終える．このときにマスターではバリア同期待ちなどの処理が必要なく動的にその都度タスクを投げることができるので，非常にロードバランスが良くなる．

4.4.3 MPI による実装

本システムでは，並列計算ライブラリとして MPI(Message Passing Interface) [18] を使用した．MPI は分散メモリ並列計算機を対象とした粗粒度並列向けの仕様で，計算ノード間の通信をおこなうことができる．

次に提案アルゴリズムを用いた並列レンダリングの様子を示す．

1. 各スレーブがシーンデータの読み込みをおこなう．ここではポリゴンなどの全データがメモリ上にすべて載ることを想定している．マスターは画像などのメモリを確保し，スレーブからのタスク要求と計算結果の送信を待つ．

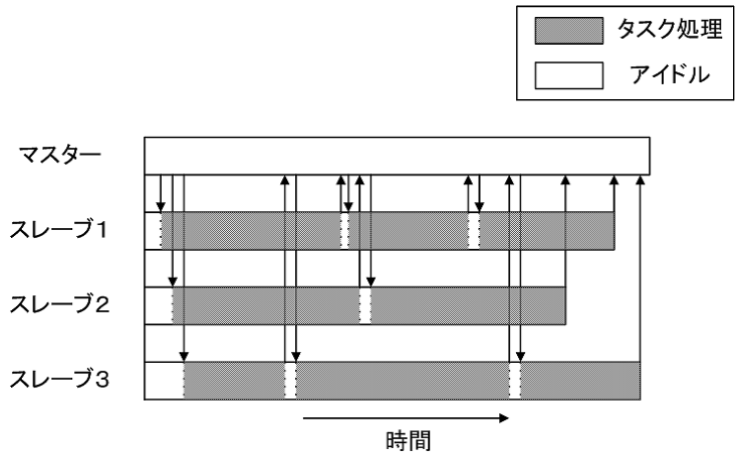

図 4.11: Master-Slave 法によるタスクフロー

2. スレーブがマスターにタスクを要求し，マスターはスレイブにタスクを割り当てる．マスターは各ピクセルの層単位を 1 タスクとする．

3. タスクを処理し終えたスレーブはマスターに計算結果を返す．計算結果の内容はピクセル値，ピクセル分散等である．

4. マスターは全ピクセルの走査を終えた時点で，その層での画像分散平均とピクセル分散の値により，適応的サンプリングをおこなうためのピクセル選別をおこなう．

5. 指定サンプル数を取り終えるまで，2〜4 を繰り返す．指定サンプル数に達した時点でスレーブに終了メッセージを送り画像を書き出す．

メッセージデータ

通信に使用するメッセージデータは構造体で表 4.3 のように定義した．データサイズの削減のために，マスターによるデータ送信時と受信時ではフィールドの使用用途を変更している．

Master-Slave 法擬似コード

次に MPI を用いた Master-Slave 法による実装の擬似コードを，マスターは **Procedure 3** Master に，スレーブは **Procedure 4** Slave にそれぞれ示す．

マスターは 2 行目の MPI::Recv で常にメッセージ待機状態になっている．スレーブからのメッセージは，タスクの要求をする DemandTask と計算結果の返却をする ReturnResult の 2 種類である．DemandTask を受け取ったマスターはスレーブにタスクを割り当てる．ReturnResult を受け取ったときはピクセルと分散データをバッファにストアする．また 13 行目で層における最後のピクセルの計算が終了したかどうかをチェックし，分散閾値の計算と次の層で計算するべきピクセルの選択をおこなう．

表 4.3: メッセージデータ

	マスター送信時	マスター受信時
int	処理するピクセルの位置	処理したピクセルの位置
int	処理するサンプル数	処理したサンプル数
float	今まで処理したサンプル数	ピクセル分散
float	現ピクセル位置の R 成分合計	処理したサンプル数分の R 成分合計
float	現ピクセル位置の G 成分合計	処理したサンプル数分の G 成分合計
float	現ピクセル位置の B 成分合計	処理したサンプル数分の B 成分合計

Procedure 3 Master

```
1: for all task do
2:   MPI::Recv(Message)
3:   if Message :: Tag == DemandTask then
4:     if task exist then
5:       select task
6:       MPI::Send(AllocateTask, task)
7:     else
8:       MPI::Send(KillSlave)
9:     end if
10:  else if Message :: Tag == ReturnResult then
11:    add pixel color to frame buffer
12:    add variance to variance buffer
13:    if pixel processing is over in this layer then
14:      calcurate image variance
15:      select high variance pixel calcurated in next layer
16:    end if
17:  end if
18: end for
```

Procedure 4 Slave

```
1: while True do
2:   MPI::Send(DemandTask)
3:   MPI::Recv(Message, task)
4:   if Message :: Tag == AllocateTask then
5:     for all sample such that 1 ≤ sample ≤ max do
6:       color[sample] = TraceRay(x, y)
7:     end for
8:     data :: color = Σ color
9:     data :: variance = CalcurateVariance(color)
10:    MPI::Send(ReturnResult, data)
11:  else if Message :: Tag == KillSlave then
12:    break
13:  end if
14: end while
```

スレーブは 2 行目の MPI::Send でマスターにタスクを要求する．マスターが常にメッセージ待機状態であるのに対し，スレーブはアクティブにタスク要求をおこなう．スレーブの処理は単純で，6 行目で計算するべきピクセルにおいて指定数のサンプルを取る．これだけで，あとはピクセル値と分散値の計算をおこないマスターに計算結果を返却してやるだけである．

Procedure 2 Variance Balanced Algorithm におけるコードを，マスターとスレーブでこのように処理を分担してやることで並列化を実現した．

4.5 まとめ

本章では，並列レンダリングの一手法として，分散平衡並列化アルゴリズムを提案した．

まずピクセルの収束の様子を見るために，cornell-box シーンの四点を取って解の収束具合を観察した (図 4.1)．この実験により，ピクセル間の収束速度の違いを確認し，より多くの計算を必要とするピクセルとそうでないピクセルの差を明らかにした．提案アルゴリズムは，このピクセル間の収束の違いをピクセル分散という値で定義し，その値によって適応的にサンプリング処理をおこなう手法である．これにより，分散の大きいピクセルに計算資源を集中させ，同サンプル数を処理した場合での画像全体の品質を向上させるのが狙いである．

また，並列化に際するタスクサイズの問題としてサンプル粒度に関する実験をおこない，適切なサンプル粒度の決定をおこなった．サンプル粒度は，大きすぎると層が少なくなり無駄なサンプリング処理を生むことになる．小さすぎると並列化に際するスケーラビリティが著しく悪くなり，また分散値も信頼性のないものになってしまう．そのため適切

なサンプル粒度を選択することはとても大切である．

並列化においてはMaster-Slave法を利用し，マスターがスケジューリング，スレーブがサンプリング処理をおこなうようにした．Master-Slave法を用いることにより，ロードバランスの良い並列処理が可能となる．

実際にこのアルゴリズムを適用した場合の評価は第5章でおこなう．

第5章　評価結果

5.1　はじめに

本章では，第4章で提案したアルゴリズムの評価をおこなう．分散平衡並列化アルゴリズムは，収束の遅いピクセルに計算資源を集中させサンプリング問題を解決するためのものである．従来では各ピクセルに均等にとっていたサンプル数を動的に調整し，ピクセル分散の大きい部分のサンプリングを多くおこなう．これにより画像全体のノイズの軽減を図る．そのためアルゴリズムの質としての評価項目としては次の二点でおこなう．

- サンプリング処理の動的調節具合
- 画像の品質

また並列化におけるアルゴリズムの性能も見ておく必要がある．並列化においての性能は次の三点で評価する．

- ロードバランス
- スケーラビリティ
- アルゴリズムの速度

それぞれにおいて，提案アルゴリズムの適用・不適用の場合の評価をおこなう．ここで提案アルゴリズム不適用というのは，サンプル粒度によるタスク分割をおこない，適応的サンプリング処理はおこなわない場合のことを指す．

5.2　並列化における性能

並列化における性能評価の指標としてロードバランス，スケーラビリティといったものがある．

ロードバランスはすべてのプロセッサが均等な量のタスクをこなしているかというもので，各プロセッサの性能の違いや通信待ちなどによって大きく変わってくる．ロードバランスが良い場合は，どのプロセッサ資源も効率的に利用できているということになる．

スケーラビリティはプロセッサ数の増加に対する速度向上比をあらわす．今日の並列計算機システムにおいては，数百といったプロセッサ数を持ったものも珍しくなくなってき

た．そのためスケーラビリティという指標は，並列計算において最も重要な指標の一つになる．

あとはレンダリングシステムの速度評価に，秒間あたり何本の光線を処理できるのかという単位を用いる (rays/second)．これは光線追跡法を用いたシステム評価において，比較的利用されることの多い単位である．

評価に使用するデータとして cornell-box(図 5.1)[16] と happy-buddha(図 5.2)[17] を利用した．それぞれのシーンのポリゴン数は表 5.1 の通りである．

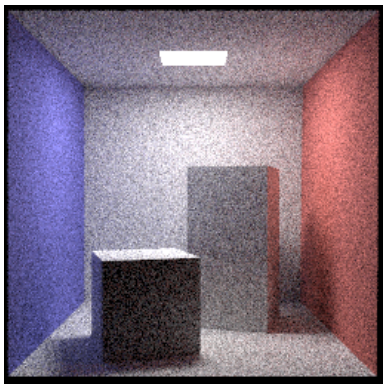

図 5.1: cornell-box

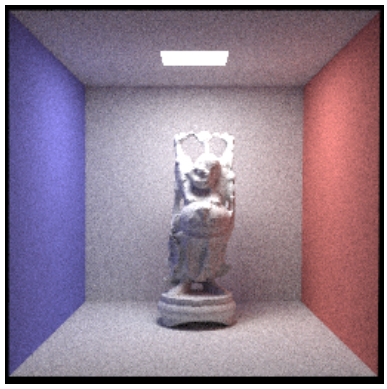

図 5.2: happy-buddha

表 5.1: シーンデータ

	ポリゴン数
cornell-box	32
happy-buddha	15548

5.2.1 ロードバランス評価

ロードバランスの評価は，各スレーブが処理した光線の数を比較することによりおこなう．スレーブに光線を何本処理したのかというカウンタ変数を用意し光線数を測定した．スレーブ数を 8〜32 まで取った場合に，アルゴリズム適用，不適用それぞれで各スレーブが処理した光線数を表 5.2 から表 5.5 に示す．また，処理した光線数の平均と標準偏差を表 5.6 から表 5.9 に示す．

表 5.2 から表 5.5 を見ると，どの場合においても，各スレーブが同数の光線を処理していることがわかる．これはサンプル空間という細かいタスク単位を利用しているためである．また，表 5.6 から表 5.9 を見ると，平均値からに対する標準偏差の値が低く抑えられていることがわかる．これはほぼ平均光線処理数付近の数の光線を各スレーブが処理していることを示す．この結果から，Master-Slave 法を利用したことにより，アルゴリズム適用・不適用に関わらず良好なロードバランスを維持していることがわかる．

表 5.2: cornell-box:光線処理数，アルゴリズム不適用

スレーブ数	8	16	24	32
1[SlaveID]	30383344	15181580	10130210	7597582
2	30422858	15189675	10128497	7612013
3	30394549	15189997	10134832	7543420
4	30367610	15170133	10140247	7551608
5	30416368	15221175	10136035	7626154
6	30380011	15220973	10127143	7616803
7	30399837	15192034	10135589	7611308
8	30400040	15215345	10126053	7610268
9		15194335	10129951	7626851
10		15215923	10133836	7603438
11		15187103	10123950	7617462
12		15222947	10130950	7607733
13		15204174	10139505	7560176
14		15194604	10133189	7620273
15		15194461	10132385	7595010
16		15177534	10138777	7571770
17			10123058	7615039
18			10129375	7577827
19			10131157	7631657
20			10132227	7621868
21			10138028	7569893
22			10130084	7574653
23			10133625	7615665
24			10143201	7562964
25				7615528
26				7587545
27				7601025
28				7615433
29				7618152
30				7567340
31				7619888
32				7588502

表 5.3: cornell-box:光線処理数，アルゴリズム適用

スレーブ数	8	16	24	32
1[SlaveID]	17012373	8490142	5696378	4250535
2	17023622	8524930	5679863	4258457
3	17027799	8521866	5661406	4244732
4	17040215	8530142	5683908	4248493
5	17117950	8534558	5669824	4259737
6	17025764	8524403	5681821	4251880
7	17025992	8463398	5700339	4249391
8	17038952	8475354	5669761	4248717
9		8495268	5684342	4252874
10		8501650	5665822	4259118
11		8468670	5683240	4237267
12		8488618	5677346	4254849
13		8532173	5706403	4247099
14		8539663	5691415	4250879
15		8501712	5702926	4249176
16		8535407	5666674	4254358
17			5678748	4246337
18			5686827	4255667
19			5698570	4251216
20			5699335	4259701
21			5694871	4240024
22			5667010	4255635
23			5676605	4252818
24			5668222	4248332
25				4240536
26				4243100
27				4252833
28				4250948
29				4244614
30				4234985
31				4246364
32				4233977

表 5.4: happy-buddha:光線処理数，アルゴリズム不適用

スレーブ数	8	16	24	32
1[SlaveID]	31192511	15575833	10404525	7793037
2	31194394	15606942	10391037	7797527
3	31207989	15613809	10395535	7787721
4	31206408	15605471	10400140	7818174
5	31199315	15588734	10409087	7785525
6	31199869	15601386	10394188	7800334
7	31165262	15587710	10391239	7792685
8	31173809	15568022	10413318	7809087
9		15595349	10392820	7800216
10		15612600	10399157	7793707
11		15587944	10390208	7799295
12		15602194	10395719	7794362
13		15608004	10404807	7797705
14		15597385	10398887	7802093
15		15601960	10393460	7797650
16		15617265	10393553	7810634
17			10391308	7800337
18			10398412	7782747
19			10401549	7823212
20			10406314	7781988
21			10398465	7779577
22			10400735	7794153
23			10398927	7799795
24			10387641	7811133
25				7796423
26				7795618
27				7800249
28				7790025
29				7803338
30				7803357
31				7799522
32				7807371

表 5.5: happy-buddha:光線処理数，アルゴリズム適用

スレーブ数	8	16	24	32
1[SlaveID]	17495389	8759787	5833720	4395205
2	17492241	8746375	5838228	4374719
3	17501960	8765616	5837688	4395693
4	17507284	8760540	5837783	4396618
5	17519223	8752759	5820980	4398241
6	17514487	8768821	5837556	4398259
7	17502583	8741056	5838369	4393924
8	17500990	8755602	5847724	4384217
9		8766374	5837549	4398012
10		8754625	5852212	4407988
11		8771912	5833181	4386573
12		8749301	5827414	4386858
13		8773628	5841503	4395628
14		8747149	5822362	4398827
15		8743277	5838802	4385439
16		8761211	5846329	4391529
17			5837326	4385108
18			5843816	4385032
19			5839637	4390556
20			5826658	4379426
21			5846644	4389184
22			5843266	4398702
23			5820889	4387552
24			5843656	4391336
25				4377457
26				4376819
27				4387505
28				4380695
29				4390631
30				4389331
31				4389731
32				4387054

表 5.6: cornell-box:光線処理データ，アルゴリズム不適用

スレーブ数	8	16	24	32
平均	30395577	15198249	10132579	7598589
標準偏差	18476	16606	5127	24540

表 5.7: cornell-box:光線処理データ，アルゴリズム適用

スレーブ数	8	16	24	32
平均	17039083	8507997	5682985	4249207
標準偏差	33058	25586	13377	6819

表 5.8: happy-buddha:光線処理データ，アルゴリズム不適用

スレーブ数	8	16	24	32
平均	31192444	15598163	10397959	7798393
標準偏差	15254	13674	6405	9744

表 5.9: happy-buddha:光線処理データ，アルゴリズム適用

スレーブ数	8	16	24	32
平均	17504269	8757377	5837220	4389807
標準偏差	9099	10242	8458	7494

5.2.2 光線処理速度

まずは光線処理速度を測りレンダリングシステムの評価をおこなう．cornell-box と happy-buddha，それぞれでアルゴリズム適用・不適用の場合に，処理した光線の総数と処理にかかった時間，そして処理速度の計測をおこなった．

cornell-box シーンの実験結果を図 5.10，図 5.11 に，happy-buddha シーンの実験結果を図 5.12，図 5.13 に示す．cornell-box，happy-buddha シーン，ともに提案アルゴリズムを適用したほうが速度が落ちている．これは提案アルゴリズムは適応的サンプリングをおこなうためのスケジューリング処理が必要になるため，その部分がオーバーヘッドになっていると考えられる．しかしながらポリゴン数が増加すると，速度の低下率が下がっている．このことはスケーラビリティ評価の節で考察する．

表 5.10: cornell-box:アルゴリズム不適用，光線処理速度実験結果

スレーブ数	処理光線数 [rays]	処理時間 [sec]	処理速度 [rays/sec]
8	243164617	214.376	1134290
16	243171993	109.380	2223185
24	243154848	74.462	3265489
32	243181904	56.942	4270694
64	243133758	35.970	6759348
96	243136760	32.622	7431353
128	243132818	30.881	7873217

表 5.11: cornell-box:アルゴリズム適用，光線処理速度実験結果

スレーブ数	処理光線数 [rays]	処理時間 [sec]	処理速度 [rays/sec]
8	136312667	123.857	1100564
16	136127954	65.141	2089743
24	136391656	46.972	2903679
32	135974649	37.619	3614520
64	136037693	27.920	4872410
96	133669366	26.071	5127128
128	135936790	25.001	5437254

表 5.12: happy-buddha:アルゴリズム不適用，光線処理速度実験結果

スレーブ数	処理光線数 [rays]	処理時間 [sec]	処理速度 [rays/sec]
8	249548597	721.827	345718
16	249551031	362.326	688747
24	249570608	242.319	1029925
32	249539557	182.683	1365970
64	249532793	92.511	2696165
96	249546867	62.783	3974749
128	249530474	48.762	5117314

表 5.13: happy-buddha:アルゴリズム適用，光線処理速度実験結果

スレーブ数	処理光線数 [rays]	処理時間 [sec]	処理速度 [rays/sec]
8	140034157	403.972	346643
16	140118033	204.420	685441
24	140093292	137.672	1017587
32	140473849	104.986	1338024
64	140628009	55.921	2514762
96	140141975	40.344	3473675
128	139929961	33.383	4191653

5.2.3　スケーラビリティ評価

光線処理速度の比較によりスケーラビリティの評価をおこなう．速度評価による処理速度をグラフにあらわし，スケーラビリティを図る．スレーブ数8〜128までによるcornell-boxの結果は図5.3に，happy-buddhaの結果は図5.4に示す．

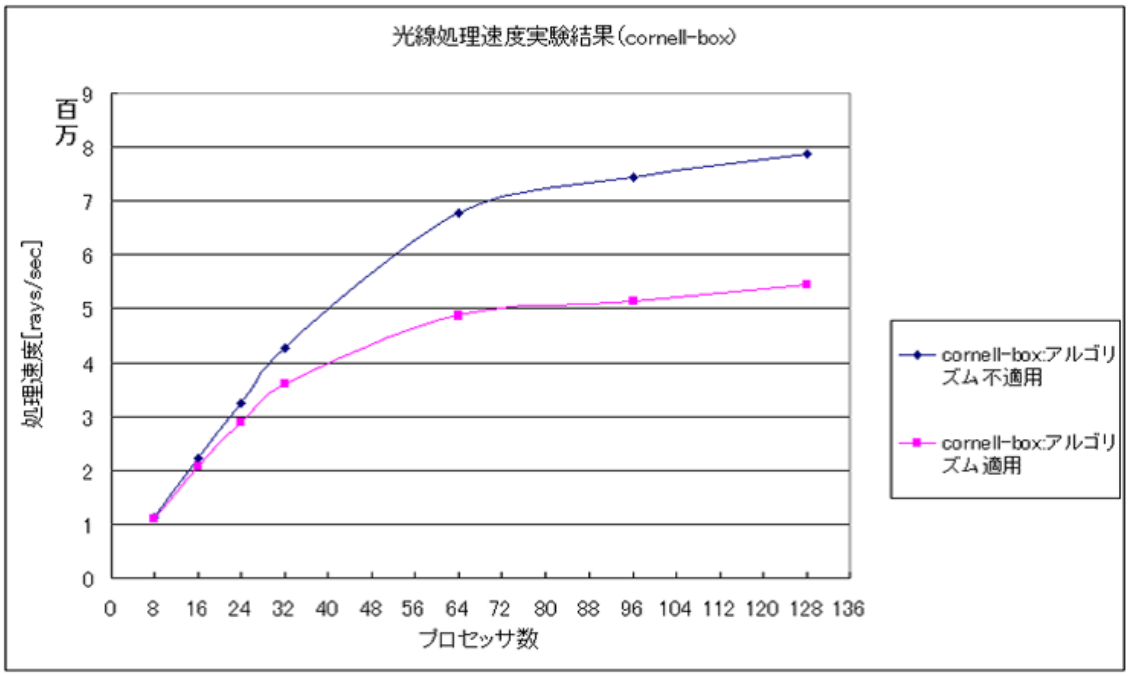

図 5.3: スケーラビリティ:cornell-box

この結果から，スレーブ数の増加によりかなりの処理速度の増加が見られる．しかしながら，提案アルゴリズムを適用したものは，マスターによるスケジュール時間に引きずられ，スレーブ数の増加に伴ってスケーラビリティが落ちているのがわかる．それでもhappy-buddhaのようなポリゴン数の多いシーンになると，タスク当りの処理時間が増加し，提案アルゴリズムの全体に占める時間割合が小さくなるので，スケーラビリティとしては良好な結果を示している．これは，ポリゴン数が多いシーンになればなるほど，提案アルゴリズムの効果が高いということを示している．実用的なシーン(10万〜100万ポリゴン程度)になると，提案アルゴリズムによるオーバーヘッドはほとんどないものと考えられる．

5.3　アルゴリズム性能評価

アルゴリズムの性能評価は，サンプリング問題の解決という面においてピクセルの分散値によりサンプル数を動的に変更するという観点から考察する．ここでは，適応的サンプ

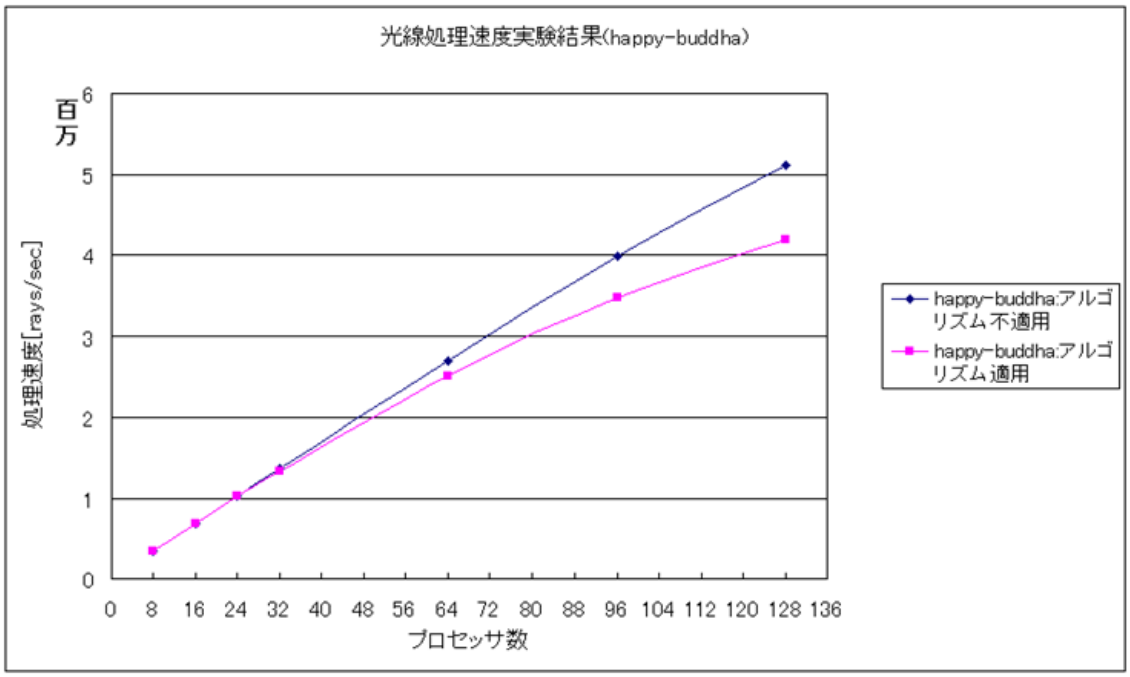

図 5.4: スケーラビリティ:happy-buddha

リングによる画像全体でのサンプリング数の変化，そして合成された画像の品質といった項目で評価をおこなう．

5.3.1 適応的サンプリング処理評価

ここでは，提案した適応的サンプリング手法によりピクセル間でどれだけサンプル数の違いがあらわれるかを検証した (図 5.5)．実験シーンは cornell-box で最大 8192samples/pixel，サンプル粒度 32，層数は 256 となる．

アルゴリズムで説明した通り，すべての層にアルゴリズムを適用するのではなく，解に安定性がない初期層のうちはアルゴリズムを適用せず，後半になってから集中的にアルゴリズムを適用する．使用したアルゴリズム適用フラグ列により，最低サンプリング層数 126，最大で 256 となっている．

図 5.5 をみてわかるとおり，画像の各ピクセルにおいてサンプリング数に違いがでてきている．特に，4 章で検証した分散の大きい部分にサンプリング処理が集中し，分散が小さい部分はほぼ最低サンプリング層数のままである．この結果により適応的サンプリングアルゴリズムは上手く動作しており，解が早く収束するようなピクセル部分の計算資源を収束の遅いピクセル部分に集中させてやることに成功した．このアルゴリズムにより合成した画像の評価は次節でおこなう．

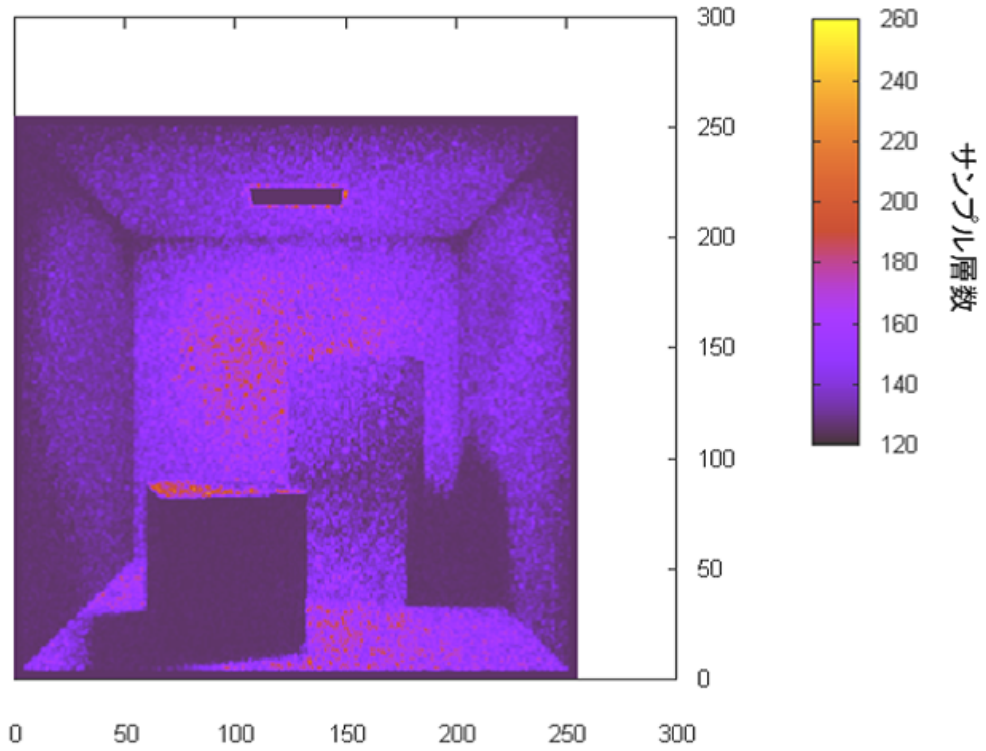

図 5.5: 適応的サンプリング

5.3.2 画像の品質評価

提案アルゴリズムでの画像の品質評価のために，ピクセル辺りの最大サンプル数を 1024 と 8192 として，実施サンプル数を決定した．提案アルゴリズムでは適応的にサンプル数を変更するため，同じサンプル数での調査は単純にはできない．そこで提案アルゴリズムにおいて，画像全体の合計サンプル数を取りそこから逆算して従来のサンプリング数を決定した．1024samples/pixel のサンプリング処理の場合，256 × 256 サイズの画像で総サンプル数は $1024 \times 256 \times 256 = 67108864$ になる．cornell-box シーンにおいて最大サンプル数 1024samples/pixel で提案アルゴリズムを利用した場合は，ピクセルのサンプリング処理棄却により総サンプル数は 38213376 となった．この数値の比は $38213376/67108864 \approx 0.569$ となり，従来手法での実施サンプル数は $1024 \times 0.569 \approx 582$samples/pixel として計算した．

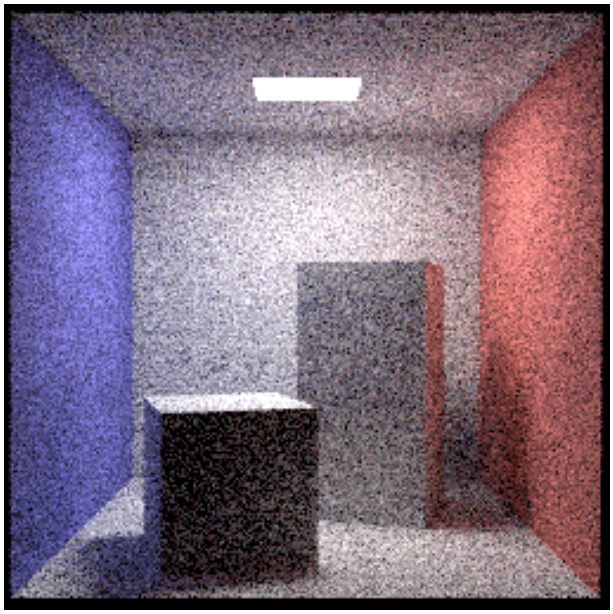

図 5.6: cornell-box:max1024samples/pixel，アルゴリズム適用

最大サンプル数 1024samples/pixel で合成した提案アルゴリズムによる画像が図 5.6，実施サンプル数 582samples/pixel の従来手法による画像が図 5.7 である．この画像を見る限り，知覚できるような大きな変化は見受けられない．同様に最大サンプル数 8192samples/pixel として合成した画像 (図 5.8, 図 5.9) においても知覚できる変化はない．これは画像のノイズ特性によるものと，シーンの各ピクセルにおける分散に大きな違いがないのが原因であると考えられる．分散値と合成画像の関係については，今後も検証が必要である．

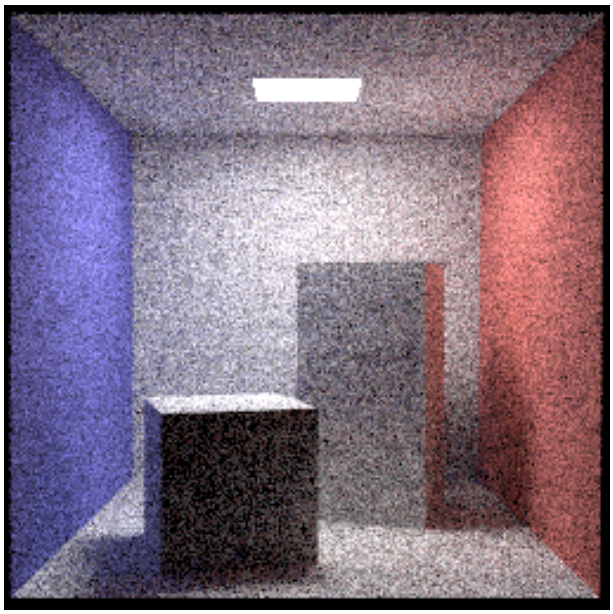

図 5.7: cornell-box:max1024samples/pixel，アルゴリズム不適用

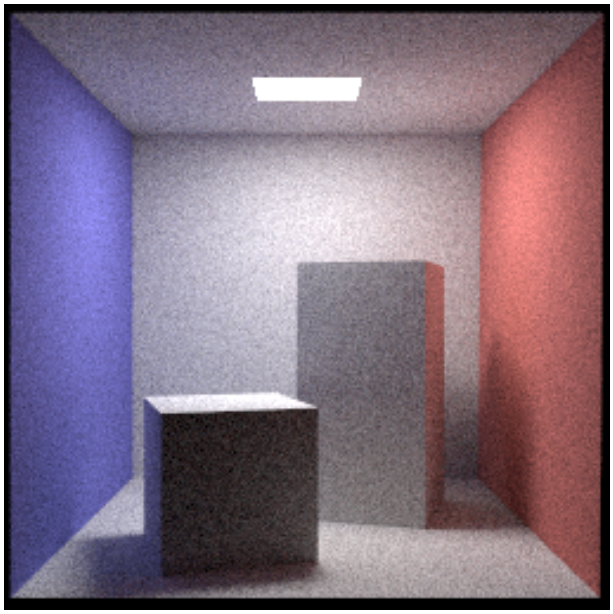

図 5.8: cornell-box:max8192samples/pixel, アルゴリズム適用

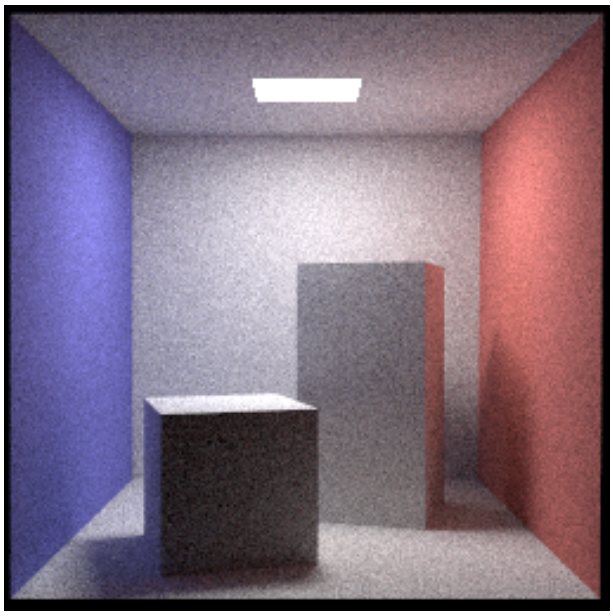

図 5.9: cornell-box:max8192samples/pixel, アルゴリズム不適用

5.4 まとめ

この章では，提案した分散平衡並列化アルゴリズムを，並列化という観点と適応的サンプリングという観点から評価をおこなった．

並列化における評価では，適応的サンプリングアルゴリズムを適用した場合とそうでない場合によって速度面に変化があらわれた．適応的サンプリング処理においては，サンプリング処理を棄却するピクセルを選択するためのスケジュール処理が発生する．そのため，スケジュール部分がオーバーヘッドとなり，ポリゴン数が少ないシーンにおいては従来手法と比較して遅くなっていることがわかる．しかしながら，ポリゴン数の多いシーンになるとタスク当りの処理コストが増加するため，速度においてほとんど差が見られなかった．

適応的サンプリングにおける評価では，分散値の大きいピクセルに適応的に計算資源を集中させることが可能となった．これにより，画像全体の分散値が均衡するようにサンプリングをおこなうことができるが，残念ながら画像自体には知覚できる大きな違いはあらわれなかった．使用するシーンによっては変化が見受けられる可能性もあるので，もう少しの検証が必要である．

第6章　結論

6.1　まとめ

本論文では，モンテカルロ光線追跡法を利用した並列レンダリングの一提案として，画像の分散の大きいピクセルに計算を集中させピクセル間の品質を一定に保つ分散平衡並列化を提案した．並列化においては，適切なサンプル粒度を選択することにより，処理速度，スケーラビリティといった評価項目の質を落とすことなく，ピクセル分散により適応的なサンプリングをおこなうことができる．また，Master-Slave 法による動的なタスク要求処理により，非常に良いロードバランスを保つことが可能である．サンプリング処理においては，分散の変動の大きいピクセルに計算を集中させることにより，画像全体の分散を低く抑えることに成功した．

第 2 章では，レンダリングの基礎知識についての説明をおこなった．現在のレンダリングの基礎となっている放射測定系の単位，それを利用した大域照明理論，すべてのレンダリングアルゴリズムが解くべき目標であるレンダリング方程式を解説した．特にレンダリング方程式はコンピュータグラフィックスにおける画像合成の基本方程式でありとても重要である．そして本研究で利用するモンテカルロ光線追跡法の一種である経路追跡法について説明し，サンプル数の違いにより画像がどのように変化するのかを示した．

第 3 章では，レンダリングの並列化における従来手法について述べた．レンダリングの並列化の枠組みとしては，イメージ空間分割手法とオブジェクト空間分割手法の 2 種類に大別することができる．モンテカルロ光線追跡法はピクセルに対し独立であるので，イメージ空間分割により良好な並列化が可能になるが，ロードバランスの問題などが存在した．そして従来の並列化手法で解決しようとした問題を示し，どのような並列レンダリングが好ましいのかを考察した．

第 4 章では，分散平衡並列化アルゴリズムを提案した．ピクセル分散を定義することにより，ピクセル値の変動具合を定式化し，これにより適応的なサンプリング処理が可能となった．解の収束速度の違いをピクセル分散によって判断し，収束の遅いピクセルに計算を集中的におこなうことができる．並列化に際しては，並列粒度の問題を考察することにより最適な並列粒度を決定し，また，並列化手法に Master-Slave 法を用いることにより，良好なロードバランスとスケーラビリティの実現を可能とした．

第 5 章では，提案アルゴリズムの評価をおこなった．提案アルゴリズムでは，計算ピクセルのスケジュール処理が発生するために，プロセッサが増えた場合には従来手法よりも速度面では劣ってしまう．しかしながらポリゴン数の多いシーンでは，処理コストに占め

るスケジュール処理の割合が小さくなるため，良好なスケーラビリティを示す．実用的なシーンをレンダリングする場合には，処理速度にはほぼ違いがないものになるだろう．また適応的サンプリング面においては，分散の大きいピクセルに計算を集中させることに成功した．これにより，収束の早いピクセルに無駄にあてられていた計算資源を収束の遅いピクセルに割り当てることが可能になった．

6.2 今後の課題

本研究では，分散平衡並列化アルゴリズムを提案し，ピクセル分散が大きい部分に集中してサンプリングをおこない，無駄な計算を省き画像全体で分散値が平衡するようなレンダリングシステムを構築した．しかしながら，解決すべき課題も残っている．

一つ目として画像の品質の問題がある．本研究においては，適応的サンプリングをおこない分散の大きいピクセルに計算を集中させた場合でも，適応的サンプリングをおこなわずに同じだけの総サンプル数を取った場合，画像の品質に知覚できるだけの変化はあらわれなかった．これは各ピクセルが単体としてノイズになるのではなく，ピクセル間の相互作用によりノイズが発生することに起因していると予想する．これを解決するには各ピクセルを独立に扱うのではなく，フィルタリング処理のように複数ピクセルをまとめて扱う必要がある．

二つ目の問題として，モンテカルロ法に関する乱数の問題がある．本研究では無視していたが，並列モンテカルロ法においては乱数の偏りによって解の収束が著しく悪くなる現象が指摘されている [4]．これは並列レンダリングにおいては，ノイズがエイリアスとしてあらわれる現象などが起こる．実用的な並列レンダリングシステムにおいて，これは重要な問題となる．この問題を避けるために，並列実装においては準モンテカルロ法 (Quasi Monte Carlo method) を利用することが好ましいとされている．

本研究における今後の課題としては，様々なシーンにおける検証をおこない，本アルゴリズムが効果的に使用できる範囲を明らかにすることが挙げられる．また，アルゴリズムの改良においては，決定的におこなっていたピクセル選択処理にランダム性を持たせ，局所解に陥らないような柔軟なものにしていく必要があると考える．

謝辞

本研究をおこなうにあたり御指導をいただいた，北陸先端科学技術大学院大学 井口 寧 助教授に深く感謝致します.

適切な御意見，御助言を頂きました北陸先端科学技術大学院大学 松澤 照男 教授，田中 清史 助教授に深く感謝致します.

副テーマにおいて御指導をいただいた，北陸先端科学技術大学院大学 宮田 一乗 教授に深く感謝致します．また，ゼミでお世話になりました宮田研究室の皆様に厚くお礼申し上げます.

そして日頃よりお世話になりました井口研究室の皆様に... ありがとう.

最後に，いつも支えてくださった両親へ感謝を込めて.

本研究に関する発表論文

清水　昭尋，井口　寧，”光線のコヒーレンスを考慮した適応的空間分割による並列レンダリングシステム”，電気関係学会北陸支部大会，Sep.2005

参考文献

[1] H.Fuchs, Z.M.Kedem, and B.F.Naylor. "On visible surface generation by a priori tree structures." Computer Graphics (Proc. SIGGRAPH'80), Vol.14, p.p.124-133(July 1980).

[2] Andrew S. Glassner. "Space subdivision for fast ray tracing." IEEE Computer Graphics and Applications, 4(10): p.p.15-22(October 1984).

[3] Goral, C.Torrance, D.Greenberg, and B.Battaile. "Modelling the interaction of light between diffuse surfaces." Computer Graphics (Proc. SIGGRAPH'84), Vol.18. p.p.218-222.

[4] P.Hellekalek. "Don't trust parallel Monte Carlo!" ACM SIGSIM Simulation Digest 28(1), p.p.82-89(July 1998).

[5] James T. Kajiya. "The rendering equation." Computer Graphics (Proc. SIGGRAPH'86), 20(4): p.p.143-150(August 1986).

[6] Toshi Kato, and Jun Saito. ""Kilauea"-Parallel global illumination renderer." Fourth Eurographics Workshop on Parallel Graphics and Visualization (2002)

[7] Eric P.Lafortune and Yves D.Willems. "Bidirectional path tracing." Compugraphics '93, p.p.95-104(1993).

[8] F.E.Nicodemus, J.C.Richmond, J.J.Hsia, I.W.Ginsberg, and T.Limperis. Geometric considerations and nomenclature for reflectance. Monograph 161, National Bureau of Standards(US), October 1977.

[9] E.Reinhard, A.Chalmers, and F.W.Jansen. "Hybrid scheduling for paralle rendering using coherent ray tasks." 1999 IEEE Parallel Visualization and Graphics Symposium, p.p.21-28.

[10] Eric Veach and Leonidas Guibas. "Bidirectional estimators for light transport." Fifth Eurographics Workshop on Rendering, p.p.147-162(1994).

[11] Eric Veach and Leonidas J.Guibas. "Metropolis light transport." Computer Graphics (Proc. SIGGRAPH'97), p.p.65-76(August 1997).

[12] I.Verd, D.Gimnez, and J.C.Torres. "Ray tracing for natural scenes in parallel processors." High-Performance Computing and Networkking, volume 1067 of Lecture Notes in Computer Science, p.p.297-305. Springer-Verlag, April 1996.

[13] Turner Whitted. "An improved illumination model for shaded display." Communications of the ACM23(6): p.p.343-349(June 1980).

[14] S.E.Hyeon-Ju Yoon and J.W. Cho. "Image parallel ray tracing using static load balancing and data prefetching." Parallel Computing, 23(7): p.p.861-872(July 1997).

[15] J.Zara, A.Holecek, and J.Prikryl. "Parallelisation of the ray-tracing algorithm." Winter School on Computer Graphics adn CAD Systems 94, volume 1, p.p. 113-117. University of West Bohemia, January 1994. WSCG 95.

[16] "http://www.graphics.cornell.edu/online/box/" The Cornell Box - Cornell University Program of Computer Graphics.

[17] "http://www-graphics.stanford.edu/data/3Dscanrep/" The Stanford 3D Scanning Repository.

[18] Message Passing Interface Forum. Document for a standard message passing interface. Technical report, University of Tennessee, Knoxville, 1993.