卒業論文

画面分割によるレイトレーシング法の並列化

指導教官　：山崎 勝弘 教授

論文提出者氏名　：吉谷 崇史
学籍番号　:2170900143-1

1995 年 1 月 27 日提出

立命館大学理工学部情報工学科

摘要

本論分では，レイトレーシング法の特徴，レイトレーシングの並列化，および画面分割を用いた並列計算による高速化について述べる．

レイトレーシング法とは，鏡面や屈折の表示が可能で高品位な画像が得られるコンピュータグラフィックスとして，よく知られる．そのアルゴリズムは非常に簡単で，スクリーン上の各画素を通る視線ごとにこれと最初に交わる物体（前後判定で最も手前のもの）を探索する方法である．各画素に行われる計算時間は物体数に対応して増え，しかもすべての画素に対して同じ処理をするので非常に時間がかかる．

本研究では，４本のシリアルリンクで接続された並列計算機トランスピュータ64台上でリング接続，トーラス接続を用い，Parallel Cにより画面をメッシュ，縦，横に等分割（最大６４分割）した．また画像の記述データより画面上でのおよその負荷（計算時間）を決定し，負荷の度合いにより画面を縦，横に分割するといった計５パターンの画面分割法で並列化を行い高速化を図った．各画素ごとに同じ負荷を持っているとは限らないので、等分割でｎ倍の相対速度は得られないから，できるだけ各プロセッサに与える負荷を均等にするための画面負荷均衡分割のアルゴリズムを考察した．

実験の結果, メッシュ等分割で21倍であるのに対し縦負荷均衡分割で52倍という相対速度が得られかなりの負荷均衡が行われた.

目 次

1　はじめに　1

2　レイトレーシング法　2
- 2.1　レイトレーシング法の特徴 2
  - 2.1.1　陰影 2
  - 2.1.2　反射 4
  - 2.1.3　屈折 4
- 2.2　レイトレーシング法の高速化 6

3　画面分割によるレイトレーシング法の並列化　7
- 3.1　マルチトランスピュータシステム 7
  - 3.1.1　アーキテクチャ 7
  - 3.1.2　相互結合網 7
  - 3.1.3　Parallel C 8
- 3.2　画面等分割 9
- 3.3　画面負荷均衡分割 10
- 3.4　マルチトランスピュータによる並列計算 11

4　画面分割アルゴリズム　13
- 4.1　物体表示位置予測 13
- 4.2　ライン単位の負荷予測 14
- 4.3　画面分割 15

5　実験と評価　16
- 5.1　実験方法 16
- 5.2　実験結果 16
- 5.3　考察 23

6　おわりに　24

謝辞　25

参考文献　26

1　はじめに

コンピュータを使ってディスプレイなどに映像を映し出す技術、いわゆるCG(Compyuter Graphics)は、近年、テレビのCMやタイトルなどで普通に見られる. その中でも3次元CG(3D-CG)の発展は目まぐるしいものがある.3D-CGを映し出すため方法として、「ペイント法」、「スキャンライン法」、「レイトレーシング法」、「ラジオシティ法」[2][3][9][10]など、さまざまな方法がある. そのなかで、本研究ではレイトレーシング法についてとりあげた.

レイトレーシング法は、光の屈折、反射など光学の法則を忠実にシミュレートした手法の1つであり、きわめてリアルな画像を生成できることでCGの分野ではよく用いられている. アルゴリズムは視点からスクリーン上の各画素を通る視線が物体と交差する点を探索し、その交点で反射、屈折など物体の表面の性質に応じて、視線を分けていく. その際に、減退する光量を計算する. これを順次続けて、光量が一定値以下あるいは反射が決められた回数に達したら処理を終了する. アルゴリズムからも明らかなように一つの画素計算に対する視点からの1本の視線が反射、屈折を繰り返すことにより多くの視線となり、これをそれぞれの視線に対して交点計算を行なうために計算量が膨大になる.

その高速化技法として、ソフトウェア的手法および、ハードウェア的手法がある. ソフトウェア的手法としては、バウンディングボリューム分割法、空間分割法など、様々な方法が考え出されている.[1] ハードウェア的手法としては、並列計算機を用いて、画素単位で画面分割をし並列に計算させる方法が、一般的である. また本研究では、英国のINMOS社で開発された、並列計算機であるトランスピュータを用いた. トランスピュータは、4本のシリアルリンクを持ち、マルチプロセッサにより並列処理を行なう.CPU間通信は20MB/Sと高速に行なわれる.

本研究では、あらかじめ画面を分割し、その部分を各プロセッサに振り分ける. 計算の終了したプロセッサより値を返し、最も計算時間のかかったプロセッサでの計算時間が最終的にその画像生成における計算時間となる. その際、画面の分割方法として、一定の範囲を振り分ける等分割(メッシュ、縦、横)による方法がある. しかし、等分割では分割される範囲の負荷がまちまちである. そこで、画像の記述データより画面上での物体の位置を予測し、レイトレーシングのアルゴリズムからその物体の持つ負荷の大きさを物体の性質(鏡面、透過屈折、通常)により決定する. この方法により画面上のおよそのの負荷分布を求め、計算時間を各プロセッサごとに平均的に振り分ける負荷均衡分割(縦、横)の2つの方法を考察した. 実験方法は、4つのテスト画像を5つの分割方法についてトランスピュータ1,4,8,16,32,64台で実行した.

本論文では、まず第2章でレイトレーシング法の特徴、高速化について述べ、第3章では、画面分割によるレイトレーシングの並列化について述べ、第4章では画面負荷均衡分割のアルゴリズムを説明し、第5章で各分割方法について実行し、結果を評価する. 第6章で本論文の結言と今後の課題を示す.

2　レイトレーシング法

2.1　レイトレーシング法の特徴

目に見える物体の多くは，それ自信が発光しているのではなく，光源によって照明されて見える．光源から出た光がいろいろな物体に当たり，反射や屈折を繰り返す．物体に当たるごとに光線は物体の色が付いてくる．光線は目に入って像を結ぶ．よって，光源から出る光線を１本ずつ追いかけ，目に入ってくる光線を見つけ出すと，目に映る像が再現できる．しかし，光源から出た光の中で目に入ってくる光線の数はごくわずかで，ほとんどの光線は物体で吸収されたり，見えない方向に進み目には入ってこない．これを，１本ずつ追いかけたのでは効率が悪い．

そこでこれを逆に考え目に入ってくる光線を逆にたどり，視点からある方向に出た視線が，スクリーン上の点を通りいろいろな物体に当たり，反射・屈折を繰り返す．そして，物体の色の情報を加えることによりスクリーン上の点の色を決定する．この要領でスクリーン上のすべての点に色を置くと目に映るものをスクリーン上に再現することができる．このようにして画像を作り出す方法を「レイトレーシング法」(光線追跡法) と呼ぶ．以下にレイトレーシング法の特徴・欠点を示す．[1][2]

- 特徴
  - 比較的簡単なアルゴリズムで画像を描くことができる．
  - 反射・屈折・陰影などの処理が可能．
- 欠点
  - 対象となる物体は基本的に関数で表現されなければならない．
  - 光線を１本１本追跡するので演算に時間がかかる．

2.1.1　陰影

実際の物体が立体的に見える条件として，光源の種類や位置および物体の材質などで定まる物体表面の色調の変化や影が挙げられる．光源と反対側の物体表面には光の当たらない陰 (shade) の部分が生じ，物体は自分自身の形に応じた影 (shadow) を光源と反対側にある他の物体に投げかける．

ここでは，陰をつけ色調の変化を表現することを陰影付け (shading)，影をつけることを (shadowing) と呼ぶことにする．陰影付けはシェーディングや，濃淡付けとも呼ばれる．

物体の陰影は，物体の性質，および光源の種類，個数，位置などにより定まる．コンピュータグラフィックスでよく使われている光に対する物体の性質は，その表面から目が受ける光の種類とその強さにより特徴付けられる．光の種類は一般に，

- 反射光
  - 拡散反射光 (diffusely reflected light)．
  - 鏡面反射光 (specularly reflected light)．
- 透過・屈折光 (transmited/refracted light)．

に分けて考えられる．

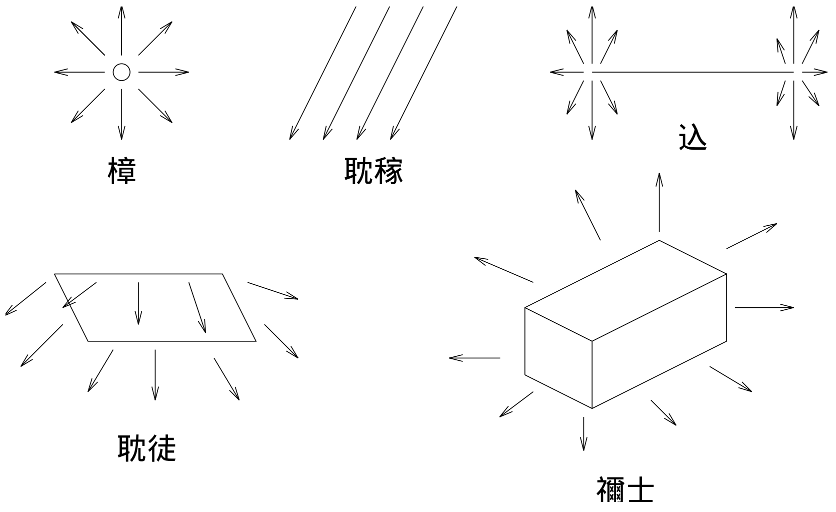


図 1: 光の種類

光源は，その形状や光の色とその強さなどにより特徴付けられる．形状としては，点光源，平行光源，線光源などが一般的である．目的によっては，面や立体などの複雑な形状をもつ光源も用いられる．

光源の配置　光源としては点光源を考える．点光源は真下が最も明るく，周辺ほど暗い．それは，光の強さが光源からの距離の 2 乗に反比例して減少するからである．このことを考慮して，光源の方向余弦も物体の交点ごとに計算する必要がある．

Lambert の余弦則　拡散反射物体はあらゆる方向に光を反射する．したがって，どの角度からその表面を眺めても同じ明るさで見える．このような表面の輝度は，Lambert の余弦則 (Lambert's cosine low) でモデル化できる．すなわち，光線の方向余弦を $\boldsymbol{L}$，表面の輝度を $\boldsymbol{I}_d$, 平行光源の光の強さを $\boldsymbol{I}_p$，入射角を$\boldsymbol{\theta}(\boldsymbol{0} \sim \boldsymbol{\pi}/\boldsymbol{2})$，面の単位法線ベクトルを $\boldsymbol{N}$, 拡散反射係数を $\boldsymbol{k}_d(\boldsymbol{0} \sim \boldsymbol{1})$ とすれば，

$$\boldsymbol{I}_d = \boldsymbol{k}_d\boldsymbol{I}_p \cos\boldsymbol{\theta} = \boldsymbol{k}_d\boldsymbol{I}_p(\boldsymbol{L}\cdot\boldsymbol{H})$$

の関係があるとするのが Lambert の余弦則である．

環境光の考慮　Lambert の余弦則では直接光による反射成分だけだが，実際は壁や床などの周囲の物体による間接光（相互反射光）が存在するため，真っ暗な陰は生じない．ここで，簡便に相互反射の効果を出すため直接光による反射光に一定の環境光による拡散反射光を加えることにする．環境光の強さを $\boldsymbol{I}_a$とすると，環境光による成分を含めた拡散反射光による表面輝度 $\boldsymbol{I}_d + \boldsymbol{I}_{ad}$は，

$$\boldsymbol{I}_d + \boldsymbol{I}_{ad} = \boldsymbol{k}_d\boldsymbol{I}_p(\boldsymbol{L}\cdot\boldsymbol{N}) + \boldsymbol{k}_d\boldsymbol{I}_a$$

となる．[1][2]

2.1.2　反射

物体の表面がなめらかだと周りの物を映すことがある．鏡のような物では周囲を明瞭に反射する．反射により，この映り込みを表現することができる．

鏡面反射をする物体でも，周りから入る光を100 %反射するものもあれば，ある程度は吸収や乱反射をして残りを鏡面反射する物もある．鏡はほぼ100 %の光を反射する．このような物体はそれ自身の色は見えない．周りの物体の映り込みで，そこに鏡があることが分かる．また，なめらかな表面を持つ陶器などでは，それ自身の色でそこに物体があるのが分かる．これは入射してきた光が物体表面で吸収や散乱をされるためである．しかし，よく見ると鏡面反射をしていて周囲の物を映し込んでいることが分かる．

視点よりスクリーン上の点への視線ベクトルを設定し，このベクトルが当たった物体の反射率によって処理をいくつかに分ける．[1][2]

- 物体表面が鏡面のときはベクトルを反射する方向に変える．この方向で物体とぶつかるかどうかさらに調べる．
- ベクトルが当たった点の物体が鏡面反射をせず，すべて散乱してしまうときは，この点の色をピクセルの色にする．

2.1.3　屈折

ガラスでできた物や水に光線が当たると，その一部は反射するが，ほとんどは中に入り込む．そのとき光線の方向が少し変わる．この屈折について説明する．

光が進行してくる物体をa，屈折して入射する物体をb，法線からの角度をそれぞれ$\boldsymbol{\theta_1}, \boldsymbol{\theta_2}$，物質abの屈折率をそれぞれ$\boldsymbol{n_1}, \boldsymbol{n2}$とするとき，

$$\boldsymbol{n_1} \sin \boldsymbol{\theta_1} = \boldsymbol{n_2} \sin \boldsymbol{\theta_2} \tag{1}$$

の関係になる．これを「Snellの法則」と呼ぶ．

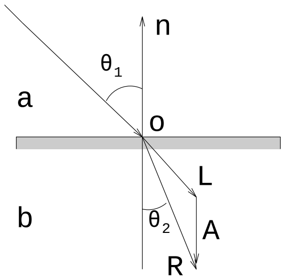


図2: 屈折

異なる屈折率の界面に光が入射すると必ず一部は反射して残りが屈折する．反射光量は$\boldsymbol{theta_1}$が大きければ大きいほど多くなる（Fresnelの公式）．

屈折率の大きな物（たとえば水）から屈折率の小さな（たとえば空気）へ光が向かうとき，ある角度以上では光は屈折せず，すべて反射される．この現象を全反射と呼び，これは，式(1)を変形して，

$$\sin \boldsymbol{\theta_2} = \frac{\boldsymbol{n_1}}{\boldsymbol{n_2}} \sin \boldsymbol{\theta_1} \tag{2}$$

となったときに起きる．$\boldsymbol{n}_2 < \boldsymbol{n}_1$より，$\boldsymbol{\theta}_1$の値によっては$\sin\boldsymbol{\theta}_2$が1より大きくなってしまう．このとき，光線は屈折をせず，全反射をする．式(2)の右辺が1となるときの角$\boldsymbol{\theta}_1$を「臨界角」と呼ぶ．

ここで，屈折する光線のベクトルの求め方を説明する．

入射光側の物体a（屈折率$\boldsymbol{n}_1$）から，屈折光側の物体b（屈折率$\boldsymbol{n}_2$）に光が進むものとし，界面にある点$\boldsymbol{O}$を始点にしたベクトルで考える．この光線ベクトルをLとする．この光線が屈折して角度が変えられるこのときのベクトルを$\boldsymbol{R}$とする．また，法線ベクトルは$\boldsymbol{n}$とする．ただし，Lの長さを$\boldsymbol{n}_1$，Rの長さを$\boldsymbol{n}_2$とする．$\boldsymbol{n}$は単位ベクトルとする．

ベクトルLの終点からベクトル$\boldsymbol{R}$の終点に向かうベクトル$\boldsymbol{A}$を考えると，

$$\boldsymbol{R} = \boldsymbol{L} + \boldsymbol{A} \tag{3}$$

になる．光は密度が高い物質中ほど速く進む．2つの物質中の速度の比が比屈折率なので，ベクトルAは法線と平行になる．

各ベクトルの方向余弦を，

$$\boldsymbol{R} = (\boldsymbol{R}_x, \boldsymbol{R}_y, \boldsymbol{R}_z)$$

$$\boldsymbol{L} = (\boldsymbol{L}_x, \boldsymbol{L}_y, \boldsymbol{L}_z)$$

$$\boldsymbol{n} = (\boldsymbol{n}_x, \boldsymbol{n}_y, \boldsymbol{n}_z)$$

とする．$\boldsymbol{A}$の大きさは，

$$\boldsymbol{n}_2 \cos\boldsymbol{\theta}_2 - \boldsymbol{n}_1 \cos\boldsymbol{\theta}_1$$

なので，式（3）を成分で表すと，

$$\begin{cases} \boldsymbol{n}_2\boldsymbol{R}_x = \boldsymbol{n}_1\boldsymbol{L}_x - \boldsymbol{n}_x(\boldsymbol{n}_2\cos\boldsymbol{\theta}_2 - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \\ \boldsymbol{n}_2\boldsymbol{R}_y = \boldsymbol{n}_1\boldsymbol{L}_y - \boldsymbol{n}_y(\boldsymbol{n}_2\cos\boldsymbol{\theta}_2 - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \\ \boldsymbol{n}_2\boldsymbol{R}_z = \boldsymbol{n}_1\boldsymbol{L}_z - \boldsymbol{n}_z(\boldsymbol{n}_2\cos\boldsymbol{\theta}_2 - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \end{cases} \tag{4}$$

となる．屈折率の式，式（1）を$\sin\boldsymbol{\theta}_2$について解いて2乗すると，

$$\sin^2\boldsymbol{\theta}_2 = \frac{\boldsymbol{n}_1^2}{\boldsymbol{n}_2^2}\sin^2\boldsymbol{\theta}_1$$

となり，

$$\sin^2\boldsymbol{\theta} + \cos^2\boldsymbol{\theta} = \boldsymbol{1}$$

より，

$$\cos\boldsymbol{\theta}_2 = \sqrt{\boldsymbol{1} - \frac{\boldsymbol{n}_1^2}{\boldsymbol{n}_2^2}\sin^2\boldsymbol{\theta}_1}$$

となる．これを式（4）に代入して，

$$\begin{cases} \boldsymbol{n}_2\boldsymbol{R}_x = \boldsymbol{n}_1\boldsymbol{L}_x - (\sqrt{\boldsymbol{n}_2^2 - \boldsymbol{n}_1^2\sin^2\boldsymbol{\theta}_1} - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \\ \boldsymbol{n}_2\boldsymbol{R}_y = \boldsymbol{n}_1\boldsymbol{L}_y - (\sqrt{\boldsymbol{n}_2^2 - \boldsymbol{n}_1^2\sin^2\boldsymbol{\theta}_1} - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \\ \boldsymbol{n}_2\boldsymbol{R}_z = \boldsymbol{n}_1\boldsymbol{L}_z - (\sqrt{\boldsymbol{n}_2^2 - \boldsymbol{n}_1^2\sin^2\boldsymbol{\theta}_1} - \boldsymbol{n}_1\cos\boldsymbol{\theta}_1) \end{cases} \tag{5}$$

となる．この各式を$\boldsymbol{n}_2$で割ると屈折光線のベクトルを求めることができる．また，ルートの中が負になるような角度のときは全反射となる．

2.2 レイトレーシング法の高速化

レイトレーシング法では膨大な計算時間を必要とする．これは，各ピクセルごとに”交点計算”が基本立体の数だけ実行される．物体の形状が複雑になればなるほど基本立体の数は膨大になり，それだけ交点計算が繰り返されることになる．この交点計算には，視線や交点に対する座標変換がともない，計算時間をさらに長くする原因となっている．したがって，より高速な計算機を使うか，交点計算の回数を減らすことが高速化の基本方針となる．以下に，高速化の方法をいくつか挙げる．[1]

バウンディングボリューム 最も無駄なことは，すべての基本立体を対象として交点計算を行っているところである．もし，明らかに交点を持つことのない基本立体が容易に検出可能ならば，実際に交点計算を行わなければならない基本立体の数を大幅に減少させることができる．

空間分割 上で述べたバウンディングボリュームにおいて明らかに無駄な交差判定が起こりうることがある．これは，視線と基本立体や，基本立体相互の位置関係が調べてみないと分からないというデータ構造を使って表現されているからである．これを解決する最も単純な方法として，ボクセル (voxel) で物体を管理する方法があげられる．まず，３次元配列により物体の形状を表現することにする．配列要素を立方体と考え，それぞれの立方体が物体の一部でも含めばその要素を”１”とし，そうでなければ”０”とする．２次元のピクセル (pixel) に対して，３次元の立方体要素をボクセルという．すると，視線と物体との交差判定は３次元配列中での”１”要素の探索として実現できる．すなわち，始点に対応する要素を出発して，順次視線が通過すべき要素を辿り，その要素の値が”１”であるかどうかチェックすればよい．この視線が辿るべき要素の決定は，

3DDDA(3D Digital Differential Analyzer)

と呼ばる３次元のディジタル直線の高速な発生アルゴリズムを用いればよい．

並列計算機による高速化 レイトレーシングのアルゴリズムは，それぞれのピクセルに対する計算がまったく独立である．すなわち，ピクセルの数だけのコンピュータが使えるのならば，１ピクセルのための計算時間で１画面すべての計算が行える．もちろん，反射，屈折，物体の表示されないピクセルなどあるので，ピクセルによって計算時間はまちまちである．ところが，現時点ではピクセルの数だけの膨大な数のコンピュータを使うことは非現実的なので，限られた台数をうまく使って速いものから次の未処理のピクセル計算させる方法がより妥当といえる．

3　画面分割によるレイトレーシング法の並列化

3.1　マルチトランスピュータシステム

3.1.1　アーキテクチャ

トランスピュータは，英国のINMOS社が開発した並列計算機プロセッサである. ここではCPU機能として浮動小数点ユニット (FPU) 内蔵のT805を使用し, メモリのアドレス空間4Gバイトの内4KBの内部メモリを内蔵している. このアドレス空間に対しては高速なアクセスが可能である. 各トランスピュータは, 他のトランスピュータと接続する10～20Mbpsのリンクインターフェイス4つを内蔵している．実行転送効率は,T800の場合片側通信で最大1740KByte/secというDMA転送なみのスピードになる.

並列プログラムは複数のトランスピュータで行う外部チャンネルの場合と，単一のトランスピュータで行う内部チャンネルの場合がある．[5]

3.1.2　相互結合網

トランスピュータの結合網には図3に示すようなものがある．

本研究で用いた並列計算機では４本のリンクにより６４台までのプロセッサをトーラス結合，ハイパーキューブ結合することが可能である. リンク４本を利用し図のような２進４次元のハイパーキューブ接続も可能である.

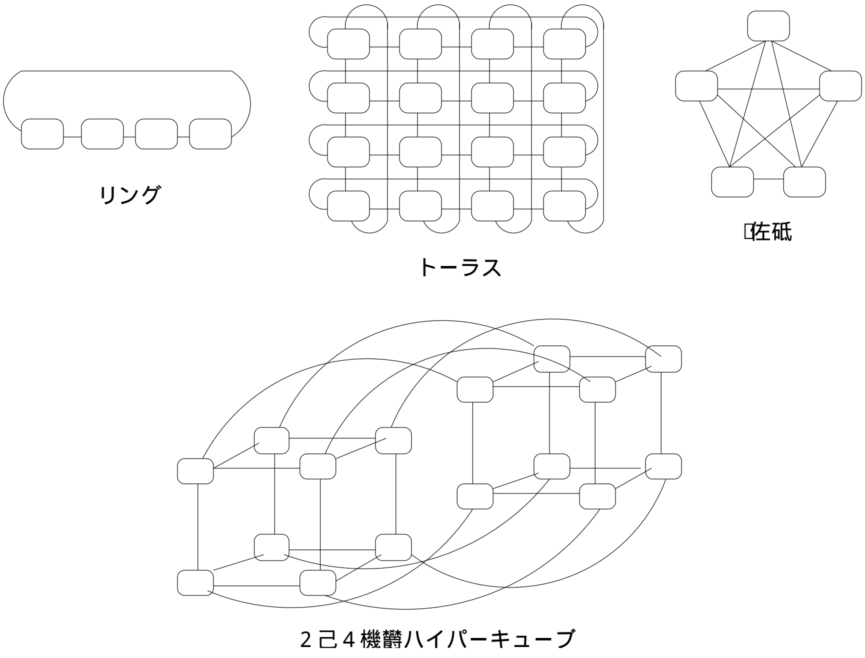


図 3: 相互結合網

3.1.3 Parallel C

Parallel C は，トランスピュータ用の並列処理言語として開発された通信, マルチスレッド (Multi Thread) の機能が拡張された C コンパイラである．

Parallel C による並列計算では各トランスピュータに複数のタスクが存在し，タスク間は通信によりデータを交換することができる．その際，タスク間の通信は各チャンネルが 1 対 1 の関係で，かつ送受信の要求が一致した場合転送が開始される．要求が片方のみの場合，もう一方の要求がでるまでウェイト状態となる．また，タスクの中にはスレッドと呼ばれるプログラム中でシーケンシャルに処理が進められる部分が複数存在し (Multi Thread)，タスク内でのスレッドは，共通の変数を使用できる．ただし，複数のスレッドで実行する場合は，セマフォにより排他制御しなければならない．[5]

3.2　画面等分割

レイトレーシングの高速化技法として，並列的にピクセルを計算する方法がある．しかし，2.2 節で述べたように各ピクセル分のプロセッサを用いることは現実的ではない．そこで，画面を実行可能なプロセッサの数に分割し，並列計算する方法が挙げられる．図 4に示すように画面を等分割し，それぞれの範囲を各プロセッサに割り当てる．

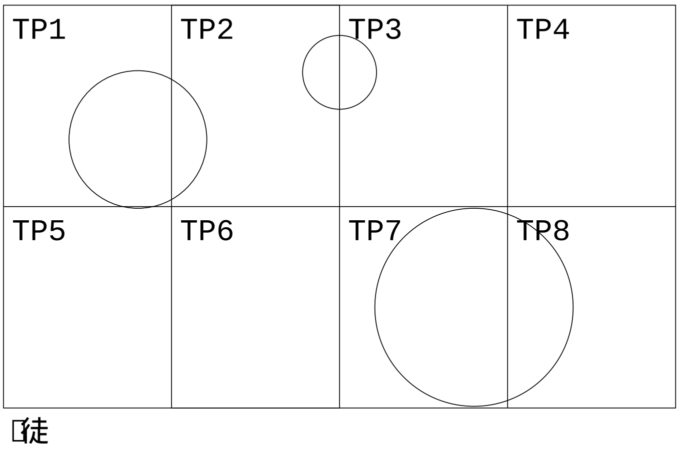


図 4: 画面等分割 (8 台の例)

しかし，前にも述べたようにピクセルごとに計算時間がまちまちで，ある分割された範囲内に負荷のかかる（計算時間のかかる）部分が集中してあると，当然その範囲はその他の範囲よりも時間がかかることになる．

3.3　画面負荷均衡分割

3.2 節の等分割において，画面を等分割しただけではそれぞれ分割された範囲によって計算時間の大きさが変わる．そこで，あらかじめ各ピクセルごとに負荷のかかり方が分かれば均等に負荷のかかるよう画面を分割することができる（図5参照）．しかし，各ピクセルの正確な計算時間（負荷）を知るためには，1度レイトレーシングプログラムを実行するしか方法はないであろう．それでは，高速に計算する意味がないのではという問題が生じる．

ここで，表示される画像の物体記述データより短時間に負荷の大きさを知るには，いくらかの犠牲は必要になってくる．この方法により，もし8台のプロセッサで計算を行えば1台のときの計算時間と比べ約8倍速い結果が得られる．

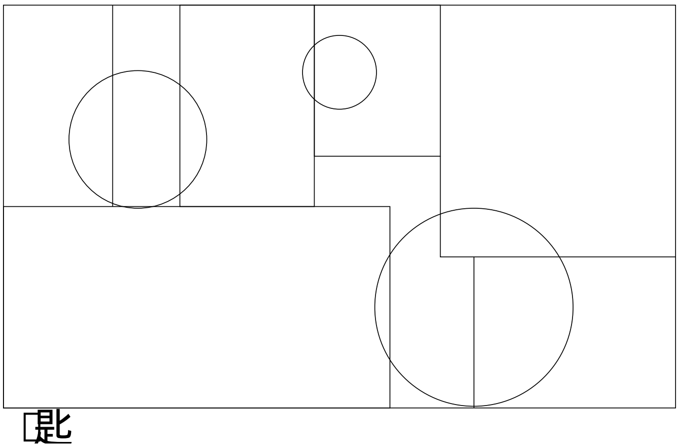
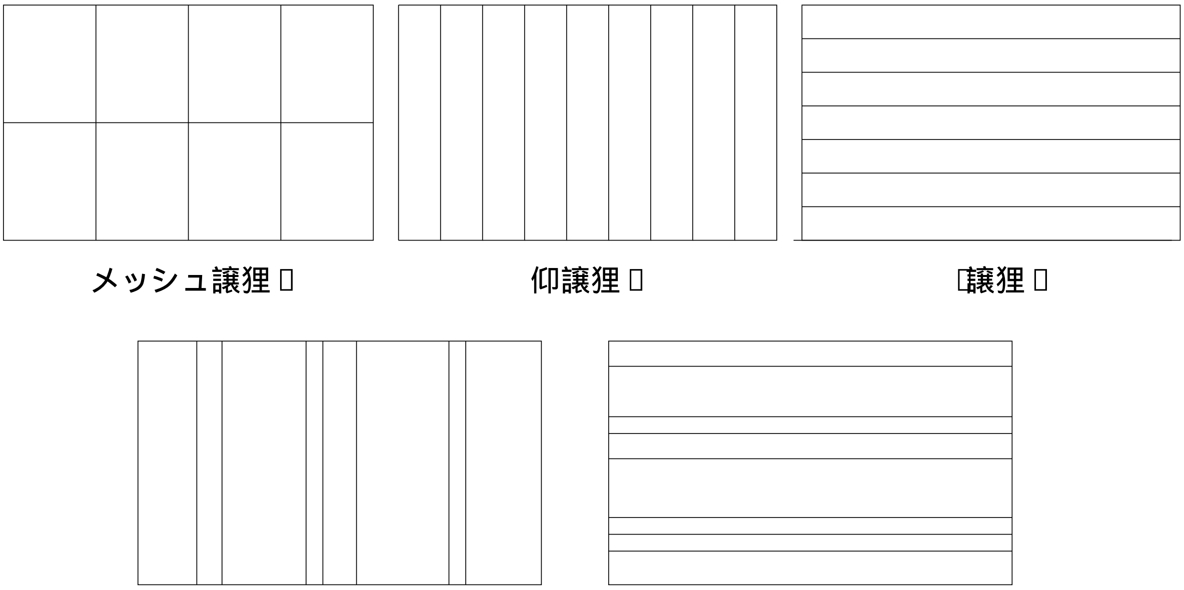


図 5: 画面負荷均衡分割

3.4 マルチトランスピュータによる並列計算

本研究では，画面等分割（メッシュ，縦，横），画面負荷均衡分割（縦，横）(図 6）について並列化を行った．



メッシュ譲狸

仰譲狸

譲狸

仰醍 箇狸

醍 箇狸

図 6: 画面分割方法

プロセッサがｎ台のとき画面をｎ個に分割し，各プロセッサにその範囲の計算をさせる方法である．トランスピュータはリング接続を用い結果を回収する (図 7参照)．各プロセッサには，あらかじめ分割された範囲の左上座標と右下座標を与え，処理がすみ次第, 結果を Root に送り返す．データを Root に戻す際，少なくとも１つのプロセッサを経由しなければならない．そうすると，そのプロセッサが実行中ならば，その処理が終わるまで待たされることになる．これでは，全体の処理時間が増すばかりである．そこで Multi thread を用いて通信部と，計算部を同時に実行させた．図 8に示すように，各プロセッサは通信部 (communication part) と計算部 (calucurate part) の 2 つのスレッドをもち，通信部は左のトランスピュータからの入力データを右のトランスピュータに送り出し，計算部はレイトレーシング法の本体であり，割り当てられた分割範囲の計算が終わり次第，右のトランスピュータへ送り出す．通信部，計算部共に同じ出力ポートを使用しているため競合を生じるがセマフォにより調停している．

画面 (640×400) を等分割（縦，横）する際，割り切れないプロセッサ数のときは余った行数を１ラインずつ適当なプロセッサに振り分ける．メッシュ等分割では分割数を限定する．

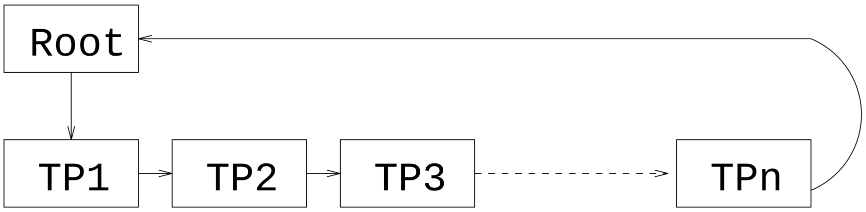


図 7: リング接続による結果の回収

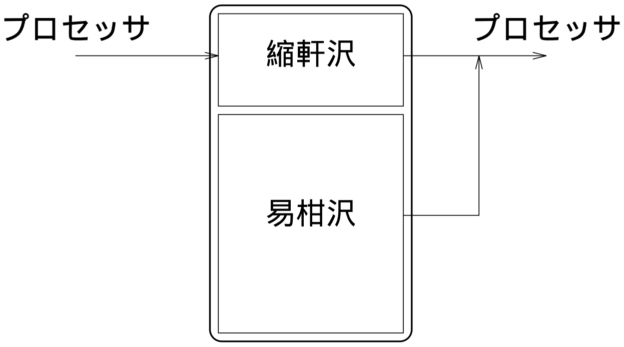


図 8: マルチスレッド

4　画面分割アルゴリズム

この章では 3.3 節で述べた画面負荷均衡分割において，本研究で作成した画面分割アルゴリズムを説明する．記述データをもとにディスプレイの縦と横のラインごとに負荷を決定し，最大 64 の範囲に分割する．

4.1　物体表示位置予測

レイトレーシングの方法で各ピクセルごとに物体の存在を調べたのでは，それだけでも計算時間を要する．よって，物体の記述データより物体の画面上でのおよその位置を計算することにする．また，本研究で使用する物体は球のみである．実際に記述データより使用するものは，

視点座標，球のパラメータ（半径），平行移動量（中心の座標）

である．

図 9において，視点 Iとある物体（球）の中心 Oを結ぶ直線を考える．その直線上で画面と交わる点を画面上での物体の中心点 O'とする．次に Iと Oの距離と Iと O'の距離を求め，比を求める．その比によって，画面上での物体の半径を求め画面上での物体の位置が決定する．本研究では，透視投影法を用いるため実際に画面上に表示される物体は正確な円ではないが，おおよその位置として定める．

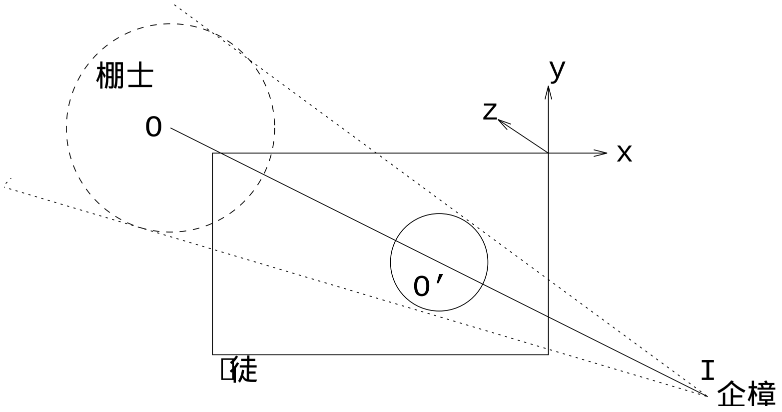


図 9: 物体位置予測

4.2 ライン単位の負荷予測

画面上での物体の位置が分かると，次にその物体が画面上でどのラインに属するかは，容易に分かる．画面上の物体の中心点を $(\boldsymbol{O'_x}, \boldsymbol{O'_y})$ とし，半径を $\boldsymbol{r'}$ とすると，縦のラインであれば $\boldsymbol{O'_x} - \boldsymbol{r'} \sim \boldsymbol{O'_x} + \boldsymbol{r'}$ までのラインに負荷がかかることになる．それぞれのラインについて長さFは，

$$\boldsymbol{F = 2 * \sqrt{r'^2 - a^2}}$$

より求められる．(図10参照)

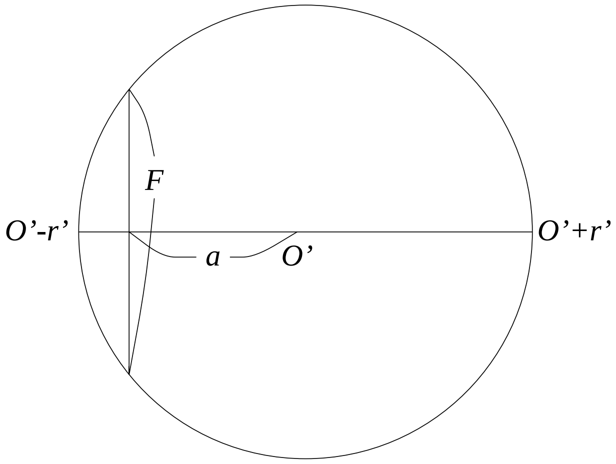


図 10: 負荷計算

この式から，負荷を決定するに当たって記述データの，

ハイライト，鏡面反射，マッピング，屈折率

と，物体の数，そして鏡面物体の数，屈折する物体の数により負荷を計算する．

今物体が100あるとすると，次のように分類される．ただし，以下の条件は詳しい計算でないため実際の研究では結果より負荷をどの程度にするのが最もいいか考慮に入れる．

物体の存在しない場合 物体の存在しないピックセルでは100回の交点計算のみ行われるので，負荷を100と置く．

通常物体（鏡面, 屈折の条件なし） 通常物体では交点計算100回と，その物体表面と光源の間に物体がないか調べる．物体がない場合99回の交点計算が行われ，物体が存在する場合では交点計算が最低1回で求められる．よって，平均を取って負荷150とする．

鏡面物体 鏡面物体では，はじめの交点計算100回と光源との間で50回，そして映し出される物体があるかどうかの交点計算99回，そして映し出される物体において新たに交点計算する必要がある. この際，その映し出される物体がまた通常か鏡面か屈折かと言う問題がでてきて繰り返し調べたのでは時間がかかるのでおよそ300程度とする．

屈折する物体　屈折する物体では，鏡面の条件にさらに視線が物体内部に進入しさらに物体からでる際，反射屈折をしさらに交点計算が行われる　(図11参照）．よって，700 程度とする．

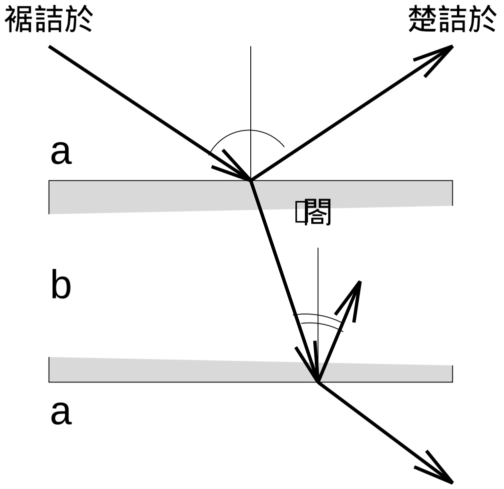


図 11: 屈折と反射

4.3　画面分割

縦または横の各ラインの負荷が決定されると、縦ラインなら 640 ライン、横ラインなら 400 ラインすべての負荷の合計を画面分割数で割ることにより負荷の平均化が図られる　(図 12参照)。これにより、画面のおよその負荷均衡が図られレイトレーシングによる計算時間が平均化されることになる。

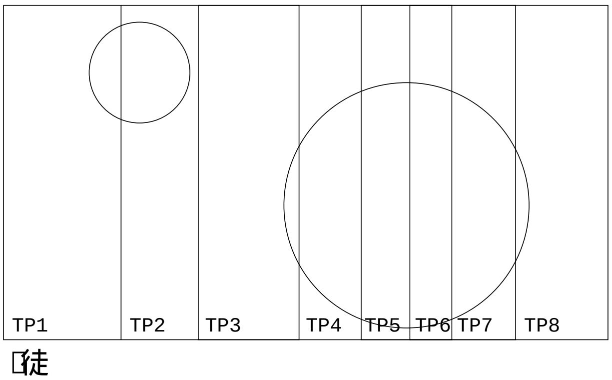


図 12: 画面分割

5 実験と評価

5.1 実験方法

トランスピュータ64台を用いて，画面等分割（縦，横，メッシュ），画面負荷均衡分割（縦，横）の実行時間と相対速度を求める．画面負荷均衡分割では負荷計算の時間も含めたものとする．相対速度は，次式で定義した値とする．また，各トランスピュータに振り分けられる各分割方法での負荷の分布を調べる．実験に用いるテストデータは表5.1に示す4データで, 球のみについて実行した．

$$相対速度 = \frac{1\,台のトランスピュータによる実行時間}{n\,台のトランスピュータによる実行時間}$$

表 1: テストデータ

データ	内容
Test01	縦に広い分布
Test02	横に広い分布
Test03	画面に均等な分布
Test04	画面に適当な分布

5.2 実験結果

表2～4にTest1～Test4の各分割画面での実行時間と相対速度を示す. 各表の上段は実行時間, 下段は相対速度を示す. 単位は秒である. また相対速度をグラフにしたものを図13～16に示し, 作成された画像を図17～20に示す.

表 2: test1 の実行時間と相対速度

画面分割方法	1	4	8	16	32	64
メッシュ等分割	409.00	103.00	76.827	45.978	39.307	27.662
相対速度	1.0000	3.9709	5.3236	8.8956	10.405	14.786
縦等分割		153.00	125.99	111.68	71.876	40.893
相対速度		2.6732	3.2464	3.6622	5.6904	10.002
横等分割		117.00	74.046	38.781	20.747	11.560
相対速度		3.4957	5.5236	10.546	19.714	35.381
縦負荷均衡分割		121.00	68.905	40.018	21.019	12.726
相対速度		3.3802	5.9357	10.220	19.459	32.139
横負荷均衡分割		105.00	57.054	32.480	18.755	10.868
相対速度		3.8952	7.1686	12.592	21.808	37.633

表 3: Test2 の実行時間と相対速度

画面分割方法	1	4	8	16	32	64
メッシュ等分割	545.00	138.00	92.169	74.000	51.693	47.169
相対速度	1.0000	3.9493	5.9131	7.3649	10.543	11.554
縦等分割		182.00	120.13	71.357	36.515	18.850
相対速度		2.9945	4.5368	7.6377	14.925	28.912
横等分割		202.00	164.75	113.24	67.876	31.849
相対速度		2.6980	3.3081	4.8129	8.0293	17.112
縦負荷均衡分割		139.00	76.132	43.828	25.223	14.247
相対速度		3.9209	7.1586	12.435	21.607	38.254
横負荷均衡分割		170.00	86.785	46.384	26.268	15.821
相対速度		3.2059	6.2799	11.750	20.748	34.448

表 4: Test3 の実行時間と相対速度

画面分割方法	1	4	8	16	32	64
メッシュ等分割	6649.0	1668.0	933.33	521.84	276.84	139.18
相対速度	1.0000	3.9862	7.1239	12.742	24.018	47.771
縦等分割		1862.0	982.55	522.26	276.33	143.17
相対速度		3.5709	6.7671	12.731	24.062	46.440
横等分割		1843.0	922.34	486.86	294.67	162.58
相対速度		3.6077	7.2089	13.657	22.564	40.896
縦負荷均衡分割		1742.0	887.71	444.77	239.11	128.87
相対速度		3.8169	7.4901	14.949	27.807	51.596
横負荷均衡分割		1749.0	875.37	469.00	247.35	140.14
相対速度		3.8016	7.5956	14.177	26.881	47.447

表 5: Test4 の実行時間と相対速度

画面分割方法	1	4	8	16	32	64
メッシュ等分割	8054.0	3185.0	1972.5	1141.6	710.60	371.744
相対速度	1.0000	2.5287	4.0832	7.0549	11.334	21.665
縦等分割		2434.0	1635.0	855.80	430.55	217.56
相対速度		3.3090	4.9261	9.4111	18.706	37.020
横等分割		3466.0	1976.9	1033.7	501.82	251.12
相対速度		2.3237	4.0741	7.7916	16.050	32.073
縦負荷均衡分割		2104.0	1078.3	543.18	287.62	153.62
相対速度		3.8279	7.4692	14.827	28.003	52.427
横負荷均衡分割		2141.0	1071.5	547.31	292.81	167.65
相対速度		3.7618	7.5164	14.716	27.506	48.042

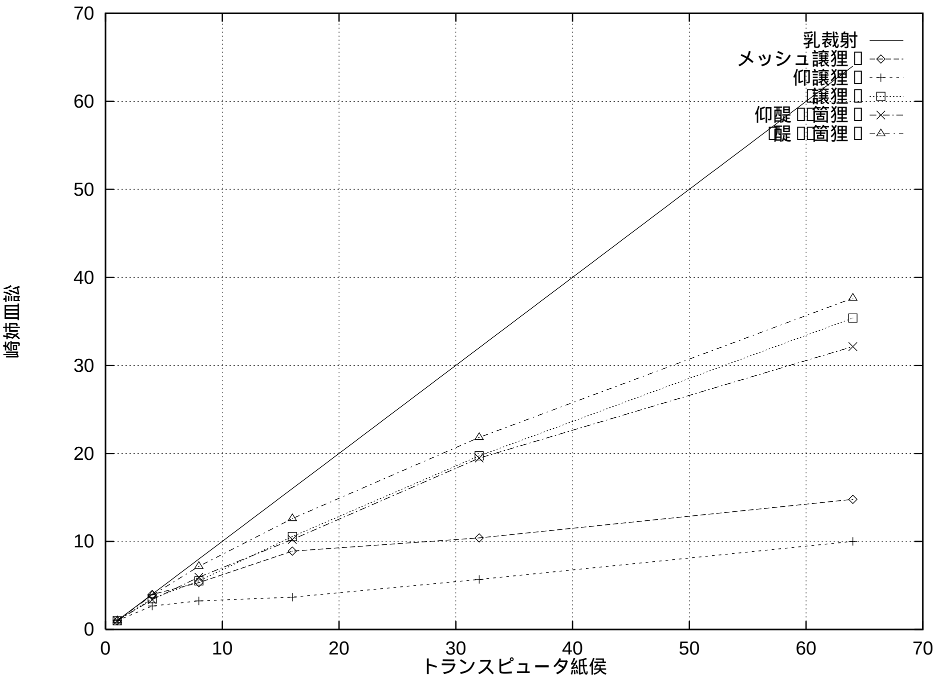


図 13: Test1 の各分割による相対速度

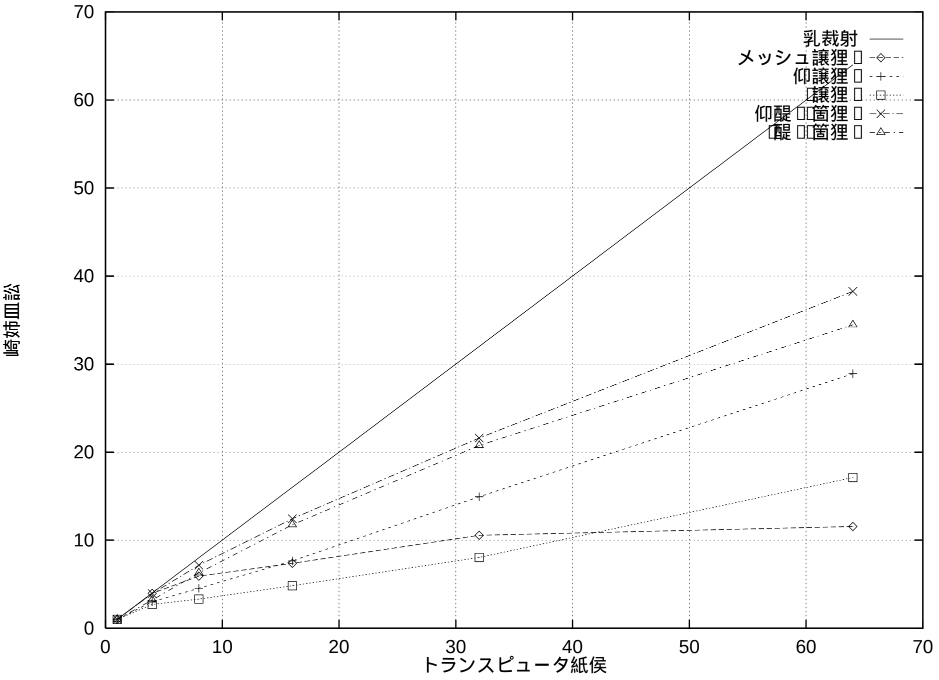


図 14: Test2 の各分割による相対速度

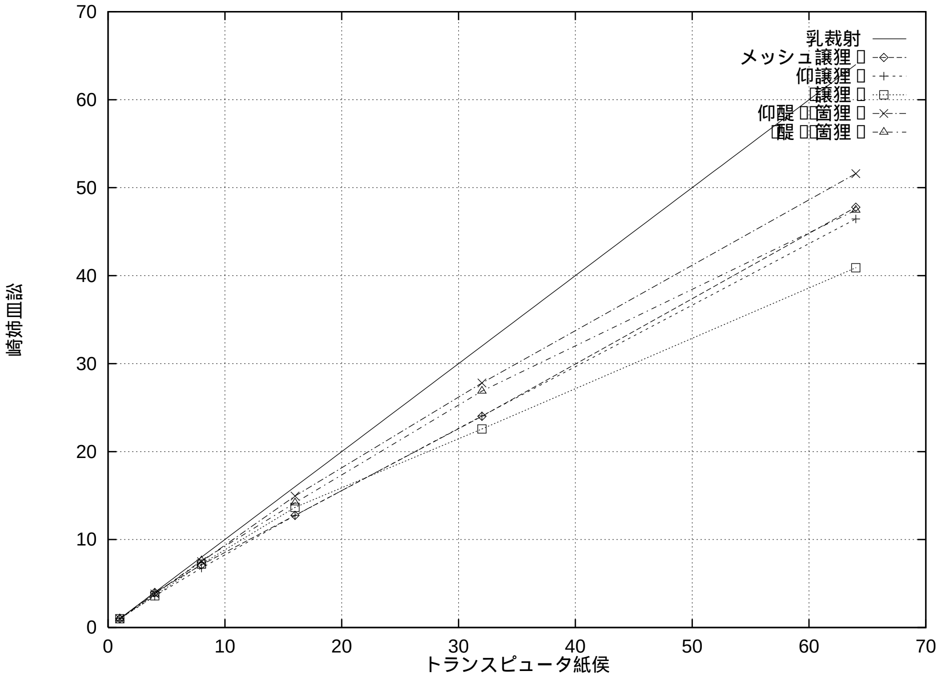


図 15: Test3 の各分割による相対速度

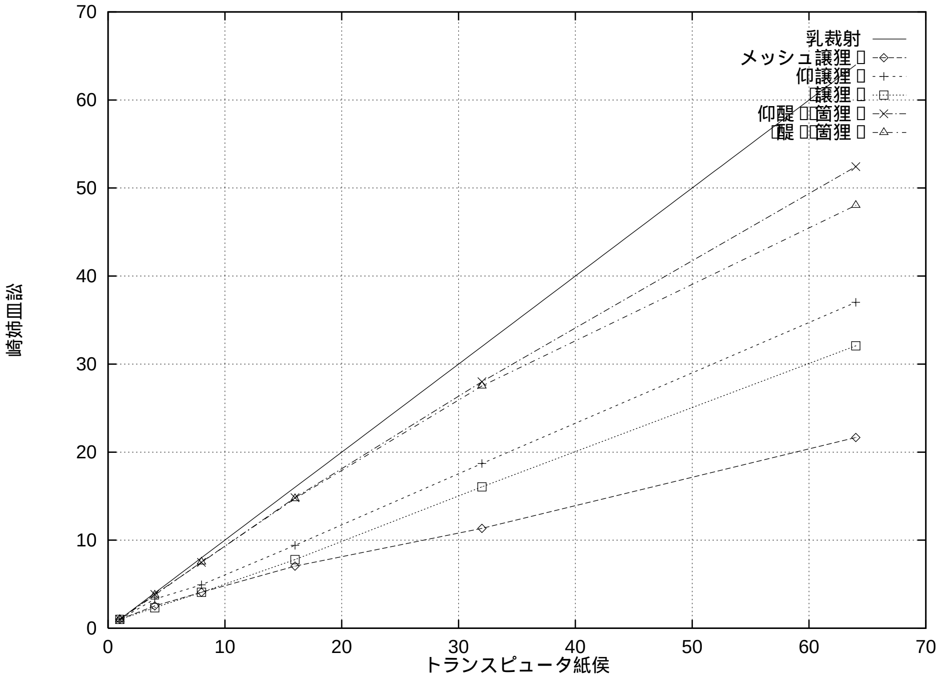


図 16: Test4 の各分割による相対速度

図 17: Test1 画像

図 18: Test2 画像

図 19: Test3 画像

図 20: Test4 画像

5.3 考察

Test1 画像では、横負荷均衡分割の相対速度が最も良い結果であるが、縦負荷均衡分割よりも横等分割の方がいい結果となっている. これは、負荷予測がおよその計算であることが１つ挙げられる. 別の原因として、画像から見て分かるように画面の中央に集中して負荷がかかっている. このため画面中央部で横に分割する際、１ラインずらしただけでかなりの負荷の変動がおこる. よって、正確な負荷均衡が図られない。これはトランスピュータ４台のときを見ると分かるように、１ラインが負荷の大きな部分とならないときには縦での負荷均衡も図られている.

Test2 画像では、n 台で n 倍の相対速度までは行かないが等分割に比べるとかなりの負荷均衡が行われているといえよう. これは、Test1 では横１ラインが 640 ピクセルであるのに対して１ライン 400 ピクセルと１ラインでの負荷の変動が小さいことが挙げられる. また、画像がやや左にあることから、等分割には不利であったといえる.

Test3 画像では、ほぼ同じような相対速度が得られたが、やはり負荷均衡でも縦の方がよりいい結果を出している. 画像は、ほぼ均等に物体が配置されておりどの分割方法においても均一化がとれているといえる.

Test4 画像は、条件を無視した画像として実験したが、負荷均衡分割の長所が一番出ているといえる. グラフから見ても分かるように、一番不利なメッシュ等分割と、縦負荷均衡分割では、21 倍から 52 倍と良い結果が得られた.

今回使用したテストデータでは、記述データよりどのような負荷の分布の画像を生成するのか単純にした. これは、どの分割方法が有利であるか、予想を付けやすくするためである. 実際にその予想は当たったが、トランスピュータ 64 台で 64 倍に近い結果が得られたとは言えない. この理由として画面負荷均衡分割では分割単位が縦で 400 ピクセル、横で 640 ピクセルとかなり大きな値となってしまうからである. また、負荷計算についても多少の問題がある. 実際に物体表面が鏡面で反射などの現象がおきてもその先は計算に入れていないことである. これは、高速化を図る上でやむを得ないが、実際に鏡面、屈折といった部分に大きな負荷がかかっていることは確かである。Test1 では、負荷均衡を図ってもトランスピュータ 64 台で最高 38 倍と 64 倍にはほど遠い.

6 おわりに

本研究では、レイトレーシング法の画像を画面分割によりトランスピュータを用いて並列化を行った. レイトレーシング法はリアルな画像を作成できることができる反面、かなりの計算時間を要する. しかし、レイトレーシングのアルゴリズムは比較的簡単で、しかも各ピクセルの計算が独立しているため並列生の頻度は高い.

画面等分割において各分割された範囲の計算時間が違うため負荷均衡分割を用い、等分割のメッシュ、縦、横の 3 つと負荷均衡分割の縦、横と計 5 つについて行った. 結果としてメッシュ等分割の相対速度 21 倍 (最小) から縦等分割の 52 倍 (最大) までの高速化が行われた.

しかし、画像により有利な画面分割方法がありどの分割方法が最も良いというわけではない. これは、負荷の計算がおよそであることに原因があると考えられる.

謝辞

本研究の機会を与えて下さり、数々の助言を頂きました山崎勝弘教授、原田博之氏に心より深く感謝の意を表します。

また、本研究にあたり、いろいろな面で貴重な御意見や励ましを頂きました本研究室の皆様に心より感謝致します。

参考文献

[1] 千葉則茂/村岡一信, “ＣによるＣＧレイトレーシング”, サイエンス社 (1991),pp.9–41.

[2] 橋本英美, “Ｃ言語による３Ｄ− ＣＧ入門”, 工学社 (1994),pp.111–132.

[3] 小堀研一/春日久美子, “ＣＡＤ ・ＣＧ基本用語集”, 工業調査会 (1994),pp.160.

[4] 水上孝一, “プログラミング入門シリーズ　コンピュータグラフィックス”, 朝倉書店 (1989),pp.155–156.

[5] 山本正樹/中井秦明/村上安範, “トランスピュータ入門”, 日刊工業新聞社 (1990),pp.159–214.

[6] 原田博之, “マルチトランスピュータシステムによるレイトレーシング法の高速化”, 立命館大学理工学部情報工学科卒業論文 (1993).

[7] 原田博之, “マルチプロセッサシステムによる画面／空間分割によるレイトレーシングの高速化”, 立命館大学理工学部情報システム学専攻修士論文 (1995).

[8] 原田博之, “画面分割と空間分割によるレイトレーシング法の高速化”, 情報処理学会 (1995).

[9] Adelene Ng , Mel Slater, “A Multiprocessor Implementation of Radiosity”, COMPUTER GRAPHICS forum,Volume 12(1993),number 5,pp.329–342.

[10] Daniel R. Baum,James M.Winget, “Real Time Radiosity Through Parallel Processing and Hardware Acceleration”, Silicon Graphics Computer Systems,2011 N.Shoreline Blvd.Mountain View,CA94039-7311.