修士論文

空間分割による並列レイトレーシング法

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 :6114950113-3

1997 年 2 月 14 日提出

立命館大学 理工学研究科
修士課程 情報システム学専攻

摘要

本論文では、空間分割を用いたレイトレーシング法による画像生成の並列化について述べる。レイトレーシング法は、視点から出た視線を追跡して交差する物体を調べ、最も手前にある物体を表示する手法である。原理は非常に単純であるが、全ピクセルについて計算することと、各ピクセルごとに全物体との交点計算を行うことで膨大な計算時間を必要とする。しかし、その計算には多くの並列性を含んでいる。そこで、本研究では分散メモリ型マルチプロセッサであるトランスピュータ64台をトーラス接続し、空間分割による並列化を行うことで高速化を行った。空間分割では等分割と適応型分割の2通りの実験を行い、それらの結果を画面分割での結果と比較した。

画面分割は、各ピクセルが独立に計算できることを利用した並列化技法である。インターリーブ分割とは、画面上を8×8ピクセル単位に各プロセッサに割り当てる手法で、64プロセッサを用いて（リング接続）約60~64倍の速度向上をあげている。

空間分割では、空間をボクセルと呼ばれる部分空間に分割し、そこに属する物体のみを登録する。すべて同じ大きさのボクセルを用いた等分割、物体の多い所は細かく分割した適応型分割をとりあげた。これにより、無駄な交差判定が削減でき高速化が図れる。また、ボクセルを担当する各プロセッサは全物体データを持つ必要がないので、1プロセッサに全てを登録できない大規模な画像でも実行可能である。トランスピュータ上で実装するために、ボクセル決定、ボクセル間移動アルゴリズム、各プロセッサのタスクとスレッド、及びパケットによるデータ収集方法について述べる。

実験の結果、画面分割と等空間分割による比較では、256物体以上扱うとき等空間分割の結果の方が良く、空間分割の有効性が確認できた。また、等空間分割と適応型空間分割による比較では、SPD(Standard Procedural Databases)から6データを用いて実験した結果、5つのデータで等分割の0.29~0.72倍に実行時間を短縮できた。

目 次

図目次

表目次

1　はじめに

コンピュータグラフィックス (CG) は過去２５年くらいの間に急速に進歩・普及し、我々にとって身近なものになった。現在では、TV のコマーシャルやゲーム、また映画などで一般の人々の間にもコンピュータグラフィックスが浸透している。これは、CG 技術の着実な積み上げと計算機技術自体の進歩により実現されてきた。現在、CG は情報処理の一つの重要なツールとして定着してきている [1]。

物体表面での光の反射は物体の材質によって異なった性質を持っている。それをモデル化して反射計算に組み込むことによりさまざまな物体がリアルに表示できる。基本的なモデルとしては反射モデル、テクスチャマッピング、レイトレーシング法、微粒子集合による散乱モデルなどが提案されている [1]–[3]。また、光の相互反射を利用し、より現実的な画像を表示するラジオシティ法 [4]–[8] などが挙げられる。本研究では、その中でもレイトレーシング法の並列化について述べる。

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。視線は、ピクセルの数、反射光の数、透過光の数の和だけ生成される。さらに、視線が鏡面物体や透明物体に当たれば、それだけ生成される視線の数は増える。それぞれの視線追跡では、交差判定が基本立体（球や三角形など）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、それだけ交差判定が繰り返されることになる。この交差判定には、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 [9]–[20]。

レイトレーシング法の高速化技法として、ソフトウェア的手法とハードウェア的手法がある。ソフトウェア的手法として、レイトレーシング法の遅さの原因である交差判定を減らすための、空間分割法 [21]、バウンディングボリューム分割法 [9]、などの方法が考え出されている。ハードウェア的手法として、並列計算機を用いた方法がある。その中で、レイトレーシング法がピクセル単位に独立に計算できることを利用し、画面を分割しそれをプロセッサに振り分け並列に計算させる画面分割法などが挙げられる [11]–[18]。本研究では、英国の INMOS 社で開発された分散メモリ型並列計算機であるトランスピュータを用いる。25MHz の T805 が 64 台用意され、これらを各トランスピュータが持つ 4 本のシリアルリンクにより、トーラス接続したマルチトランスピュータシステム上で実験を行う。

本研究では、マルチトランスピュータシステム上で空間分割法を用いて高速化を図る。レイトレーシング法は各ピクセルが独立に計算できることから、従来３通りの画面分割法による並列化を行ってきた。それらは、画面を縦、横、またはメッシュ上に等分割しプロ

セッサに割り当てる画面等分割、基本立体の式のパラメータから、画面上での物体の位置を縦、横、メッシュの点で予測し分割する負荷均衡分割、及びあるピクセルの周りのピクセルもほぼ同じ負荷であると考え、画面上を 8×8 ピクセル単位に分割し、それらに対してプロセッサ64台を割り当てるインターリーブ分割である。その中でもインターリーブ分割では、プロセッサ1台時に対して64プロセッサ使用時の実行速度が、約60~64倍となった[18]。本研究では、等空間分割との比較に画面インターリーブ分割をとりあげる。画面分割では各プロセッサに全物体を登録するため、最大でも物体数250までの結果しか得られていない。

そこで、別の視点からみた高速化として空間分割を取り上げる。レイトレーシング法では、全物体との交差判定を行うため、膨大な計算時間を要する。空間分割は、その交差判定を減らすことを目的としている。まず、物体の存在する空間をボクセルと呼ばれる部分空間に分割し、各プロセッサに担当させる。これにより、すべての物体データを持つ必要がなくなり、各プロセッサは、そこに属している物体データのみもてばよい。光線はそのボクセルに属する物体との交差判定を行うことになり、その回数を軽減できる。分割方法として、すべて一定の大きさに分割する等分割[23]、物体数をできるだけ均等にするための適応型分割[24]について述べる。この空間分割をトランスピュータに実装するアルゴリズムを述べ、さらに光線がどのボクセルを移動するかを知る3DDDA、各プロセッサ間のパケットのやりとりを行うためのボクセル間移動アルゴリズムについて述べる。実験で、画面インターリーブ分割と等空間分割を、共にプロセッサ64台使用したときについて実行時間から比較、評価する。さらに、等空間分割と適応型空間分割を実行時間から比較、評価する。

本論文では、2章でレイトレーシング法の特徴、画像生成法、そして高速化技法について述べ、3章で本研究で用いたトランスピュータシステムについて述べる。4章では、画面分割の中で最もよい結果が得られたインターリーブ分割、及び等空間分割、適応型空間分割の並列化技法について述べる。5章では、インターリーブ分割、等空間分割、および適応型空間分割についてそれぞれ実行時間を比較し、その考察を行う。6章で本研究の成果と今後の課題を述べる。

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、それ自身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 (図 1)。以下にレイトレーシング法の特徴を示す。

- 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。
  - 光線を１本１本追跡するので演算に時間がかかるが、各光線の処理は独立なので並列処理に向いている。

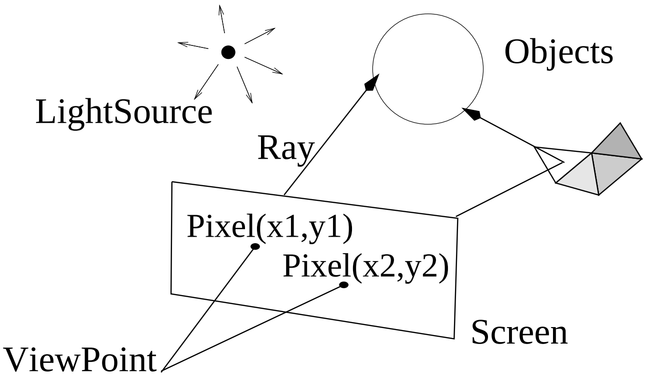


図 1: レイトレーシング法

2.2　レイトレーシング法の作図

まず、3 次元形状モデルを適当な座標系のもとで 3 次元空間に配置、表現する必要がある。実際に表示されるスクリーンのサイズは 640×400 のサイズとし、あらかじめ、視点 (v_x, v_y, v_z)、目標点 (o_x, o_y, o_z)、目標点までの距離、および傾きを決定する。[9]

2.2.1　球と三角形の表示

レイトレーシング計算を行うとき、光線による直線と物体との交点を求める必要がある。そのためには、物体は方程式で定義されなければならない。ここでは球と三角形について、光線との交点計算について示す。

まず直線の定義として、スクリーン上の点を (p_x, p_y, p_z)、視点を $\boldsymbol{V} = (v_x, v_y, v_z)$、方向余弦を $\boldsymbol{E} = (E_x, E_y, E_z)$ とすると直線のパラメータ形式は、

$$\begin{aligned} x = E_x t + v_x, E_x &= \frac{p_x - v_x}{t}\iota \\ y = E_y t + v_y, E_y &= \frac{p_y - v_y}{t}\iota \\ z = E_z t + v_z, E_z &= \frac{p_z - v_z}{t}\iota \end{aligned} \tag{1}$$

$$\iota = \sqrt{(p_x - v_x)^2 + (p_y - v_y)^2 (p_z - v_z)^2}$$

となる．t は、視点から直線上の点のまでの距離、ιは光源からの距離である．

ここで、半径が rで中心座標が (x_0, y_0, z_0) であるような球面上の点 (x, y, z) が満足すべき方程式は、

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = r^2$$

である。したがって、交点計算は t に関する 2 次方程式

$$At^2 + Bt + C = 0$$

$$A = E_x^2 + E_y^2 + E_z^2 = 1$$

$$B = 2E_x(v_x - x_0) + 2E_y(v_y - y_0) + 2E_z(v_z - z_0)$$

$$C = (v_x - x_0)^2 + (v_y - y_0)^2 + (v_z - z_0)^2 - r^2$$

を解けばよい。したがって、

$$D = B^2 - 4AC \geq 0$$

のときに交点をもち、そのとき t は、

$$t = \frac{(-B \pm \sqrt{D})}{2}$$

と表せる。

次に、三角形とは 3 頂点を持つ多角形のことで、以下の 3×3 の行列$\boldsymbol{P}_{mn}$によって定義される。

$$\boldsymbol{P}_{mn} = \begin{bmatrix} a_{x0} & a_{x1} & a_{x2} \\ a_{y0} & a_{y1} & a_{y2} \\ a_{z0} & a_{z1} & a_{z2} \end{bmatrix} \tag{2}$$

行列の行要素が x, y, zの座標値を表し、列要素が各頂点に対応している。

ここで光線の原点$\boldsymbol{V}$を頂点とし, 三角形を底とするような三角錐を考えると、視線ベクトル$\boldsymbol{E}$は$\boldsymbol{P}_{mn}$の線形結合として表すことができる。

$$\boldsymbol{E} = \boldsymbol{P}_{mn}\boldsymbol{A} \tag{3}$$

ただし、$\boldsymbol{A}^t = [a_0 \ a_1 \ a_2]$

さらに視線ベクトル E が三角形と交点を持つためには、

$$a_0 \geq 0, a_1 \geq 0, a_2 \geq 0$$

かつ

$$|a_0| + |a_1| + |a_2| \neq 0$$

である必要がある。

また、視点からの距離は、

$$t = \frac{1}{a_0 + a_1 + a_2}$$

によって求められる。

2.2.2 陰影

前節で示した画像に立体感を付けるため、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を求める。通常、光源と反対側の物体表面には光があたらない陰 (Shade) の部分が生じ、物体は自分の形に応じた影 (Shadow) を光源と反対側にある他の物体に投げかける (図 2)。この色調の変化や影の存在が物体の立体感を感じさせる要因の一つである。

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。CG 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ている。光の種類は一般に、

- 反射光
  - 拡散反射光 (diffusely reflectd light)

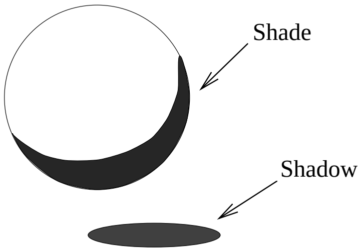


図 2: 陰 (Shade) と影 (Shadow)

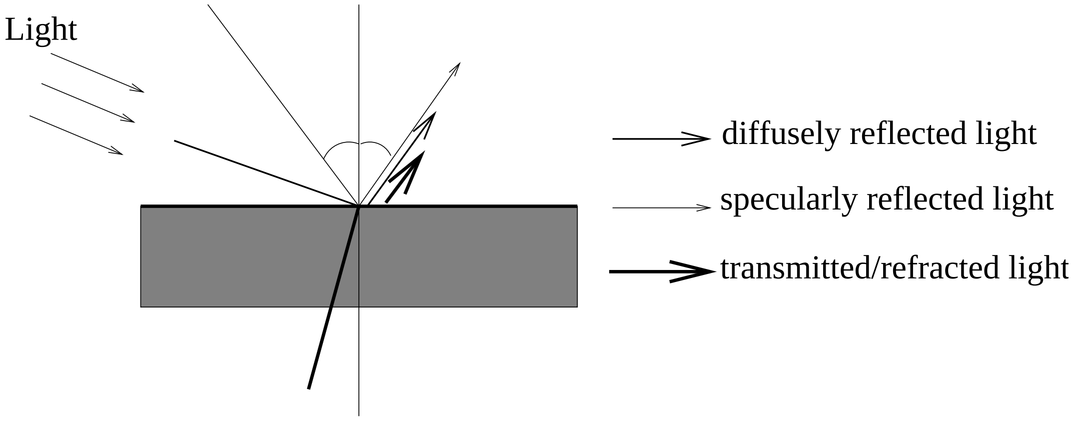


図 3: 光の種類

  - 鏡面反射光 (specularly reflected light)

- 透過・屈折光 (transmitted/refracted light)

に分けられる (図 3)。

ここで、点光源による陰影について示す。床の近くに点光源を置いた状態を想定すると、点光源の真下が最も明るく、周辺ほど暗くなっている。また、点光源を床から離していくと、全体的に暗くなっていく。したがって、表面の輝度を I_d、点光源の光度を I_q、面の単位法線ベクトルを$\boldsymbol{N}$、拡散反射係数を k_d、そして光の強さは光源からの距離ιの 2 乗に反比例して減少することから、

$$I_d = k_d I_q(\boldsymbol{E} \cdot \boldsymbol{N})/\iota^2$$

を得る。しかし、相互反射を考慮しない環境においては明暗の変化が激しいため、

$$I_d = k_d I_q(\boldsymbol{E} \cdot \boldsymbol{N})/(\iota + c)$$

とすることが多い。ここで、c は適当に選ぶ定数である。

2.2.3 ハイライト

これまでは、拡散反射光だけを考慮してきた。そのため、なめらかな物体の表面や縁がキラリと光るような光沢をもつ画像は得られない。このような光沢の表現は、プラスチックや金属などの質感表現に重要である。このような光沢、すなわち鏡面反射成分によるハイライト (high-light) 効果を出すための陰影付けについて述べる。

このような輝きは、光源と物体表面の傾きとの関係だけではなく、その表面を見る方向に依存していることがわかる。すなわち、物体表面を光源と反対側から眺め、視線と表面の角度を考えるとよい。正反射方向から眺めたときが最も輝きが強いことがわかる。すなわち鏡面反射により光源が見えているということである。

まず、プラスチックの質感について示す。鏡面反射光による輝度 I_sは、図 4に示すように

- 入射光を I_p、
- 鏡面反射係数を k_s、
- 光線の正反射方向と視線のなす角をα、
- 光線の正反射方向の方向余弦を$\boldsymbol{L}' = (L'_x, L'_y, L'_z)$、
- 視線ベクトルを$\boldsymbol{E} = (E_x, E_y, E_z)$、
- $\boldsymbol{V} = -\boldsymbol{E}$

とすると、次式が得られる。

$$I_s = k_s I_p \cos^n \alpha = k_s I_p (\boldsymbol{L}' \cdot \boldsymbol{V})^2$$

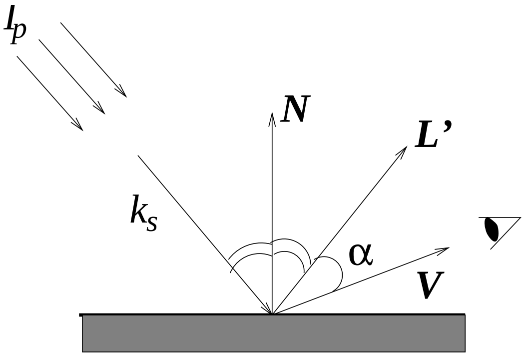


図 4: Phong のモデル

これは、Phong により提案されたもので Phong のモデルと呼ばれる。n が大きいほどハイライトの広がりは小さくシャープなものとなる。上の方向余弦$\boldsymbol{L}'$は、光源ベクトルを$\boldsymbol{L}$、表面の単位法線ベクトルを$\boldsymbol{N}$とすると、

$$\boldsymbol{L}' = 2(\boldsymbol{L} \cdot \boldsymbol{N})\boldsymbol{N} - \boldsymbol{L}$$

で求まる。このハイライトまで考慮した表示輝度は、拡散反射光による輝度との和 $I_d + I_{ad} + I_s$ となる。

また一般に、光線が物体表面にかすめるように突入するときには、反射がおこりやすいので、鏡面反射係数 k_sは入射角θに依存するが、通常は定数とおくことが多い。

次に金属の質感について示す。一般に金属ではそのハイライトの色は物体の色を反映するため光源とは異なる。すなわち、金属の鏡面反射係数は、入射角のみならず波長にも依存すると考えられる。一般に、金属の鈍い輝きと言われるのは、そのハイライトが物体の色を帯びるためである。

このような金属の質感表現にも向くように作られた陰影付けモデルに、Cook-Torrance モデルがある。このモデルでは、物体表面はランダムな方向をもつ微小な完全反射面の集合でモデル化さる。鏡面反射成分の分布 (Phong モデルでは $\cos^2 \alpha$) の形が、より実験値と合うように、入射角と視線ベクトルに依存する Beckmann 分布と呼ばれる分布モデルと入射角に依存する幾何減退係数によりモデル化さる。鏡面反射係数 (Phong のモデルでは k_s) は、入射角と波長により定まるという Fresnel の公式によりモデル化されている。

2.2.4 鏡面

図 5のように球の一つが鏡面であるとする。いま、あるピクセルを通過した視線がその球に当たったとすると、そのピクセルには鏡から視点方向に反射してくる光の色を与えればよい。それは、逆方向から考えれば、視線が鏡面で正反射し、その視線が当たる物体表面の色を与えればよいということと基本的に同じである。そのためには、

1. 鏡面と視線の交点 (最初の可視点) から正反射方向に向かう視線ベクトルを出し、
2. 再び可視点を決定するための交点計算を行い、
3. 新たな可視点の色をこれまでの陰影付けモデルにより決定し、
4. その色を鏡に映り込む色、すなわちピクセルの色とする

ことを行えばよい。

もし、鏡面で反射した視線が当たった物体もまた鏡面であったなら、その鏡面で反射する視線を、さらに追跡すればよい。

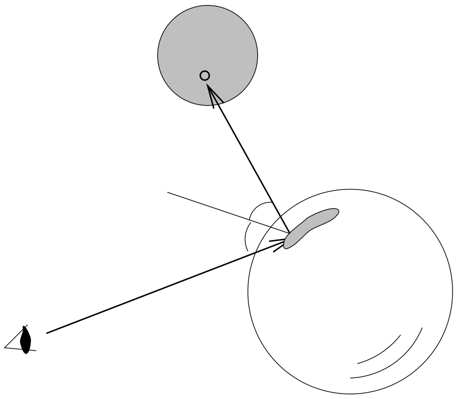


図 5: 鏡面をもった球と正反射

正反射した視線の視線ベクトル$\boldsymbol{R}$は、視線ベクトルを$\boldsymbol{E}$、鏡面上交点での単位法線ベクトルを$\boldsymbol{N}$とすると、

$$\boldsymbol{R} = 2(-\boldsymbol{E} \cdot \boldsymbol{N})\boldsymbol{N} + \boldsymbol{E}$$

で求まる。

2.2.5 視点の変更

任意の方向から物体を眺めるには、視点とスクリーンの位置を適当に定めればよい。ここでは、以下のように、視点とスクリーンの配置を定めることにする。

視線はスクリーンとその中心で直交しており、スクリーンのサイズが決まっているものとすれば、視点の座標(v_x, v_y, v_z)、視線の目標点の座標(o_x, o_y, o_z)、画角ϕが与えられれば、視線からスクリーンまでの距離が定まる。さらに、スクリーンの傾きを定めるためには、基準と回転角度γが指定されればよい。ここでは、図6に示すように、スクリーン中心を通る視線とスクリーンの交点で、視線と直交し$x-z$平面に平行な、すなわち視線に直交する水平な直線を考え、視線の方向を向き直交したときの右手方向の半直線を基準にとる。回転方向は視線の方向を向いた右ネジの回転方向にとる。

ただし、この方法ではスクリーン中心を通る視線が$x-z$平面に垂直な場合、すなわち、真上や真下を見た場合には基準となる直線が一意に定まらない。

これまで使ってきたxyz座標系をワールド座標系（世界座標系、world coordinate system）という。一方、スクリーンや視線ベクトルの表現には、図7に示すようにスクリーンに張り付けた座標系を使うほうが便利である。この座標系をスクリーン座標系（視平面座標系、view plane coordinate system）と呼ぶ。

これまでと同様に、レイトレーシングのための視線ベクトルはスクリーン座標系で容易に表現される。すなわち、視点の座標を$(0, 0, v_z)$、計算しようとしているピクセルの座標

を $(p_x, p_y, 0)$ とすると、視線ベクトル$\boldsymbol{E} = (E_x, E_y, E_z)$ は、

$$Ex = p_x/\iota, Ey = p_y/\iota, Ez = -v_x/\iota$$

$$\iota = \sqrt{p_x^2 + p_y^2 + v_z^2}$$

と表される。しかし、物体との交点計算を行うためには視線と物体を共通の座標系で表さなければならない。ここでは、視線をワールド座標系で読み直すことにする。そのためには、一般にスクリーン座標系で座標 (x', y', z') をもつ点 Pのワールド座標系での座標 (x, y, z) が求まればよい。そのためには、スクリーン座標系の各軸がワールド座標系の対応する各軸にぴったり重なるように移動する変換 Tが定まればよい。

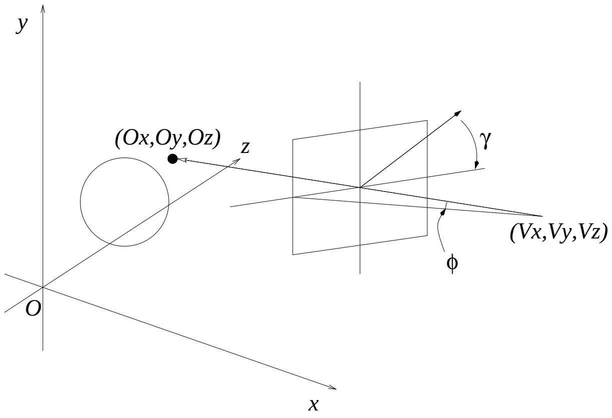


図 6: スクリーンの傾きの定義

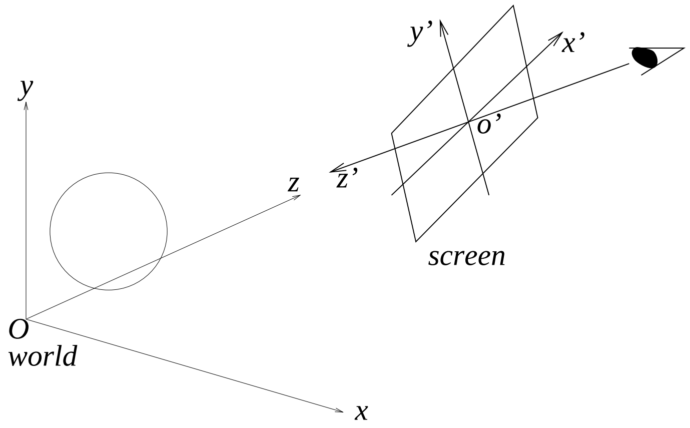


図 7: スクリーン座標系とワールド座標系

2.3 レイトレーシング法の高速化技法

視線追跡は生成された視線の数だけ呼び出される。視線は、ピクセルの数、反射光の数、透過光の数の和だけ生成される。ピクセル数は、スクリーンの解像度を $2S \times 2S$とすると$(2S)^2$であり、少なくともこれだけの視線は必ず生成される。したがって、$S = 500$ とすると、少なくとも $1000 \times 1000 =$100 万本の視線が生成される。反射や透過による視線は、それらの原因となる鏡面物体や透明物体のスクリーンへの投影像に含まれるピクセルの数だけ少なくとも生成される。さらに、それらの視線が鏡面物体や透明物体に当たれば、それだけ生成される視線の数は増える。さらに、影計算を行うと、拡散反射成分をもつ物体と交点をもった場合には、光源の数だけの視線を出して視線追跡と同様の交点計算が要求される。この影計算も、少なくともピクセルの数だけ行われる必要がある。

また、それぞれの視線追跡では、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。

以上より、2 つの方法が考えられる。

1. 光線の数を減らす方法
   - Z バッファ兼用法 :物体と交差しそうな 1 次光線に対するピクセルを Z バッファ方式で求めておいて、その 1 次光線のみをレイトレーシングする。
   - 画素選択型光線追跡法 :まず、適当なピクセル間隔で 1 次光線を生成してレイトレーシングし、物体と交差した光線に対応するピクセルの周りの部分にあるピクセルのみ再レイトレーシングする。
2. 交差判定に対象となる物体の数を減らす方法
   - 物体空間分割法 :物体空間を分割し、ボクセルで物体を管理する方法。その空間内に属する物体のみを登録し、交差判定は光線がどの空間を通るかを探索し、該当した空間にある物体のみ行う。物体の分割方式により、オクトリー分割 [28]、空間等分割法 [21] がある。
   - 物体分割法 :物体を分割し、ある外接体 (バウンディングボリューム) で取り囲む。ある光線がその外接体と交差するかを調べ、交差するなら実際の物体との交差判定を行う。外接体との交差がある限り木を葉の方向にたどり判定を進める。

3 マルチトランスピュータシステム

3.1 アーキテクチャ

トランスピュータは、イギリスのINMOS社が開発した通信機能を持ったマイクロプロセッサである。T805は32ビットCMOSのマイクロプロセッサであり、4KBの内部RAM、1MBの外部RAM、メモリインターフェイス、他のトランスピュータと通信するための4本のリンクを1チップ上に持っている。クロックは25MHzで25MIPSの性能を持っている。リンクを通した通信速度は標準で10Mbpsであり、5Mbpsや20Mbpsにすることもできる。リンクは入力と出力方向の信号線2本で1組となっている。各リンクで双方向通信を行った場合、最大転送速度は2.35Mbytes/sである。マルチトランスピュータシステムのブロック構成図を図8に示す。

トランスピュータでは、1台のプロセッサ上ですべての並列プロセスを実行することができ、並列プロセスを複数のトランスピュータに分割することもできる。複数のトランスピュータで並列プロセスを実行する場合は、各トランスピュータで同時に処理が行われるが、1台のトランスピュータで複数の並列プロセスを実行する場合は時分割処理となる。トランスピュータのプロセスの優先順位には、高順位と低順位があり、プログラム中でどのプロセスがどちらの優先順位を持つか指定できる[22]。

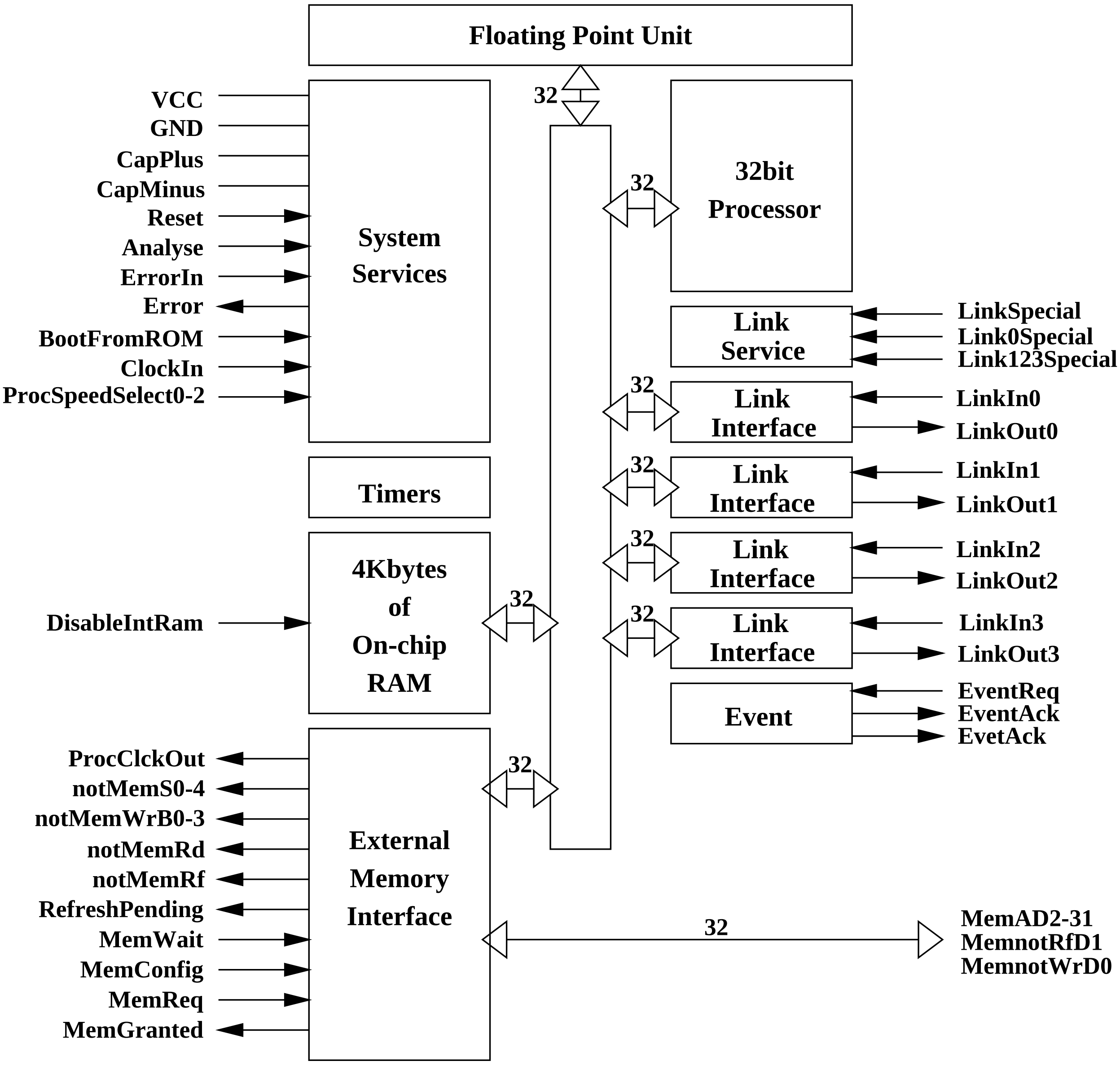
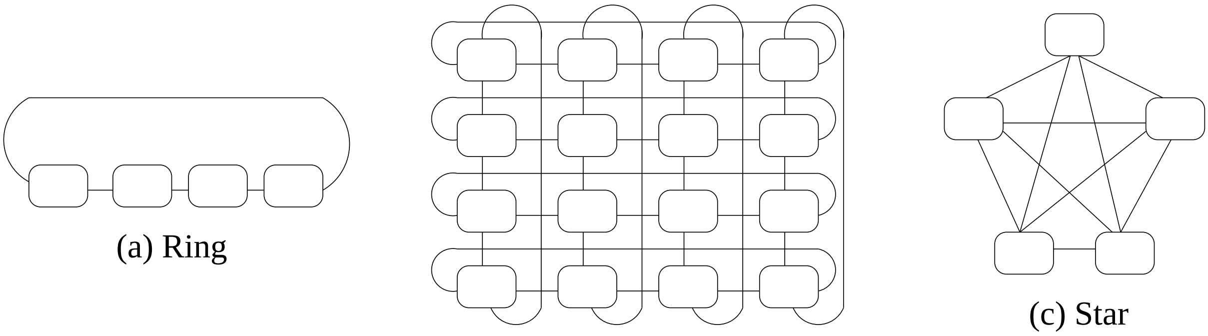


図 8: トランスピュータ T805 のブロック構成図

3.2 相互結合網

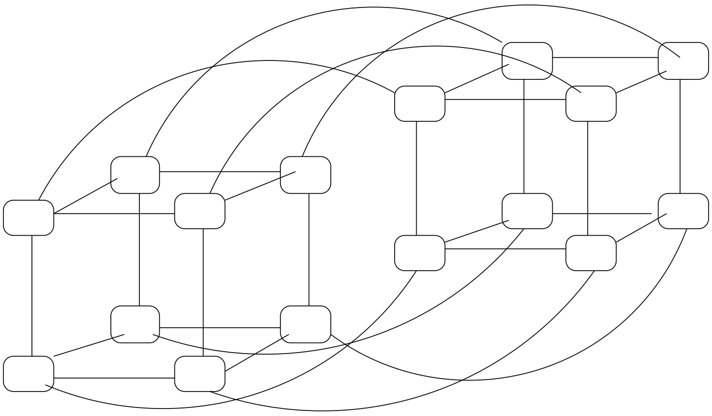
マルチプロセッサでは、プロセッサ間の接続方式が非常に重要となる。接続ネットワークをどのように設定するかは、トータルパフォーマンスを左右する。接続網の種類はリング、トーラス、スター、ハイパーキューブなど様々である（図9）。現在、研究室にある並列計算機トランスピュータの可能な接続網は、64台によるトーラス接続とハイパーキューブ接続である。トーラス接続は図9に示すようにリング接続を発展させたものである。非常に扱いやすい接続であり、本研究ではこの接続を利用している。ハイパーキューブ接続とは、配列の次元を増やして3次元の立方体、さらに多次元の超立方体にプロセッサを接続して、プロセッサ間の平均距離を短縮させたものである。ハイパーキューブ接続は、リング接続、木接続、格子接続などを含み、汎用性が高い。



(a) Ring

(c) Star

(b) Torus



(d) Hypercube

図9: 相互結合網

3.3 Parallel C

Parallel C は英国 3L 社により開発され、通常の C 言語に並列化と通信機能を加えたものである。並列化や通信は関数コールで行われる。Parallel C による並列プログラムは基本的にハードウェアの接続を意識して作る必要がある。プログラムを並列化する単位はタスクとスレッドであり、タスクは静的なプロセスで、あらかじめどのプロセッサで実行されるか決められている。これに対してスレッドは動的なプロセスで、プログラムの実行中にタスクの中で起動される。タスクは 1 つのプログラム形式をとり、スレッドは関数の形式をとっている。

通信はチャネル通信によるものとセマフォを使った共有変数による通信とがある。チャネル通信はタスク間の通信に使われ、共有変数による通信は同一タスク内のスレッド間の通信に使われる。同一タスク内のスレッド間ではチャネル通信を行うことはできない。しかし、各タスクが同じ処理を行う場合は Flood-fill と呼ばれる機能を使うことができる。これは、処理を分割して配る master タスク、各プロセッサで処理をする worker タスク、master と worker タスクの属性について記述したコンフィギュレーションファイルを用意すれば、どのようなハードウェア構成でも動作させせることができる。また、master タスクと各 worker タスクとの間のメッセージルーティングはシステムが行ってくれる。この場合の処理の流れは master タスクが仕事をネットワーク上に出し、手の空いている worker タスクがそれを処理して master タスクに返すことを繰り返す。

3.4 タスクとマルチスレッド

各プロセッサは、複数のタスクをもつことができる。さらに、Parallel C ではマルチスレッドが可能であり、各タスクは複数のスレッドを同時に走らせることができる。複数の入力チャネルを同時に監視し、あるチャネルからメッセージが届き次第、それに応じた処理を開始したいとき、非常に有効な手段となる。プロセッサの2本のリンクを用いマルチスレッドを使用した場合を図10に示す。1プロセッサ上にCom TaskとMain Taskの2つのタスクがあり、Com Taskには6つのスレッドがあることを示している。各スレッドは入力(1,4,5)、または出力(2,3,6)であり、入力スレッド(1)から送られた情報はCom Taskで自分の必要とする情報であるか調べる。自分の必要とする情報であれば、出力スレッド(3)からMain Taskへ送られる。そうでなければ、出力スレッド(6)から次のプロセッサへと送られる。この方法を用いることにより、Main Taskが計算中であってもCom Taskが稼働することで並列効果をあげることができる。

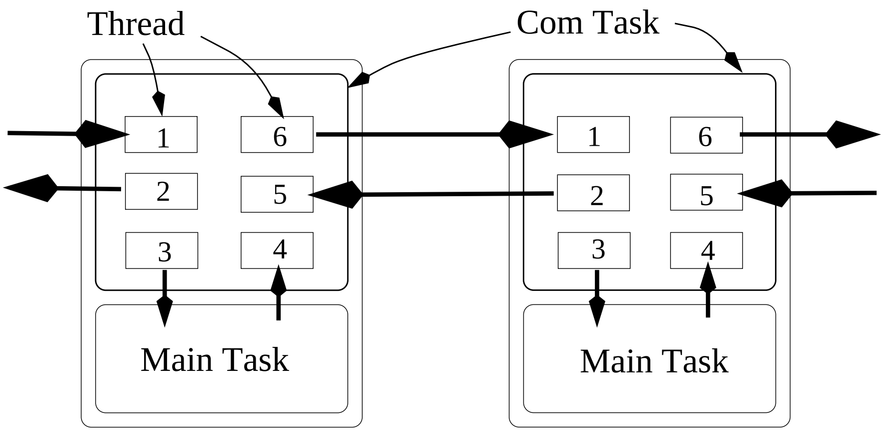


図10: タスクとスレッド

4 画面·空間分割による高速化

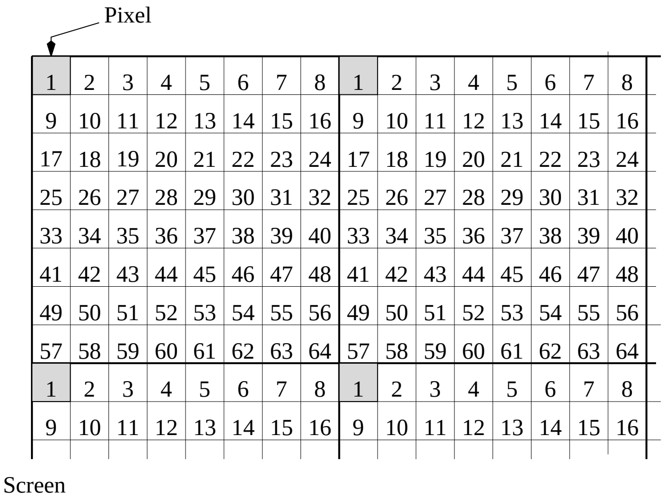
4.1 画面分割

レイトレーシングでは、画面上の各ピクセルを独立に計算することができる。それを利用してスクリーンを分割し、複数プロセッサで並列処理をすることで高速化を図る。従来、画面分割では、等分割（縦分割、横分割、メッシュ分割）、負荷均衡分割（縦分割、横分割、メッシュ分割）、そしてインターリーブ分割を行ってきた。物体記述データから、物体の3次元空間内の位置やパラメータを知ることはできるが、実際にそれが画面上に描かれる領域は、物体の回転、移動、視点の位置などに依存するため、知ることはできない。画面分割では負荷の予測が最も重要となる。できるだけ負荷均衡を考慮した方法として、負荷均衡分割、インターリーブ分割が挙げられる。

本研究で用いた空間分割が、どれくらいの効果があげられるかを比較するため画面分割を取り上げる。画面分割の中で、トランスピュータ64台使用時、相対速度60 ~ 64倍と最も良い性能を出したインターリーブ分割との比較を行う。

4.1.1 インターリーブ分割

あるピクセルの周りのピクセルも、ほぼ同じ負荷であると考えられる。図11に示すように、トランスピュータ64台使用時には、8×8のピクセルに対して、64台のプロセッサを割り当てる。トランスピュータ1は斜線部分の計算を担当する。



1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64	57	58	59	60	61	62	63	64
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	9	10	11	12	13	14	15	16

図11: インターリーブ分割

4.1.2 インターリーブ分割による並列化

インターリーブ分割によって割り当てられたピクセルを、各トランスピュータで計算させ並列化を行う。接続網はリング接続を使用する。図12（右）に示すように、各プロセッサには通信部と計算部の2つのスレッドがあり、通信と計算が滞りなく行われている。通信部は、隣のプロセッサから入力されたデータを通過させる。計算部は、レイトレーシングの本体であり、与えられたピクセルの計算を行う。リング接続は図12（左）に示すような構成になる。データがプロセッサからプロセッサへ移動する通信距離を1とすると、最も離れたプロセッサからデータがルートに戻るときの最大通信距離は、使用するプロセッサ数になる。各プロセッサは、ローカルメモリとして1MBをもっている。各プロセッサとも同じ数のピクセルの処理を行うため、1プロセッサ当たり$(640 \times 400)/64 = 4000$ピクセルを受けもつことになる。ピクセルはRGB（各1バイト）の値とトランスピュータが32bitであることから、各プロセッサのデータ領域は$4000 \times 4 = 16$KB必要となる。

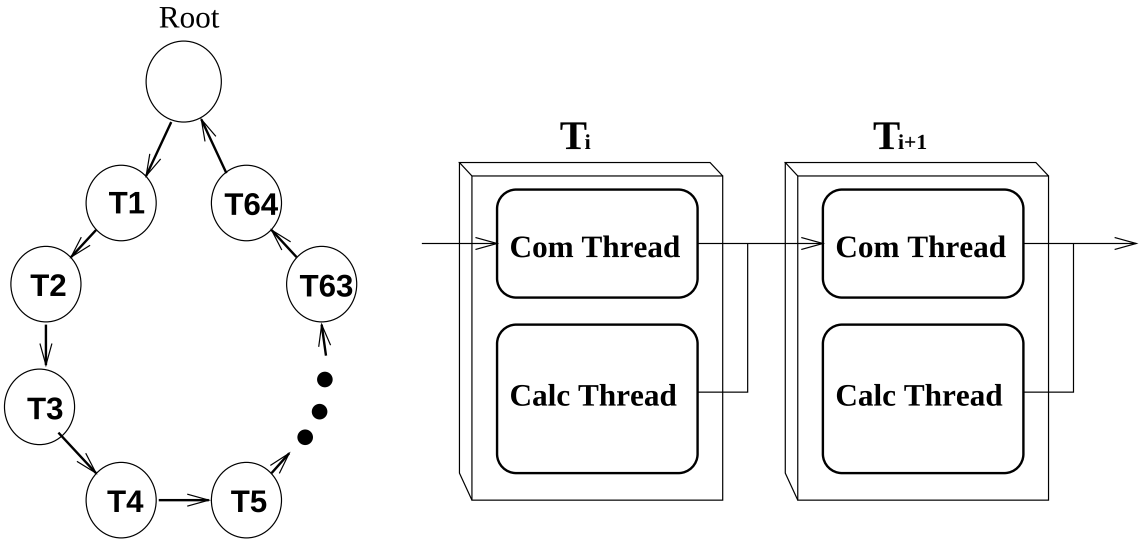


図12: リング接続（左）と各プロセッサ内のスレッドの様子（右）

4.2 等空間分割

空間分割 (Space Subdivision) とは、物体が存在する空間を決め、それ以外の空間は考えないものとする。等分割ではボクセル (Voxel) とよばれる一定の大きさに分割しその各空間に属する物体をリストとして得る (図 13)。次に、実際にレイトレーシングを行う。基本的なレイトレーシングでは、視線ごとに全物体と交差するかを計算してきた。例えば、物体が 100 個あるとき、その視線と明らかに交差しなくても、100 個の物体すべてと交差判定を行ってきた。しかし、空間分割では視線があるボクセルに入ると、そのボクセルに属する物体に対してのみ交差判定を行う。

ボクセルに物体が属していれば、その物体に対して交差判定を行い、もし交点が存在すれば、その視線に対する追跡は終了する。交点が存在しないか、あるいはその空間に物体が 1 つも属していないときは、その視線が通る次のボクセルに視線を移し、再び交差判定を行う。反射のときも、同様にボクセル間を移動し判定を行う。視線が全体の空間を出た場合は、その視線の追跡は終了となる。

図 14は視点から出た光線の物体との交差回数を、通常のレイトレーシング法と等空間分割によるものと比較している。通常のレイトレーシング法では、どの光線に対しても全物体数 4 の交差判定が行われる。一方、等空間分割で最も左の光線を見ると 2 つのボクセルを通過するが、どちらのボクセルにも物体は登録されていない。よって、交差判定も 0 回となっている。図の 5 本の光線の交差判定を合計すると、通常のときが 20 回、等空間分割のときで 9 回と半分以下の回数となっている。さらに物体数が増え各ボクセルに均等に物体を配置したとき、1/(ボクセル数) の交差回数となる。この方法により各光線の交差判定を減少させることで高速化が図れる。

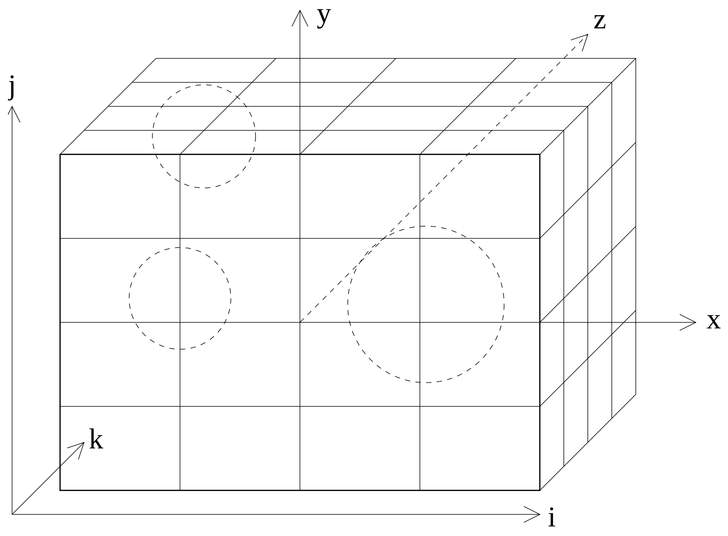


図 13: 等空間分割

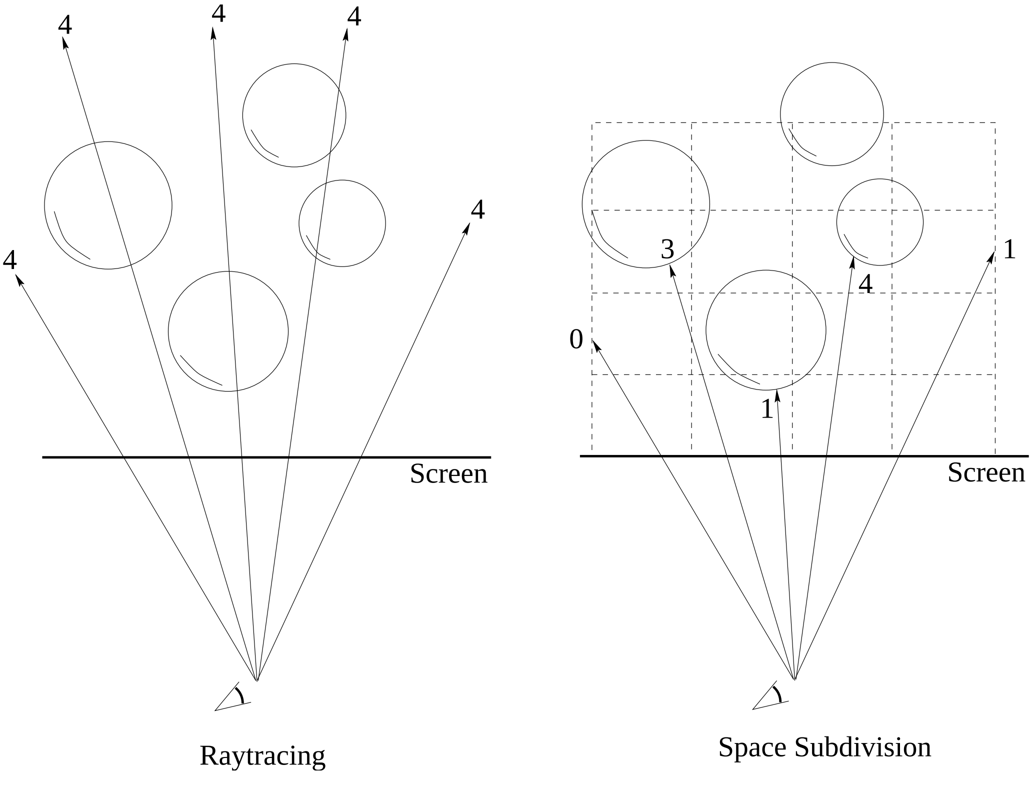


図 14: 交差判定の回数

視線が 2 次元上で各ボクセル間を移動する様子を図 15に示す。図では、鏡面物体 A と交差し、その反射方向に視線を飛ばし、物体 B と交差すると陰影付けのため光源へ視線を飛ばしている。その時の視線は、ボクセル番号により表すと、

$$(120) \rightarrow (121) \rightarrow (221) \rightarrow (222) \rightarrow (223) \rightarrow (123)$$

と移動していることが分かる。

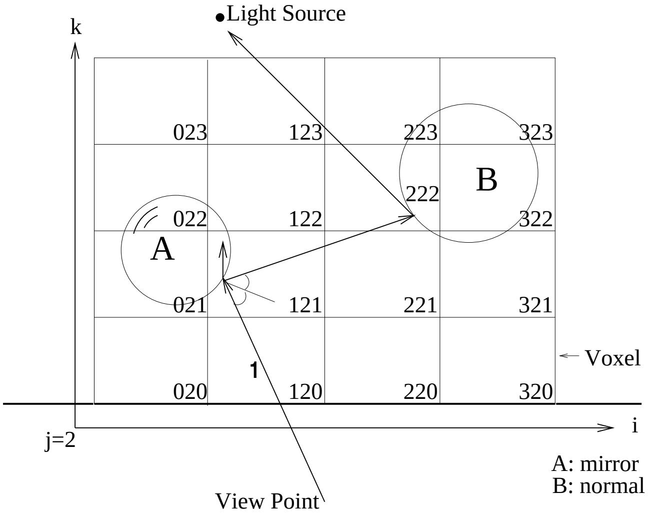


図 15: 光線の移動

4.2.1 2DDDA と 3DDDA

空間分割ではボクセルで空間を管理しているため、光線がどのボクセルを通過するかを知る必要がある。等分割では 3DDDA を用いる。DDA(Digital Differential Analyzer:デジタル微分解析法) とは、CRT のディスプレイの解像度による線分の断裂を防ぐための線分発生アルゴリズムである。2 次元の場合を 2DDDA、それを 3 次元に拡張したものを 3DDDA と呼ぶ。

図 16では、2DDDA により示す。3DDDA は z軸方向に拡張したものである。まず図 16に示す各変数を以下のように決める。Vx,Vy,Vz は 1 つのボクセルの各辺の長さを表している。

- (i, j, k):現在のボクセルの x, y, z方向のインデックス。
- (dx, dy, dz):直線が x, y, z方向それぞれについて、1 ボクセル分進むときの直線上の距離。

- (tx, ty, tz):現在のそれぞれ x, y, z 方向のインデックスが最大に有効な地点までの直線上の距離。

- (di, dj, dk):dx, dy, dz だけ進むときのインデックスの増分、すなわち $+1, 0, 1$ のいずれか。

点 1 で光線が $(i, j) = (0, 0)$ のボクセルに入ったとする。その光線が次にどのボクセルを通るかを決めるために、tx と ty の値を比べる。図 16 のように、$tx < ty$ なので、$(i+1, j) = (1, 0)$ を通ることが分かる。次に、点 2 において新たに tx と ty を比較し、$tx_{NEW} < ty_{NEW}$ になるので、$(i, j+1) = (1, 1)$ を通ることになる。このように、tx と ty の比較により、インデックス値の増減を決定でき、どのボクセルを通るかを知ることができる。この例では、$(0, 0) \rightarrow (1, 0) \rightarrow (1, 1) \rightarrow (2, 1)$ を通る。

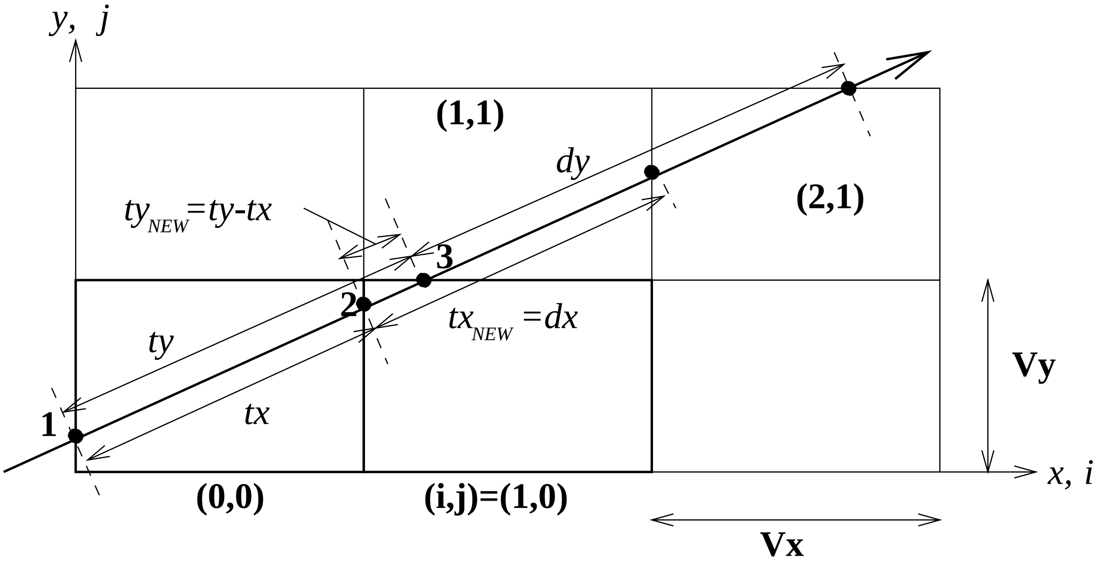


図 16: 2DDDA を用いた光線の移動

4.3 適応型空間分割

等空間分割では、一定の大きさのボクセルを扱った。そのため、全体の交差判定は減ったものの、各ボクセルの物体数を比較すると、まだまだばらつきがある。例えば、物体数0のボクセルも出てくることにより、平均の値(全物体数/ボクセル数)を大きく上回る値も出てくる。適応型分割では、あらかじめ物体の分布を調べ、物体数の多い所はより細かく分割する。その方法により、物体数を均等にするようボクセルサイズを決定する(図17)。その結果、どのボクセルに視線が入っても、ほぼ同じ回数の交差判定が行われることになる。

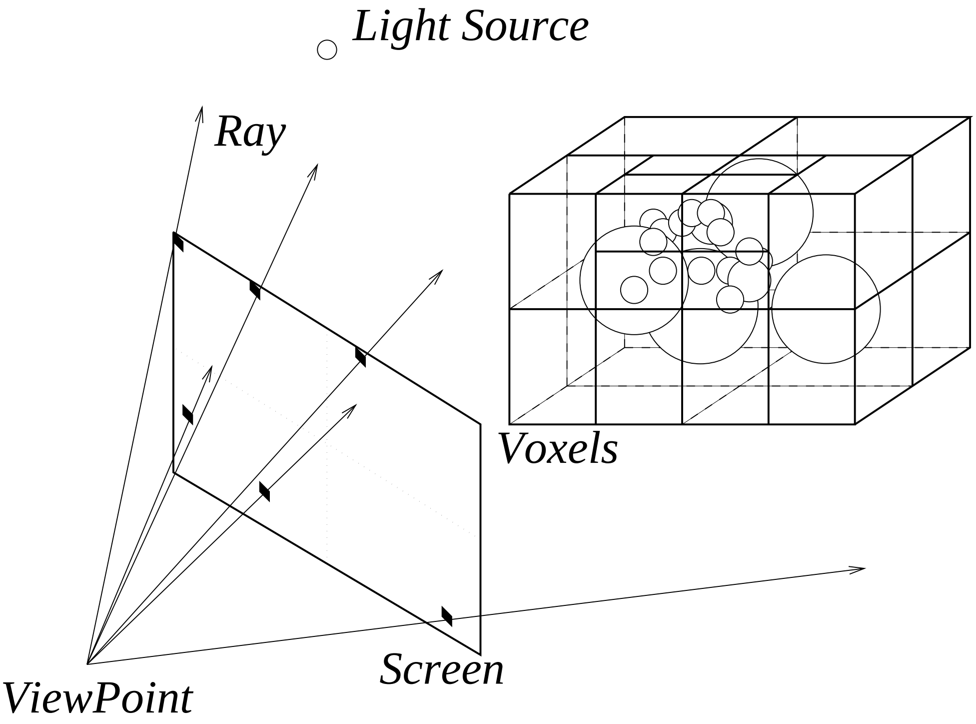


図17: 適応型空間分割

4.3.1 ボクセル決定

適応型分割の各ボクセルは、等分割のように単純に決定できないため、サブボクセル(SubVoxel)を用いて決定する(図18)。サブボクセルとは、物体の存在する空間を16×16×16に分割した一定の大きさの空間である。まず、それらのサブボクセルに対して属する物体を登録する。次に、そのサブボクセルに登録された物体数を用いて空間を分割していく。図19に示すように、物体の存在する空間を2つに分割する。2つに分割された空間のうち物体数の多い空間をさらに分割する。これをボクセル数が64個になるまで繰り返すことで、ボクセルを決定する。そのとき、物体数が0となる空間は数にいれず、新たに空間を分割

することにする。

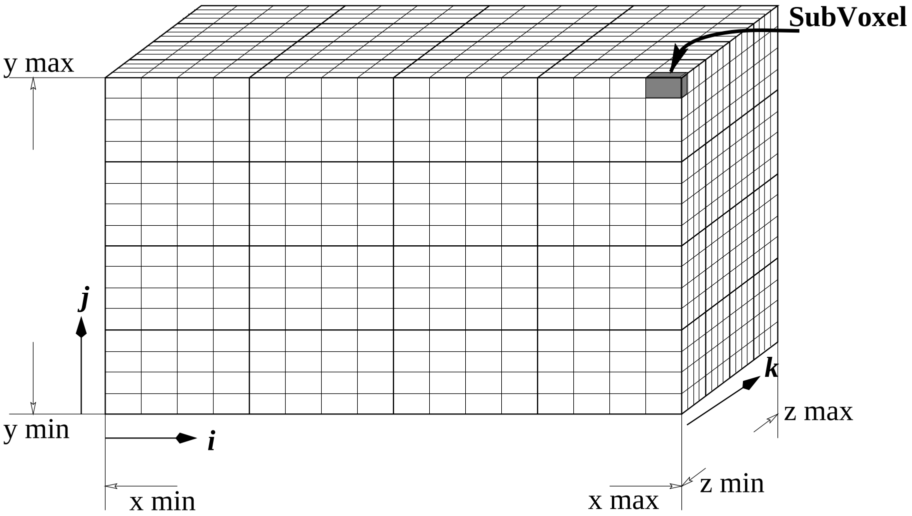


図 18: サブボクセルへの分割

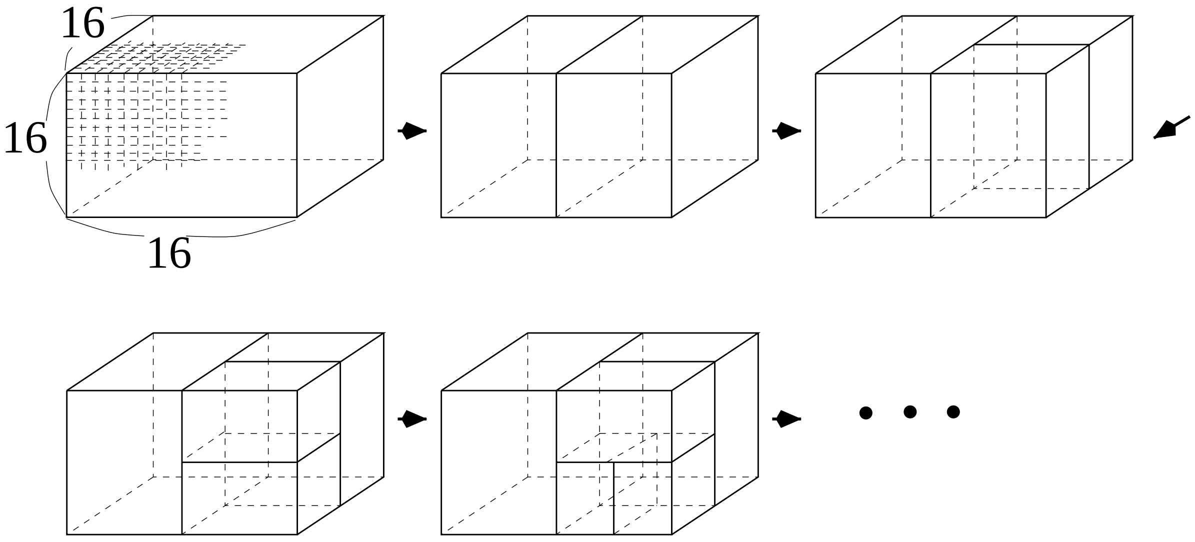


図 19: ボクセルの設定

4.3.2　ボクセル間移動アルゴリズム

適応型空間分割では、視線のボクセル間の移動として 3DDDA を用いる。ただし、ボクセル単位では各ボクセルの大きさが異なるため、サブボクセル単位で 3DDDA を適用する。

そのとき、(i,j,k) はそれぞれ $0 \sim 15$ の値となる。また、各サブボクセルには対応するボクセル番号が登録される。図 20に示すように、視線が $(0,0)$ に入ったとき、サブボクセルに対して 3DDDA の計算を行う。次のサブボクセルは $(1,0)$ となるが、ボクセル番号が変化しないためさらに 3DDDA 計算を行う。これを繰り返し、ボクセル番号が変わった時点で、はじめて次のボクセルを決定することができる。図 20に示すように、サブボクセルのボクセル番号の変化から、1,3,5,8,7 へと視線が通過していることが分かる。

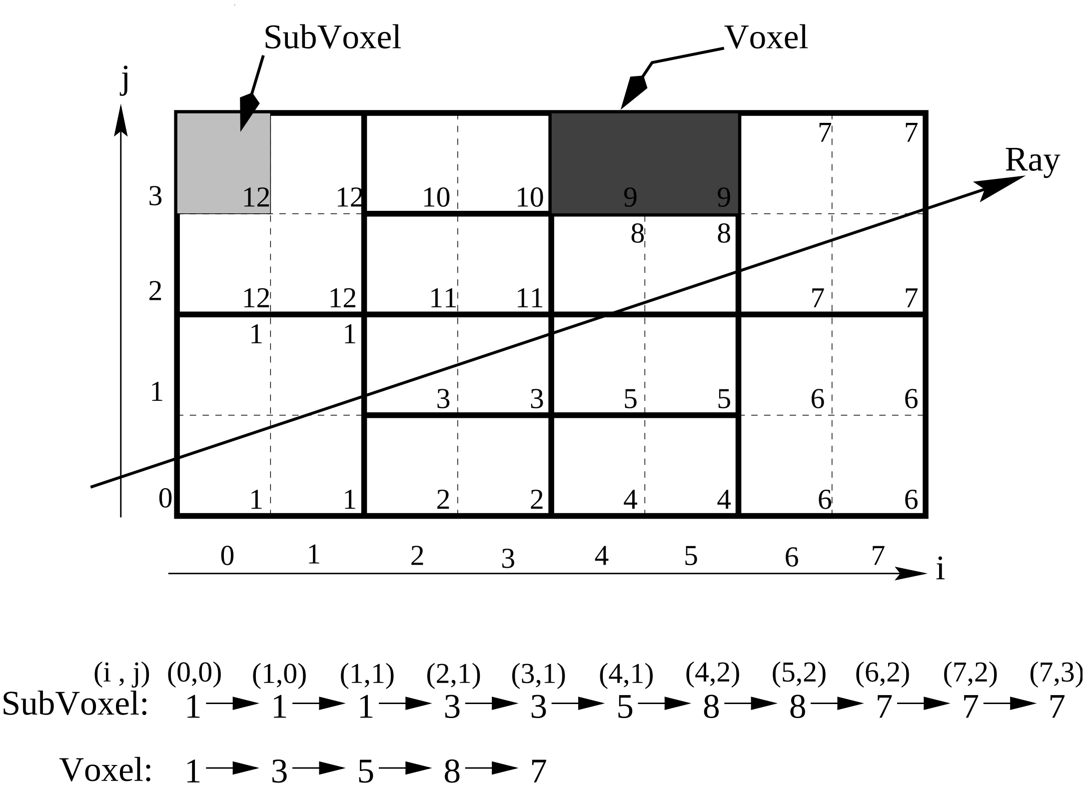


図 20: 3DDDA を用いた光線の移動

4.4　マルチトランスピュータによる空間分割

等分割、適応型分割では、決定された各ボクセルをトーラス接続されたプロセッサに割り当てる (図 21)。ボクセルはランダムにプロセッサに割り当てられる。プロセッサ間で光線情報パケットをやり取りすることでレイトレーシング計算を行う。

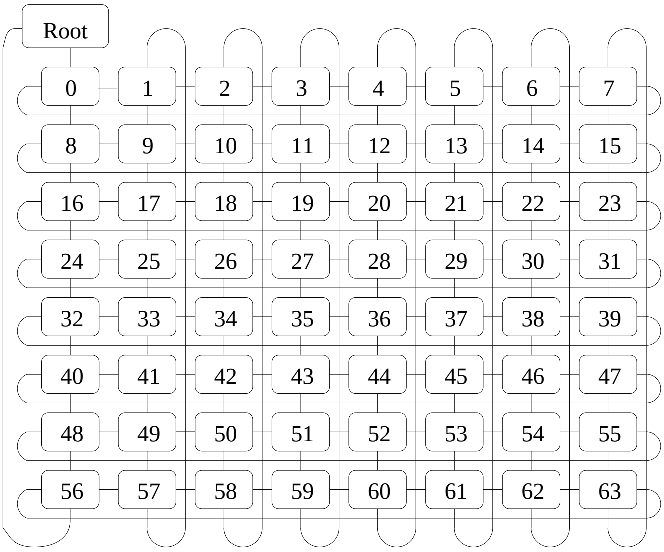


図 21: トーラス接続上での Voxel 番号

4.4.1　レイパケットとリターンパケット

レイパケット (Ray Packet) とは、その光線に関するデータを含んだパケットである。このパケットがボクセルに入ったときに、参照され計算が行われる。またリターンパケット (Return Packet) は、レイパケットによって計算された結果データを含んだパケットである。レイパケットは、交差判定を行い影計算のためボクセルを通過する。そのとき、ある物体と交差するか、または全空間を出ていくときルートにリターンパケットを返す。

リターンパケットは、鏡面物体と交差したとき光源の方向と正反射方向に光線を飛ばす。そのため、鏡面物体による反射が繰り返すと、交差回数も増え光線がいくつ返ってくるか予想がつかない。そこで、図 22に示すように、はじめの光線の重みを 1 とする。鏡面と交差したとき、光源の方向と正反射方向に飛ぶ光線のそれぞれの重みを半分 1/2 の値にする。光源の方向に飛んだ光線は、そのまま影計算をした後、リターンパケットとして重み 1/2 を持ちルートに戻ってくる。正反射方向に飛んだ光線が、次の鏡面物体と交差したとき、さらに重みを半分の 1/4 とする。その光源に向かった光線もリターンパケットとして重み 1/4 をルートに戻す。正反射方向に飛んだ光線が、それ以上交差しなかったとき、そのままの重み 1/4 をルートに戻す。それぞれルートに戻ってきたリターンパケットの重みを合

計すると$1/2+1/4+1/4=1$となる。ルートではそのピクセルに対するリターンパケットの重みを合計し1となることですべてのリターンパケットが戻ってきたことを知る。

表1にレイパケット、表2にリターンパケットを示す。

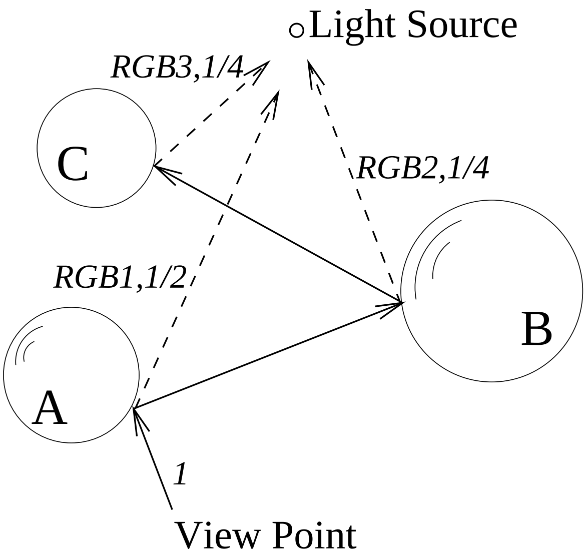


図 22: レイパケットの移動

表 1: レイパケットの定義

変数名	バイト数 (データ型)	内容
$Destination$	$4(int)$	行先ボクセル番号 (i, j, k)
$pixel\_xy$	$4(int)$	光線が発生したスクリーン上での座標 (p_x, p_y)
$kinds$	$1(int)$	パケットの種類
$weight$	$4(float)$	パケットの重み
dx, dy, dz tx, ty, tz di, dj, dk	$4(float) \times 9$	次のボクセル検索用データ
Ex, Ey, Ez	$4(float) \times 3$	光線の方向ベクトル
$Kssr, Kssg, Kssb$	$4(float) \times 3$	計算された交点の輝度
att	$4(int)$	交差した物体の色データ
nx, ny, nz	$4(float) \times 3$	交点の法線ベクトル
x, y, z	$4(float) \times 3$	交点
$level, inout$	$4(int) \times 2$	反射回数、透過回数

表 2: リターンパケットの定義

変数名	バイト数 (データ型)	内容
$pixel\_xy$	$4(int)$	光線が発生したスクリーン上での座標 (p_x, p_y)
$weight$	$4(float)$	パケットの重み
$red, green, blue$	$4(float) \times 3$	計算された交点の輝度

4.4.2　各プロセッサのタスクとスレッド

図23に示すように、各プロセッサは計算タスク (Main Task) と通信タスク (Com Task) からなり、さらに通信タスクには、各ポートからの入出力スレッド (in,out) が10個存在する。計算タスクではレイトレーシング計算が行われる。通信タスクでは、いずれかの in スレッドから来たデータが自分の受持ちであれば、計算タスクへ送る。そうでなければ、目的プロセッサ番号へ最も短い経路になるための out スレッドから送り出す。

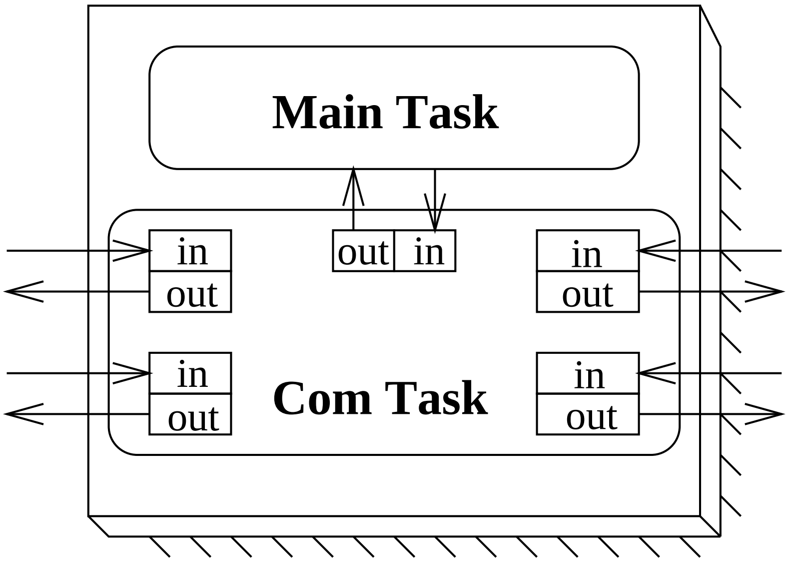


図 23: タスクとスレッド

5　実験と評価

5.1　実験方法

等空間分割について、64 プロセッサを用いて実行時間を測定する。そして、画面インターリーブ分割の 64 プロセッサ使用時と実行時間を比較する。これらの実験では、物体数 64 ~ 8192 について実験を行う。画面分割ではリング接続、空間分割ではトーラス接続を用いる。実験のデータは、

- すべて通常物体（均等配置、ランダム配置）
  物体数 64,128,256,512,1024,2048,4096,8192
- すべて鏡面物体（均等配置、ランダム配置）
  物体数 64,128,256,512,1024,2048,4096,8192

を用いる。ここで、通常物体とは物体表面が反射しないもの、鏡面物体とは鏡のように反射し映り込みするものである。また、均等配置は各ボクセルに同数の物体を配置した状態で、ランダムなものがランダム配置である。なお、この実験で用いた物体は球のみである。

次に、空間分割において等分割と適応型分割による実験を行う。ともにトーラス接続を用いる。実験では一般にレイトレーシング法の研究で使用される SPD(Standard Procedural Databases) から表 3に示す 6 つの物体についてすべて通常物体、すべて鏡面物体を用いる。データ 1 ~ 5 については三角形、データ 6 では球を用いている。

表 3: 空間分割で用いた SPD 画像

データ		物体数	物体形状
1	Jack	312	三角形
2	Skull	1243	三角形
3	Teapot	2329	三角形
4	Tetra	4096	三角形
5	Mount	8192	三角形
6	Balls	14762	球

5.2 実験結果と考察

5.2.1 等空間分割と画面インターリーブ分割との比較

図24と図25は、画面インターリーブ分割(Screen)と等空間分割(Space)による実験結果であり、通常物体を用いたものと鏡面物体を用いたものである。横軸は物体数を、縦軸は実行時間を表している。画面分割では、全物体を各プロセッサに登録するため最大で256物体までの結果となっている。

いずれの結果においても、空間分割の実行時間は画面分割と比べて、ゆるやかに上昇している。また、空間分割ではランダム配置の方が均等配置よりも実行時間が長い。これは、ランダム配置では物体が2つ以上のボクセルにまたがり、各ボクセルの物体数を増やし、交差判定の回数を増やしているためである。通常物体と鏡面物体を比較すると、鏡面物体では光線が当たるとさらに反射方向に光線を飛ばし、その光線についても交差判定を計算するため、計算時間が増えることになる。図から、鏡面物体の実行時間は通常物体の約2倍となっている。図24の空間分割・均等配置を見ると、64~256物体ではほとんど変化が見られない。これは、均等配置では各ボクセルの物体数は等しく256物体のときでボクセル当たり4物体となり、交差判定に時間はかからない。このことから、3DDDAによる次のボクセルを決定する処理時間、及びレイパケットが各プロセッサ間を行き来する通信によるオーバヘッドが、全実行時間の大部分を占めていると考えられる。

図24からは、128個より多い物体を扱うとき、空間分割の結果が良くなっていることが分かる。図25では、256個より多い物体を扱うとき、空間分割の結果が良くなっている。また、64物体のランダム配置、256物体の均等配置、及び8192物体のランダム配置の結果が大きくずれている。64、256物体では、用いた画像が大きい物体を扱ったため、ほとんどの光線が手前のボクセルで交差し、それ以降にあるボクセルに進んだ光線が少く実行時間が短くなったと考えられる。8192物体では、ボクセルに登録できる物体数が最大600個であり、それ以上の物体は登録されていない。そのため、全物体数が減り実行速度が短くなったと考えられる。

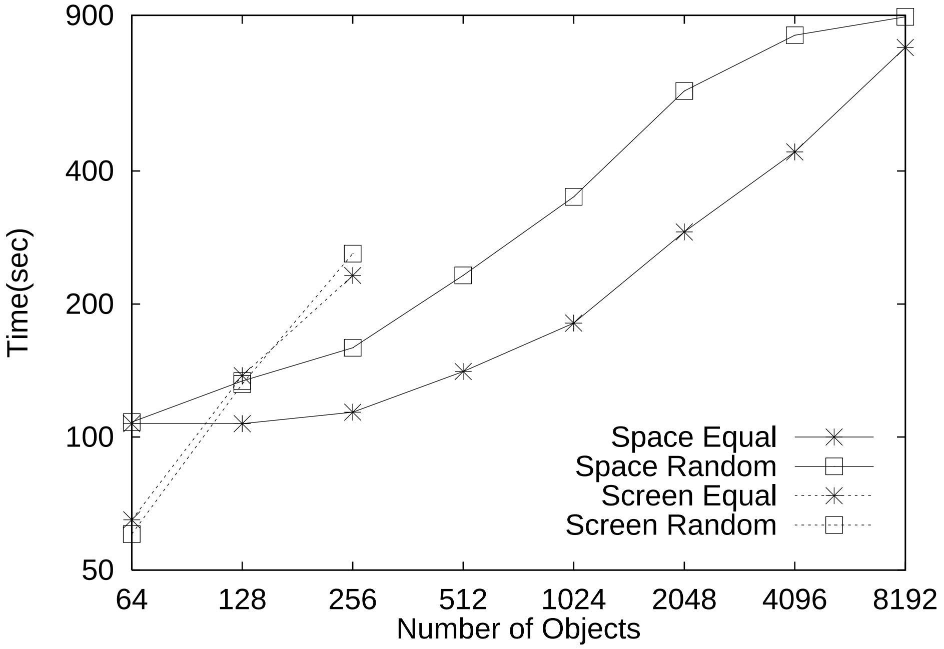


図 24: 画面分割と等空間分割の結果 (通常物体)

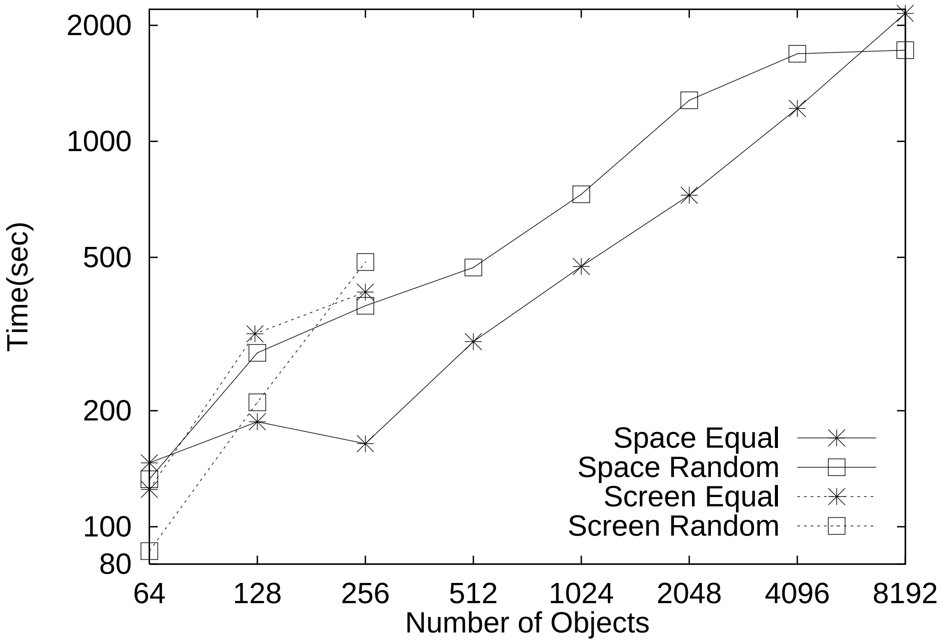


図 25: 画面分割と等空間分割の結果 (鏡面物体)

5.2.2　等空間分割と適応型空間分割による実行結果

表 4は、空間分割のとき分割されたボクセルに割り当てられた物体数、その平均物体数、及び最大物体数について示している。

表 4から、等分割、適応型分割のどちらの場合を見ても、ボクセルに割り当てられる物体数が本来の物体数よりも増えていることが分かる。特に適応型分割の方が増えている。適応型分割では、より細かくボクセルを決定するため、物体がボクセル間をまたがる率が高くなっているためである。平均物体数は、分割後の物体数を 64 で割った値である。適応型分割により、最大物体数との差が小さくなり、それだけ均等化されていることになる。

図 26 ~ 図 31には、物体登録後の各プロセッサの物体数を示す。各画像とも等分割では物体数が大きかったものが、適応型分割により均等化されていることが分かる。特に Skull(図 27) と Teapot(図 28) は、最も均等化されている。等分割で局所的に物体数の多いボクセルが存在するとき、適応型分割でより均等化されることが分かる。

表 4: 空間分割後の物体数

データ		物体数	分割後の物体数		平均物体数		最大物体数	
			等分割	適応型分割	等分割	適応型分割	等分割	適応型分割
1	Jack	312	876	1188	13.7	18.6	55	31
2	Skull	1243	1869	2145	28.3	33.5	135	52
3	Teapot	2329	3267	3627	51.0	56.7	312	92
4	Tetra	4096	5356	6048	83.7	94.5	260	160
5	Mount	8192	10365	10745	162.0	167.9	549	280
6	Balls	14762	15148	15765	236.7	246.3	600	447

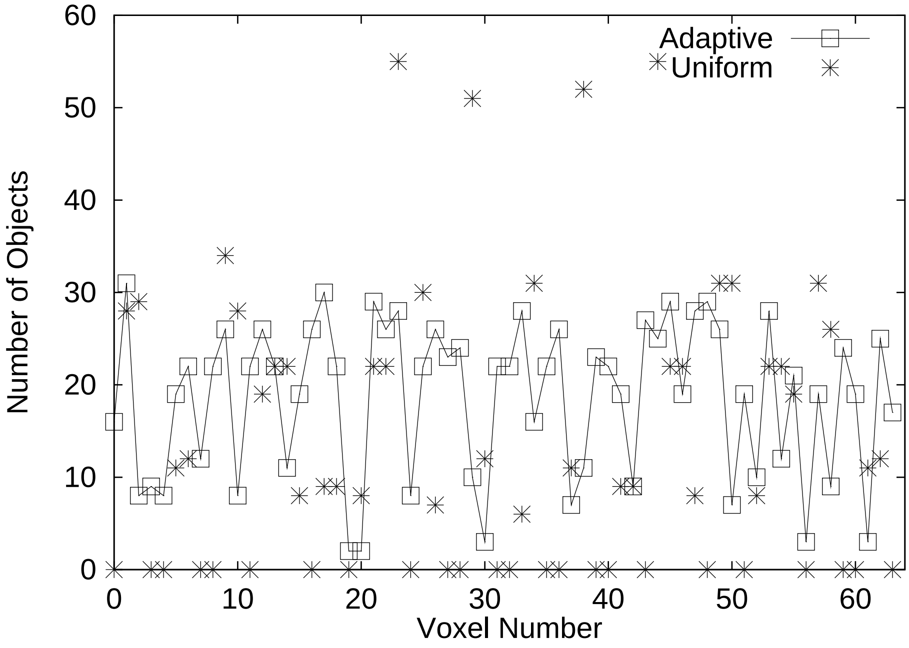


図 26: Jack の各ボクセルの物体数

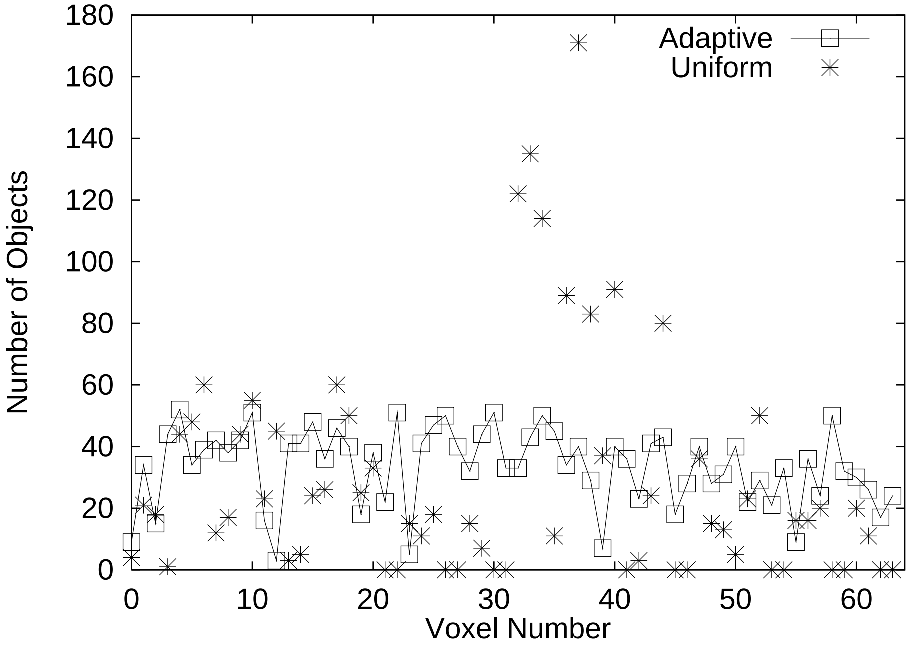


図 27: Skull の各ボクセルの物体数

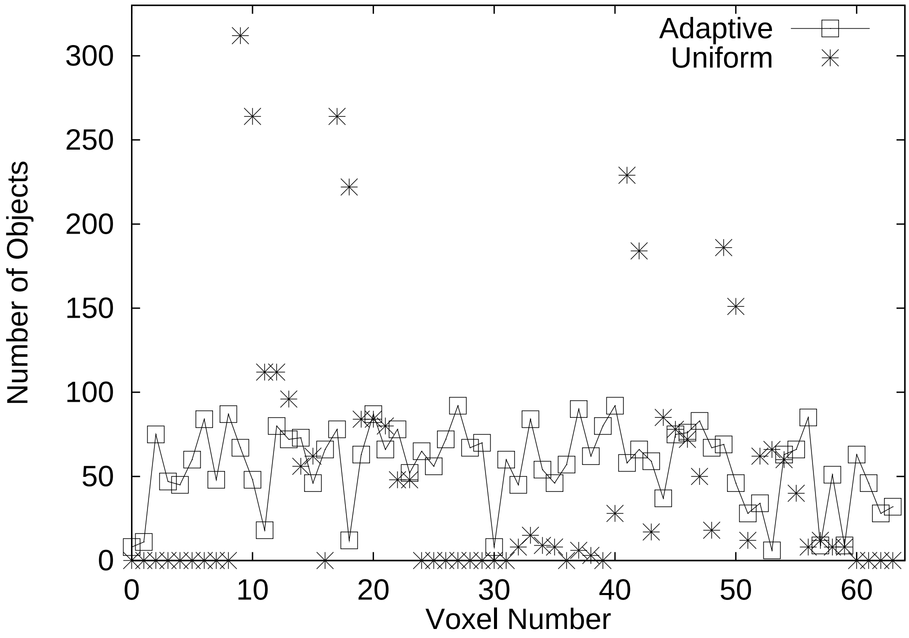


図 28: Teapot の各ボクセルの物体数

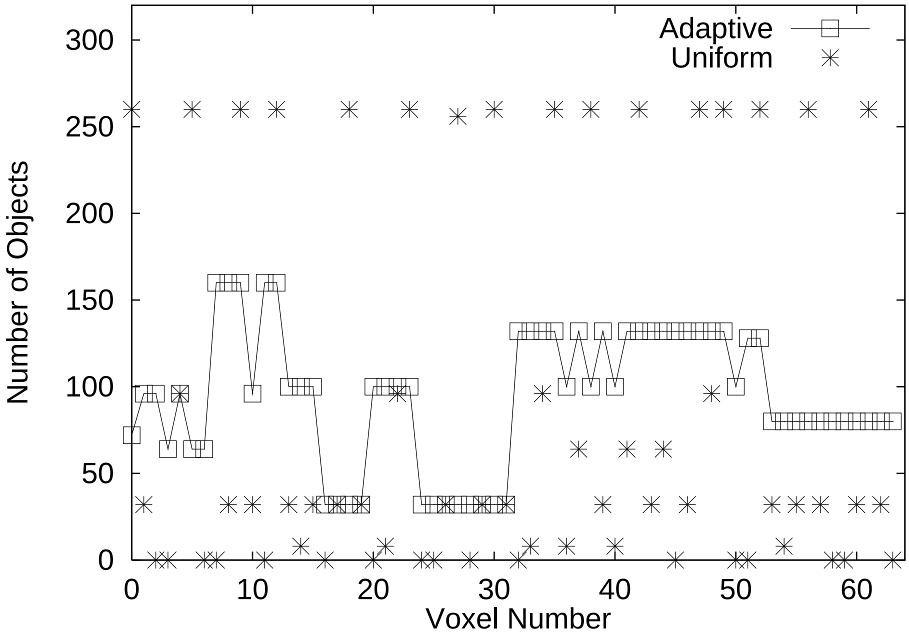


図 29: Tetra の各ボクセルの物体数

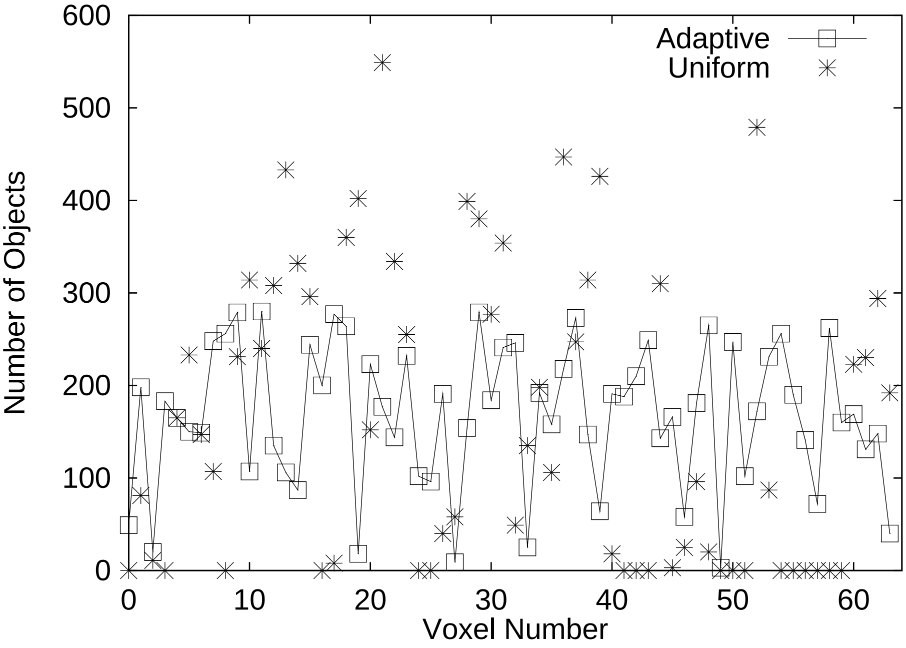


図 30: Mount の各ボクセルの物体数

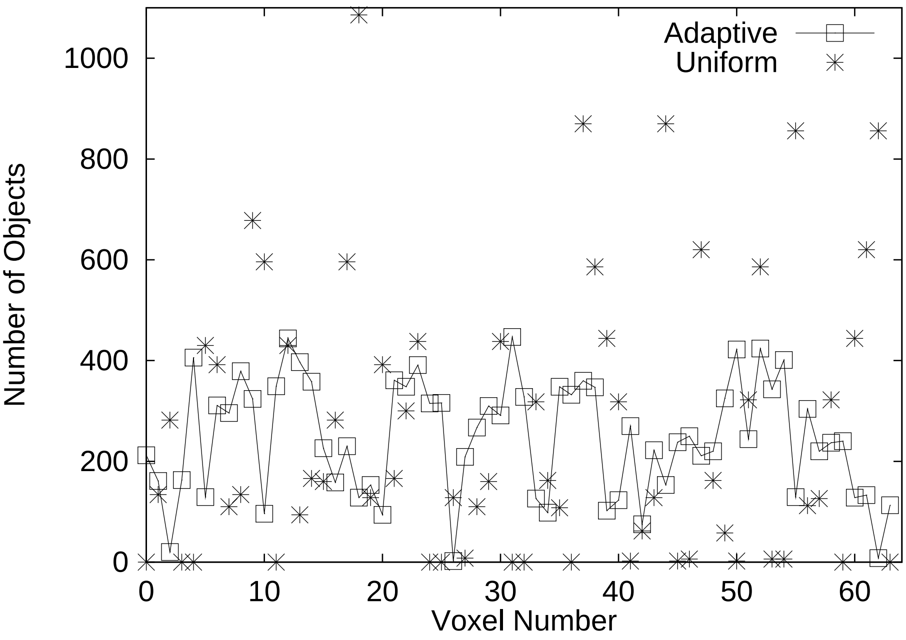


図 31: Balls の各ボクセルの物体数

表 5に、等空間分割と適応型空間分割による結果を示す。それらをグラフにしたものを図 32と図 33に示す。データ 1 ~ 6 について、通常物体と鏡面物体の実行時間を、等分割と適応型分割を比較して示す。また、() 内は等分割に対する比を示している。各データの画像とそのボクセルの配置を、付録の図 34 ~ 図 45に示す。

等分割と適応型分割の実行時間は、通常物体の場合は適応型分割が等分割の 0.29 倍から 1.05 倍、鏡面物体の場合で適応型分割が等分割の 0.31 倍から 1.03 倍である。データ 2 ~ 6 では、1.5 倍から 3 倍程度の速度向上が図られている。データ 1 では、適応型分割の方が悪い結果となった。データ 1 では物体数は少ないが、分割後の物体数が大きく増えているためである。

図 32と図 33を比較すると、通常物体に比べ鏡面物体では実行時間がほぼ 2 倍の値となっている。これは、鏡面により光線が約 2 倍の数になっているためと考えられる。

表 5: 等空間分割と適応型空間分割の実行結果 (単位:秒)

データ (物体数)			通常物体		鏡面物体	
			等分割	適応型分割	等分割	適応型分割
1	Jack	(312)	153 (1.00)	162 (1.05)	194 (1.00)	200 (1.03)
2	Skull	(1243)	380 (1.00)	233 (0.61)	590 (1.00)	297 (0.50)
3	Teapot	(2329)	2670 (1.00)	778 (0.29)	3571 (1.00)	1098 (0.31)
4	Tetra	(4096)	774 (1.00)	439 (0.57)	1035 (1.00)	561 (0.54)
5	Mount	(8192)	2916 (1.00)	1557 (0.53)	4378 (1.00)	2426 (0.55)
6	Balls	(14762)	2715 (1.00)	1950 (0.72)	5710 (1.00)	3605 (0.63)

() 内は等分割に対する比

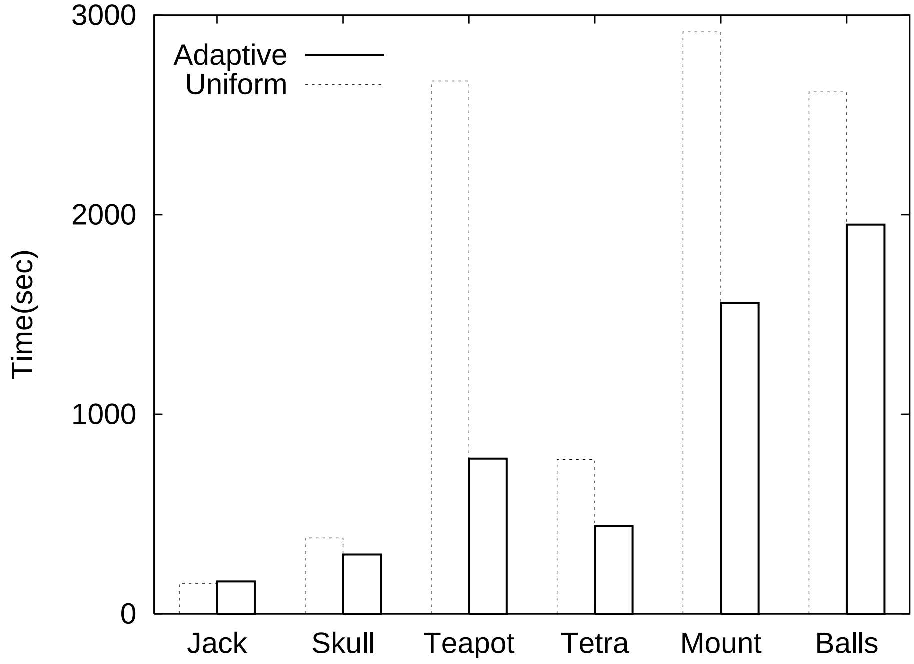


図 32: 通常物体を用いた空間分割の結果

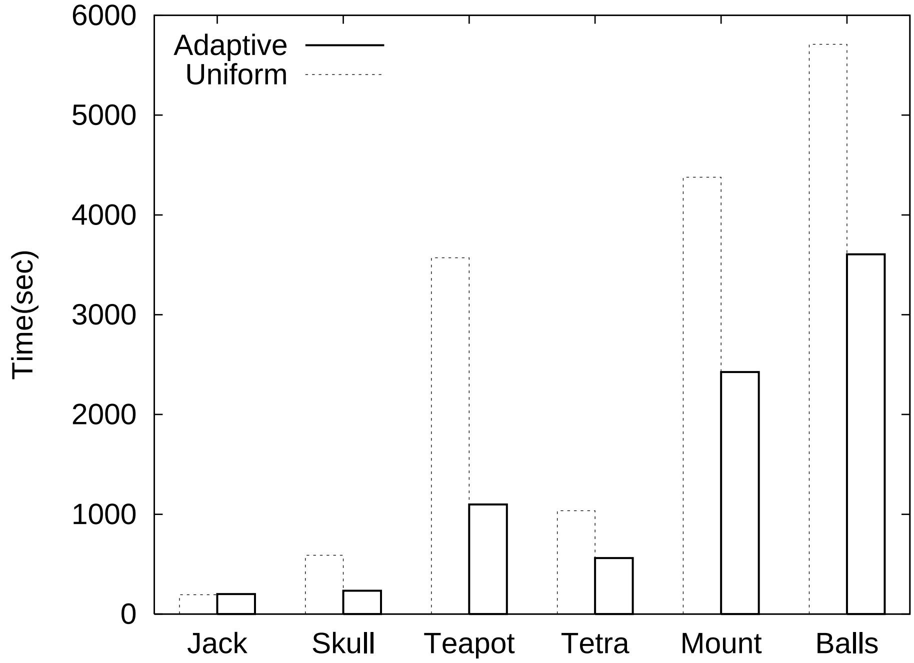


図 33: 鏡面物体を用いた空間分割の結果

6　おわりに

本論文では、空間分割によるレイトレーシング法の並列化について述べた。レイトレーシング法がその名の通り光線を探索し、すべての物体との交差判定を行うことから、空間分割を行い並列化を行った。従来の研究で約60～64倍の並列効果を出した画面インターリーブ分割と、本研究でとりあげた等空間分割に対して実行時間を比較することで、空間分割の有効性が確認できた。さらに、等空間分割と、物体数を均等にするようなボクセル決定をした適応型空間分割の実行時間の比較を行うことにより、後者の有効性を示した。

画面インターリーブ分割と等空間分割の比較では、64プロセッサ使用時、通常物体、鏡面物体それぞれに対して64,128,256,512,1024,2048,4096,8192物体を均等に配置したものとランダムに配置したものを用いた。なお、画面インターリーブ分割では全物体を各プロセッサに登録するため、最大でも256物体までの結果となっている。どちらを扱った場合に対しても、約256物体以上のとき等空間分割の有効性が得られた。さらに等空間分割と適応型空間分割の比較では、SPD(Standard Procedural Databases)の中から6つのデータを用い実験した。その中でも、2329物体を扱ったTeapotデータについて、等分割と比べたとき実行時間を0.29倍にまで短縮できた。そのとき、プロセッサに割り当てられた各ボクセルには、平均して約60物体が登録されているが、最大で登録されても92物体と最も均等化が図れている。最も物体数の少ない312物体を扱ったJackデータについては、適応型分割後の全物体数が大きく増え、等空間分割と比べると適応型分割の結果が悪くなった。以上のことから、物体数が約300以下のときには画面インターリーブ分割の方が良いものの、複雑で品質の良い画像生成を考えそれ以上の物体数を用いたとき、適応型空間分割の有効性が得られる。

今後の課題として、本システムで実装できる基本立体は球および三角形だけであるため、より複雑な画像を考えると基本立体形状の追加が挙げられる。しかし、複雑な基本立体であるほど、そのボクセルへの登録計算は複雑になる。また、本研究ではプロセッサ64台のみの結果であるため、並列効果を知る意味でプロセッサ数を変化させた場合の実験が必要である。そして、各プロセッサにおける交差判定の回数をより均等化するための別のボクセル決定方法についても考えていかなければならない。

謝辞

本研究に機会を与えてくださり、御指導を頂きました山崎勝弘教授に心より深く感謝いたします。また、本研究に際し、数々の助言を頂きました上嶋明氏に感謝いたします。さらに、本研究に助言をして頂いた山崎研究室の皆様に感謝の意を表します。

参考文献

[1] 横井 茂樹, 安田 孝美, “コンピュータグラフィックス [I]”, 電子情報通信学会誌, Vol.76 No.3, pp.277–283(1993.3).

[2] 横井 茂樹, 安田 孝美, “コンピュータグラフィックス [II]”, 電子情報通信学会誌, Vol.76 No.5, pp.529–537(1993.5).

[3] 横井 茂樹, 安田 孝美, “コンピュータグラフィックス [III ・完]”, 電子情報通信学会誌, Vol.76 No.6, pp.626–633(1993.6).

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

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

[6] Adelene Ng and Mel Slater, “A Multiprocessor Implementation of Radiosity”, Computer Graphics forum,Volume 12,number 5,pp.329–342(1993).

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

[8] 上嶋明，山崎勝弘，渡部透，得丸英勝, “高並列計算機 AP1000+上での並列ラジオシティ法”，情報処理学会第５３回全国大会,7P-10(1996.9).

[9] 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,pp.9–41(1991).

[10] 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,pp.155–156(1989).

[11] 河合利幸, 大西充久, �富木順一 , 大西啓修, “画像生成システム MAGG のための画面分割型並列視線探索法とその評価”, 電子情報通信学会論文誌 D-II, Vol.J76-D-II, No.8, pp.1763-1771(1993.8).

[12] 大西充久, 竹安俊幸, 河合利幸, 椦木順一 , 大西啓修, “画像生成装置 MAGG のための画面分割による並列視線探索法”, 情報処理学会第 43 回全国大会, 5U-1(1991).

[13] 竹安俊幸, 大西充久, 河合利幸, 椦木順一 , 大西啓修, “画像生成装置 MAGG のための空間分割による並列視線探索法”, 情報処理学会第 43 回全国大会, 5U-2(1991).

[14] 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).

[15] 原田博之, “マルチプロセッサシステム上での画面／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).

[16] 原田博之, 山崎勝弘, “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”, 情報処理学会第 50 回全国大会, 1C-6(1995.3).

[17] 吉谷崇史, “画面分割によるレイトレーシング法の並列化”, 立命館大学理工学部情報工学科卒業論文 (1995).

[18] 吉谷崇史, 原田博之, 山崎勝弘, “画面分割によるレイトレーシング法の並列化”, 第 39 回システム制御情報学会, 3004,pp.169–170(1995.5).

[19] 吉谷崇史, 山崎勝弘, “空間分割型並列レイトレーシング法の実現と評価”, 情報処理学会第 52 回全国大会, 4L-6,6-131(1996.3).

[20] 吉谷崇史, 山崎勝弘, “適応型空間分割による並列レイトレーシング法”, 情報処理学会第 54 回全国大会,4Q-6(1997.3).

[21] Andrew S.Glassner, “Space Subdivision for Fast Ray Tracing”, IEEE Computer Graphics and Applications, vol.4,no.10,pp.16–22(1987.10).

[22] 山本正樹, 中井泰明, 村上安範, “トランスピュータ入門”, 日刊工業新聞社,pp.159–215(1990).

[23] John G.Cleary, Brain M.Wyvill, Graham M.Birtwistle and Reddy Vatti, “Multiprocessor Ray Tracing”, IEEE Computer Graphics and Applications, vol.5, no.1, pp.3–12(1986.3).

[24] Robert Endl and Manfred Sommer, “Classification of Ray-Generators in Uniform Subdivisions and Octrees for Ray Tracing”, Computer Graphics forum,vol. 13,no. 1,pp.3–19(1994).

[25] B.S.S.Pradhan and A. Mukhopadhyay, “Adaptive Cell Division for Ray Tracing”, Computer & Graphics, vol.15, no. 4, pp.549–552(1991).

[26] Hiroaki Kobayashi, Hideyuki Kubota and Susumu Horiguchi, “Load Balancing Based on Load Coherence between Continuous Images for an Object-Space Parallel Ray-Tracing”, IEICE TRANS.INF.&SYST, vol.E76-D, no. 12 , pp.1490-1498(1993).

[27] M.D.J.McNeill, B.C.Shah, M.-P.Hebert, P.F.Lister and R.L.Girmsdale, “Performance of Space Subdivision Techniques in Ray Tracing”, Computer Graphics forum,vol.11, no.4, pp.213–220(1992.10).

[28] Granigantini and H.H. Atkinson, “Ray Tracing an Octree:Numerical Evaluation of the First Intersection”, Computer Graphics forum, vol.12, no.4, pp.199–210(1993).

[29] Didier Badouel and Thierry Priol, “An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architectures”, Computer & Graphics, vol.15, no.4, pp.549–552(1991).

[30] 高桑昌男, “CG レイトレ物語”, アスキー出版局,pp.64-67(1990.8).

A　付録

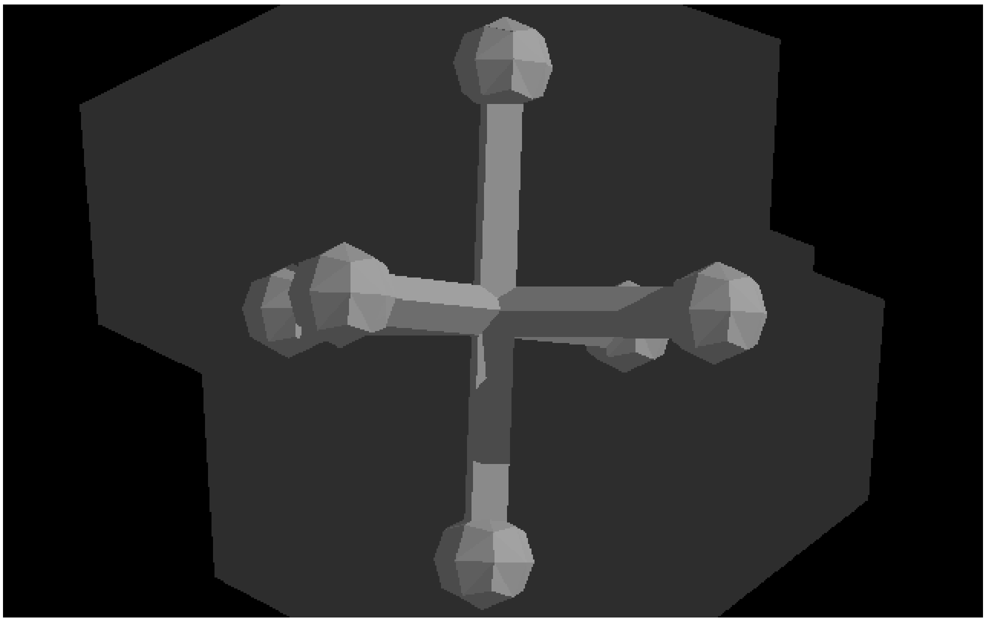


図 34: データ 1(Jack) による画像

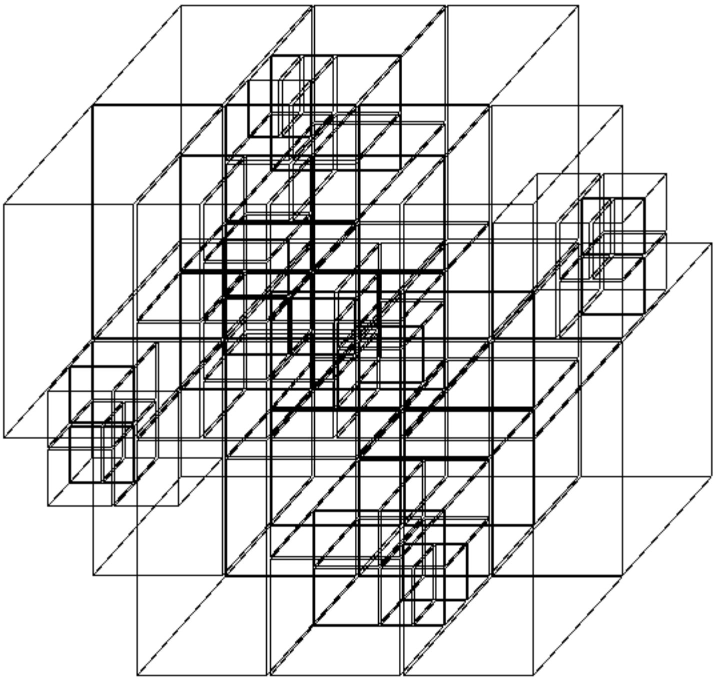


図 35: データ 1 のボクセル配置

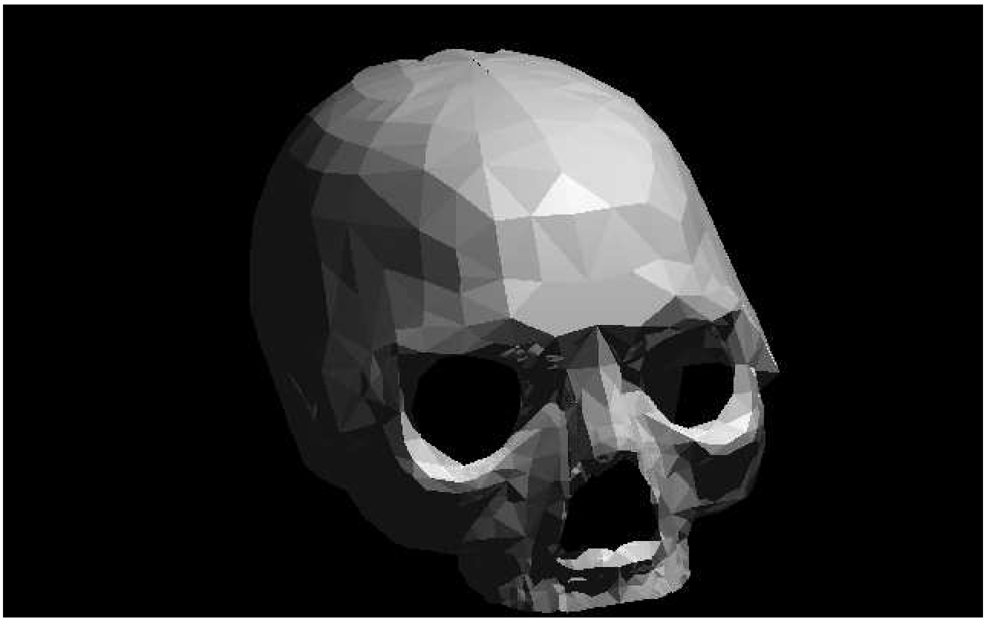


図 36: データ 2(Skull) による画像

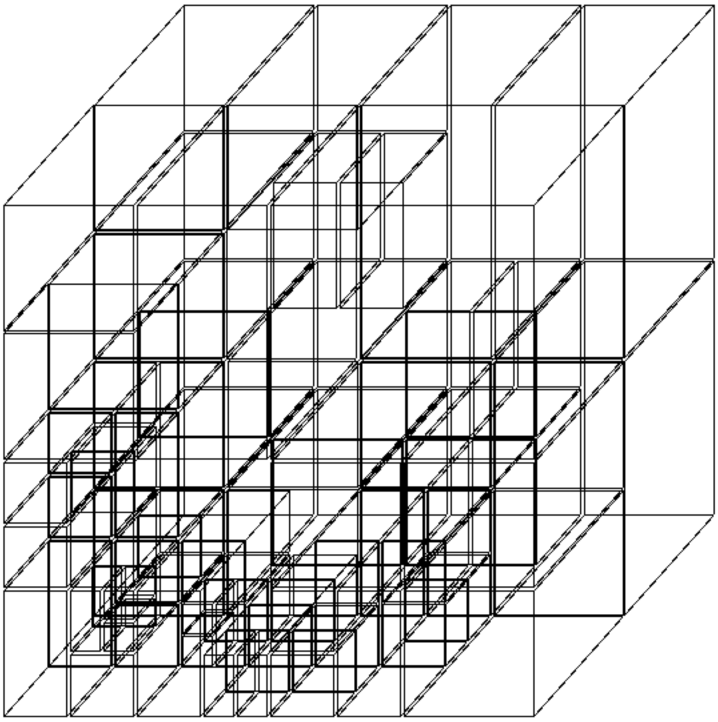


図 37: データ 2 のボクセル配置

EPS File teapote.ps.epsf not found

図 38: データ 3(Teapot) による画像

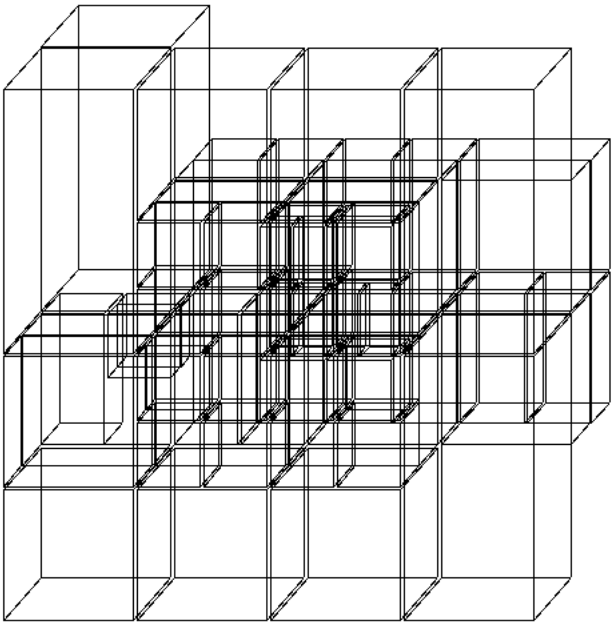


図 39: データ 3 のボクセル配置

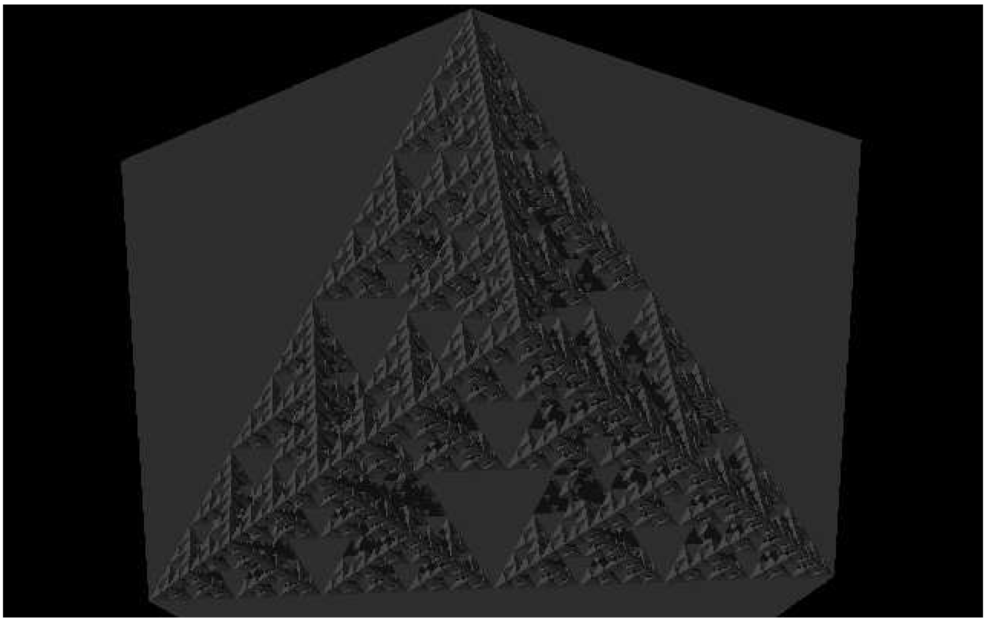


図 40: データ 4(Tetra) による画像

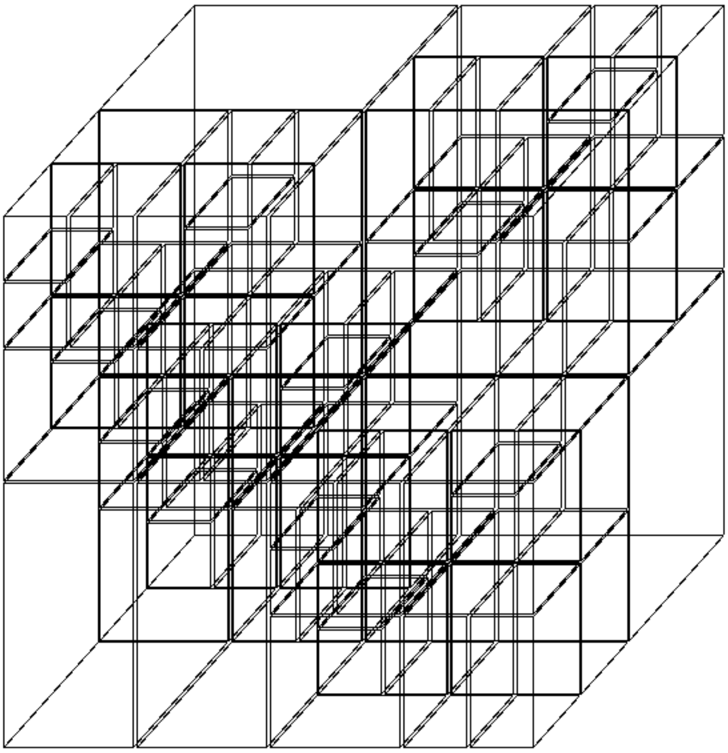


図 41: データ 4 のボクセル配置

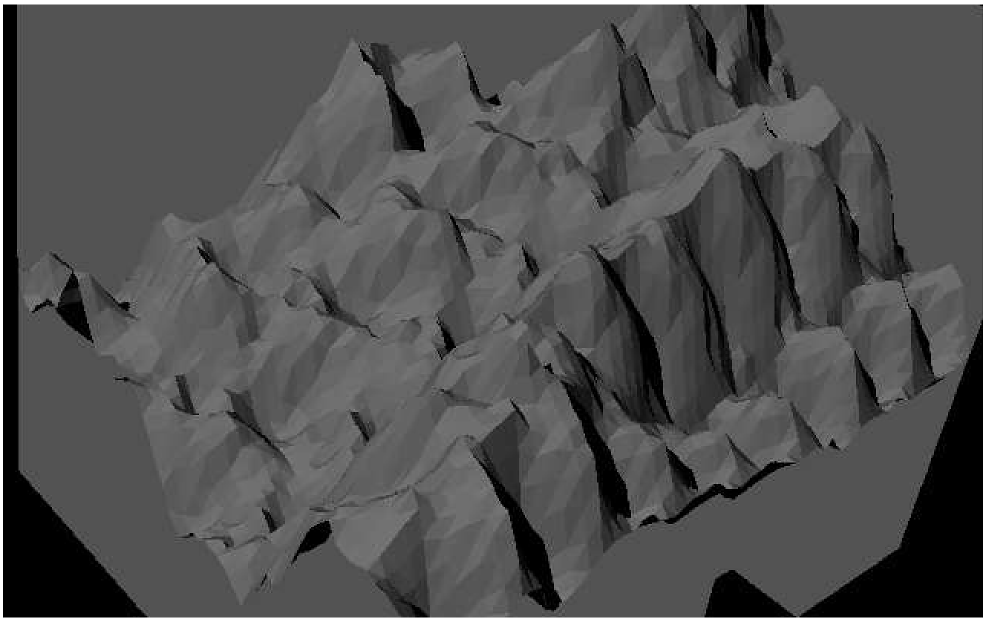


図 42: データ 5(Mount) による画像

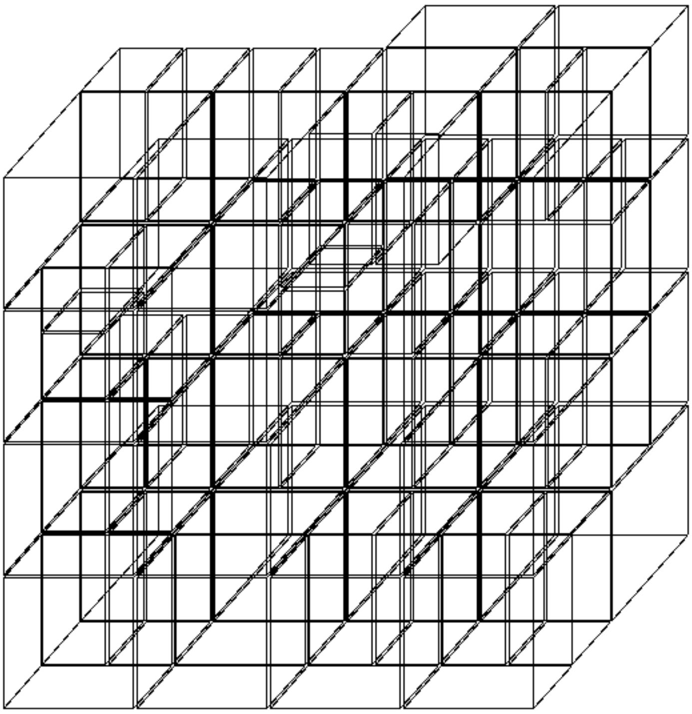


図 43: データ 5 のボクセル配置

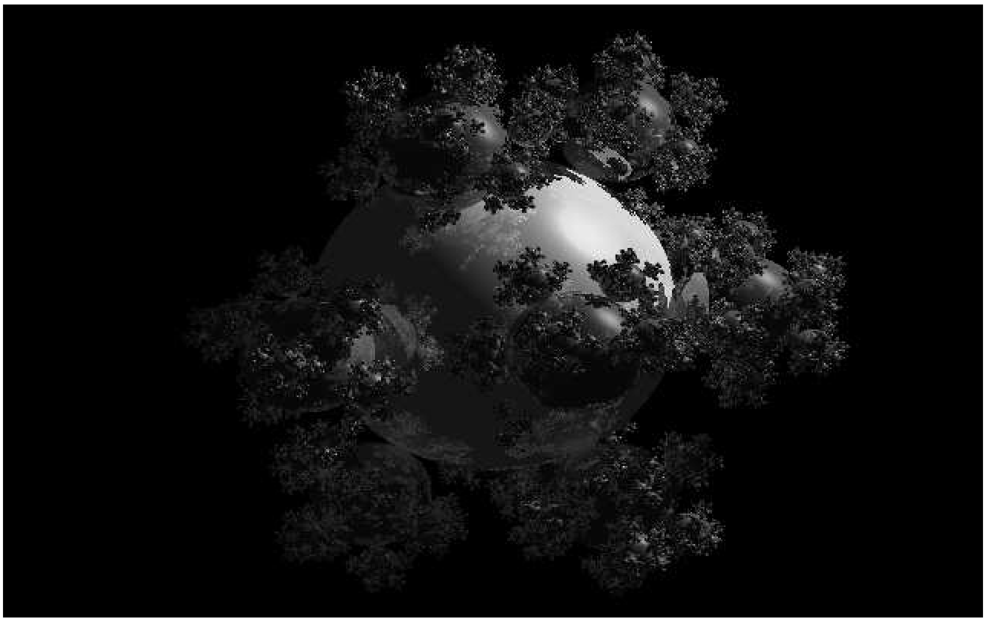


図 44: データ 6(Balls) による画像

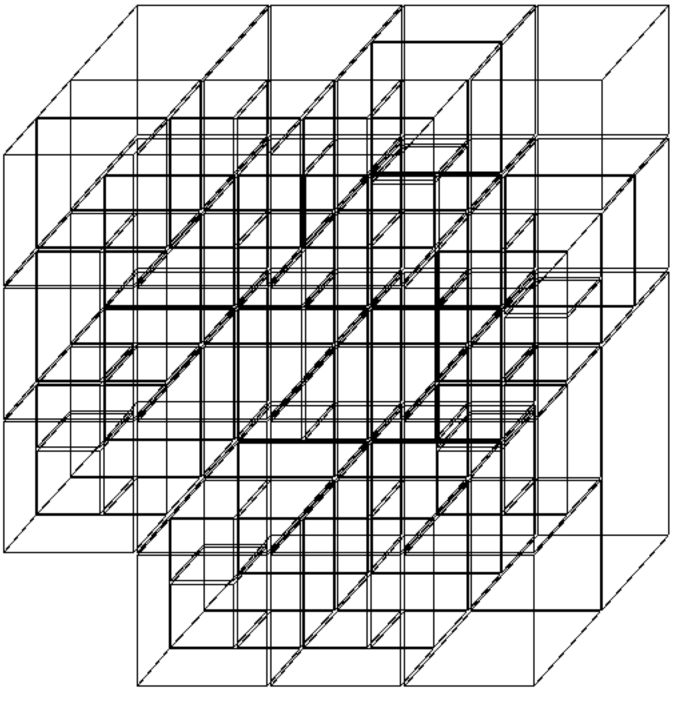


図 45: データ 6 のボクセル配置