修士論文

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法の高速化

指導教員 ： 山崎 勝弘 教授

論文提出者氏名 ： 原田 博之
学籍番号 6027930019−2

1995 年 2 月 17 日提出

立命館大学大学院理工学研究科情報工学専攻

内容梗概

本論文では，レイトレーシング法による画像生成の並列化手法とその効果について述べる．

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，膨大な計算量を必要とし，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台をリング接続，トーラス接続で用いて，画面分割と空間分割による並列化を行い，高速化する方法について検討した．

画面分割では，物体記述データから画面全体の負荷を予測し，画面を分割する負荷均衡分割と等分割との比較を行う．各々で画面をメッシュ分割，縦分割，横分割し，分割されたブロックを各トランスピュータに割り当てる．そして，それをリング接続とトーラス接続で比較した．

空間分割では，空間をボクセルとよ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．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，各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プロセッサにロードできない物体データを用いたものでも実行可能となる．本システム上で実装するために，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，処理の単位となるパケットの定義とそれらデータの収集方法を考えた．そして，3DDDAを用いた空間分割の効果を試すために，1プロセッサによるテストを行った．

並列化の結果，画面分割では，規則的な画像において最高で59.4倍の相対速度が得られ，不規則な画像では等分割と比べて負荷均衡分割の方がよくなることがわかった．空間分割では，3DDDAにより高速化が見られ，空間分割とそうでないものを比べると，実行時間を最高で2.8倍にまで高速化することができた．

目 次

図目次

表目次

第 1 章　はじめに

最近，仮想現実感，科学技術計算結果の可視化などグラフィックス処理の普及に伴って，超高速，実時間処理を可能とするシステムの要望が強くなってきている．また，RISC プロセッサの高速化は目を見張るものがあり， SuperSPARC, Alpha(DEC), R4400(MIPS), PowerPC などの多数のプロセッサが開発され，商用化されている．これらのプロセッサは単体性能としては超高速であるが，リアルタイム処理を可能とするグラフィックス処理を行うには，なお不十分であり，並列処理による高速化が求められている [1]．

並列処理には多くの方法があるが，本研究では，マルチプロセッサによる並列処理を中心に研究を行う．マルチプロセッサはそのアーキテクチャの種類によりいくつかの種類に分けられるが，そのうちの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システムを用いて並列処理を行う．トランスピュータシステムは，25MHz の T805 が 64 台用意されており，これらを，各トランスピュータがもつ 4 本のシリアルリンクにより，トーラス状あるいはハイパーキューブ状に接続することができる．

レイトレーシング法は，1968 年に Appel によって考案された画像生成手法である．1980 年に Whitted によって提案された透過，鏡面反射モデルにより，さらに写実的な画像を生成できるようになった．この手法は，光線追跡法，あるいは視線逆追跡法と呼ばれるように，画面上の各ピクセルの方向から視点に入ってくる光線を逆に追跡し，その輝度，色を求めるものである．光線の追跡の過程で，最初にぶつかった物体がそのピクセルの方向に見えることになる．レイトレーシング法は，このように反射，屈折などの光学現象を容易にシュミレーションできる反面，計算量が非常に多いことが欠点である．

その高速化手法として，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物体空間分割法 [2]，バウンディングボリューム分割法 [3]，そして並列処理を用いた手法がある．並列処理の方式として現在，次のような方法が研究されている．命令レベルの並列処理では，ポリゴンや 2 次曲面の交点を求めるための方程式の各係数を独立に計算を行う [4]．パイプライン処理では，1 本の光線の処理を物体空間探索，交点計算，輝度計算の 3 つのタスクに分け，それぞれ別の専用装置を設け，パイプライン処理を行う [4][5]．マルチプロセッサレベルの並列処理では，光線，物体，物体空間の処理が互いに独立である点に着目して，各処理をプロセッサで並列に行う．一般的に，空間分割方式と画面分割方式がある．空間分割方式では，空間等分割による分割 [6]，各プロセッサの物体データが均等になるまで空間を分割し，負荷分散を行う方法 [7]，オクトリによる分割 [2][8]～[10] などがある．画面分割方式では，画面をピクセル単位で分割し，各プロセッサに分配して並列処理を行う．これにはデータベース複写型 [11]～[13]，データベース共有型 [14] [15] の 2 つの方式がある．また，仮想共有メモリをもったトランスピュータシステムによる並列処理 [16]，レイトレーシングによるアニメーションの並列処理 [17] など盛んに研究されている．

本研究では，データベース共有型画面分割と空間等分割を扱ったマルチプロセッサレベルの並列処理 [18] を行う．

従来の画面等分割 [11] では，どうしても各プロセッサに負荷の偏りが生じていた．そこで画面全体の負荷を予測し，負荷の均衡を図ることで高速化を図る．物体記述データから画面上の物体のおおよその位置を把握し，描かれる物体の情報を得る．そして全体の負荷を求め，分割される画面の負荷が均等になるようにする．うまく負荷均衡が行われれば，高速化が望める．画面の分割は，メッシュ分割，縦分割，横分割の 3 通りで行い，実行時間を比較する．また，データの回収方法を 3 パターン用意し，その通信距離，画像データの関係を考察する．

空間分割は，物体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，物体が記述されているデータベースすべて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をもてばよい．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，交差判定を軽減できる．本研究では，空間分割をトランスピュータに実装するためのアルゴリズムを考える．光線がどの空間を突き抜けるかを知るための 3DDDA，各プロセッサ間の処理の基本となるパケットの定義，各プロセッサに実装されるタスクとスレッド，そしてプロセッサ間で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．

本論文では，2 章でレイトレーシング法による画像生成方法と，その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，並列化手法の 1 つである，負荷の予測から負荷均衡を図る画面分割について述べる．5 章では，3 次元的な分割を行う空間分割による並列化手法について述べる．そして 6 章では，レイトレーシング法による画像生成の並列化と，画面負荷均衡の効果を示すためにトランスピュータの台数による実行時間を比較し，その考察を行う．また，空間分割では，3DDDA を用いた 1 台による性能評価を行う．最後に 7 章では，本研究の成果と今後の課題を述べる．

第 2 章　レイトレーシング法

2.1　レイトレーシング法の特徴

レイトレーシング法の原理は非常に単純である．図 2.1 に示すように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，つまり最も手前に存在する物体を探索する方法である．人が物を見る過程と逆方向，つまり視点から光源への視線を出し，視線と物体の交点から物体を表示する．レイトレーシングでは陰面消去と輝度計算からなり，陰面消去では視線と交差する物体があるかどうかを調べ (交差判定)，最も視点に近い物体を求める．輝度計算では各ピクセルの色を決めるため，物体から光源の方向に二次光線を出し，影による交差判定を行う．物体に反射，屈折特性があるときは，反射，屈折方向に二次光線を出す．交差判定は陰面消去と輝度計算に現れ，レイトレーシングの処理のうち簡単な画像で約 75%，複雑な画像では約 95%を要すると指摘されている [19]

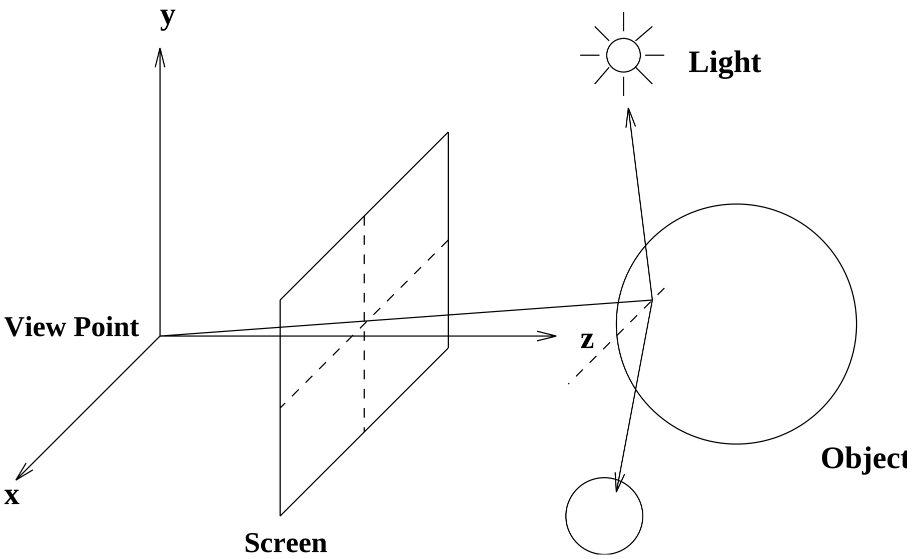

図 2.1. レイトレーシング法の原理

2.2 レイトレーシング法による作図

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，$x - y$平面上に水平に置くことにする．視点 (v_x, v_y, v_z) は，z軸の負方向つまり，スクリーンの中心に立てた垂線上に置いている．

2.2.1 2次曲面との交点計算

レイトレーシングを行うということは直線と物体との交点を求めることにほかならない．ゆえにどのような物体を描くにせよ，物体を定義した方程式が必要である．ここでは交点計算に，簡単な根の公式が使える2次曲面を用いる，その中でも楕円面 (ellipsoid)，一葉双曲面 (hyperboloid of one sheet)，二葉双曲面 (hyperboloid of two sheets)，楕円錐面 (elliptical cone) を扱い，これらと視線との交点を求める方法を述べる．

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 (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，

$$\begin{aligned} x &= E_x t + v_x & E_x &= \tfrac{p_x - v_x}{\iota} \\ y &= E_y t + v_y & E_y &= \tfrac{p_y - v_y}{\iota} \\ z &= E_z t + v_z & E_z &= \tfrac{p_z - v_z}{\iota} \end{aligned} \tag{2.1}$$

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

となる．t は，視点から直線上の点のまでの距離である．

- 楕円面

  $$\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

  $a = b = c$ のとき球面となり，a, b, c のいずれか1つを∞にすると楕円柱面，いずれか2つを∞にすると，平行な2つの平面が仮想的に作れる．軸方向のパラメータを無限大にすると円柱ができる．

- 一葉双曲面

  $$\frac{x^2}{a^2} - \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$$

  いずれか2つの係数が正，係数が負となる変数の座標軸方向に曲面が広がる．

- 二葉双曲面

  $$-\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1$$

  いずれか2つの係数が負，係数が正となる変数の座標軸方向に曲面が広がる．

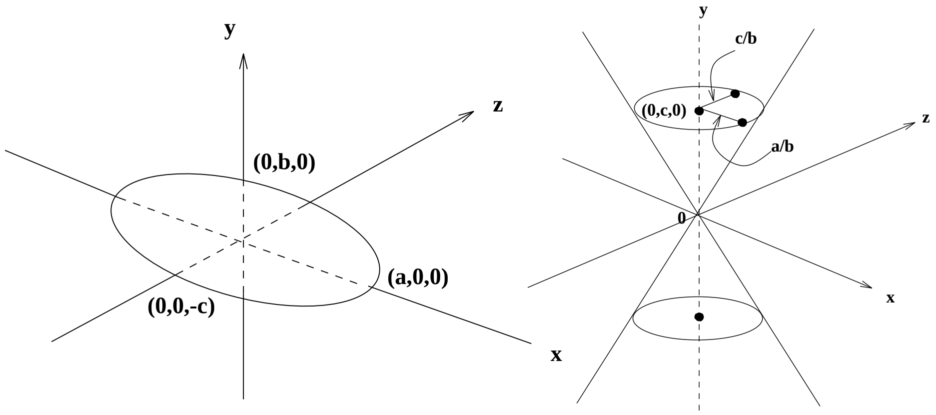

図 2.2. 楕円面 (左) と楕円錘面 (右)

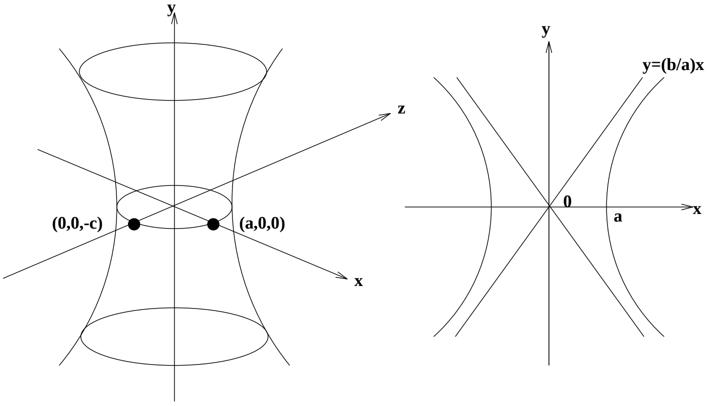

図 2.3. 一葉双曲面

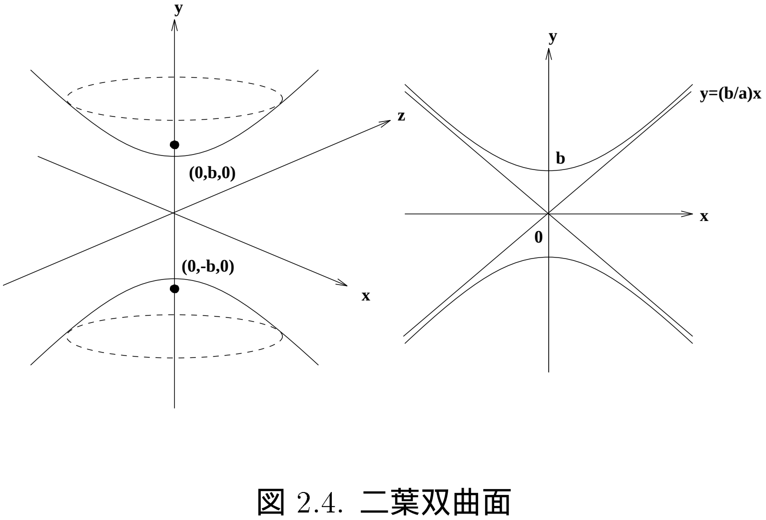

図 2.4. 二葉双曲面

- 楕円錐面

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} + \frac{z^2}{c^2} = 0$$

いずれか 1 つの係数が負で，係数が負となる変数の座標軸方向に曲面が広がる．

これらの 2 次曲面の一般的な標準方程式は，定数$\alpha, \beta, \gamma, \delta$を適当に選べば，次にようになる．

$$\alpha\frac{x^2}{a^2} + \beta\frac{y^2}{b^2} + \gamma\frac{z^2}{c^2} = \delta \tag{2.2}$$

従って，交点計算は t に関する 2 次方程式を解けばよい．

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

$$\begin{aligned}
A &= \alpha' {E_x}^2 + \beta' {E_y}^2 + \gamma' {E_z}^2 = 1 \\
B &= 2(\alpha' E_x v_x + \beta' E_y v_y + \gamma' E_z v_z) \\
C &= \alpha' {v_x}^2 + \beta' {v_y}^2 + \gamma' {v_z}^2 - \delta \\
\alpha' &= \alpha/a^2, \beta' = \beta/b^2, \gamma' = \gamma/c^2
\end{aligned}$$

$$D = B_2 - 4AC \geq 0$$

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

また，平面の場合は次のようになる．原点を通る平面の方程式は，a, b, c をパラメータとして

$$ax + by + cz = 0$$

となり，x, y, zを代入し，t を求めると，

$$t = -\frac{av_x + bv_y + cv_z}{aE_x + bE_y + cE_z}$$

$$\boldsymbol{N} = (a, b, c)$$

一般に交点が求まれば，その点での 2 次曲面の法線ベクトルが必要となってくる．2 次曲面を

$$\alpha\frac{x^2}{a^2} + \beta\frac{y^2}{b^2} + \gamma\frac{z^2}{c^2} = \delta$$

とすると，楕円面，一葉双曲線，楕円錐面では，

$$\boldsymbol{N} = \left(\alpha\frac{x}{a^2},\ \beta\frac{y}{b^2},\ \gamma\frac{z}{c^2}\right)$$

で与えられ，2 次双曲面では，符合を逆にした

$$\boldsymbol{N} = -\left(\alpha\frac{x}{a^2},\ \beta\frac{y}{b^2},\ \gamma\frac{z}{c^2}\right)$$

となる．これらの法線は，内部から外部に向かって出ていく法線ベクトルである．従って，屈折物体内で法線ベクトルを求めたときは，符合を逆にする必要がある．

2.2.2　ハイライト

- Phong のモデルによるハイライトは次式で表される．

鏡面反射光による輝度	I_s
入射光	I_p
鏡面反射係数	$k_s\ (k_{sR}, k_{sG}, k_{sB})$
光線の正反射方向と視線が成す角	α
視線ベクトル	$\boldsymbol{E} = (E_x, E_y, E_z)$
光源ベクトル	$\boldsymbol{L} = (L_x, L_y, L_z)$
光線の正反射ベクトル	$\boldsymbol{R} = (R_x, R_y, R_z)$
法線ベクトル	$\boldsymbol{N} = (n_x, n_y, n_z)$
	$\boldsymbol{V} = (V_x, V_y, V_z) = -\boldsymbol{E}$

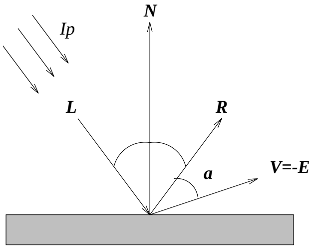

図 2.5. ハイライト

Phong のモデル

$$\begin{aligned} I_s &= k_s I_p cos^n \alpha \\ &= k_s I_p (\boldsymbol{R} \cdot \boldsymbol{V})^n \qquad (2.3) \end{aligned}$$

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．

$$R = 2(\boldsymbol{L} \cdot \boldsymbol{N})\boldsymbol{N} - \boldsymbol{L} \qquad (2.4)$$

$$(\boldsymbol{L} \cdot \boldsymbol{N} = L_x n_x + L_y n_y + L_z n_z)$$

- 金属質感

  鏡面加工された金属，特にそれ自体に地色をもつ金や銅などは反射する色自体もその色の影響を受けてしまう．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．Phong のモデルでは，波長については考慮されていない．

  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** がある．このモデルでは，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衰係数 (Phong では k_s) によりモデル化されている [3]．

  現在のレンダリングシステムではあまり採用されていない方法で，反射率を物体ごとにコントロールする方法がある．この手法を使うことで金属などの質感が飛躍的に向上する．自然に存在する物体はその原子配列や分子結合によって表面のミクロ単位での凹凸に個別の特性を持っている．そこに光が当たったときに，独自の色だけを反射したりすることで色を出すわけであるが，それともう 1 つの特性として，一定の角度で入射した光は一定の反射率反射する．つまり言い換えると光の入射角 (レイトレでは視線の入射角) によって反射率が変わるということである．図 2.6に金，銀，ガラスの反射率のグラフを示す．

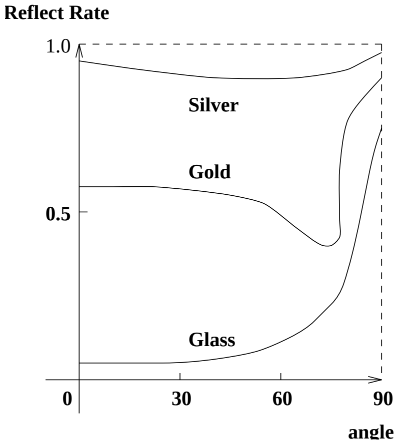

図 2.6. 反射率

2.2.3 反射光と環境光

反射光には拡散反射光 (diffusely reflected light) と鏡面反射光 (specularly reflected light) があり，前者は物体の色そのものを反映し，後者は物体が鏡のような色を反映する．

- 拡散反射光
  拡散反射はあらゆる方向に反射する．従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．

直接光による拡散反射光	$I_d\ (I_{dR}, I_{dG}, I_{dB})$
直接光の光の強さ	$I_p\ (I_{pR}, I_{pG}, I_{pB})$
拡散反射係数	$k_d\ (k_{dR}, k_{dG}, k_{dB})$
入射角	$\theta(0 \sim \pi/2)$

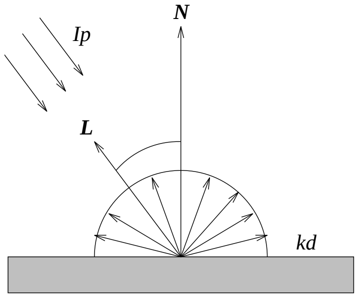

図 2.7. 拡散反射光

Lambert の余弦則 (Lambert’s cosine low) による拡散反射光は次式で表される．

$$I_d = k_d I_p cos(\theta) = k_d I_p(\boldsymbol{L}\ \cdot \boldsymbol{N}) \tag{2.5}$$

- 鏡面反射光
  鏡面反射は視線が鏡面で正反射し，その視線があたる物体表面の色を与える．

視線ベクトル	$\boldsymbol{E} = (E_x, E_y, E_z)$
正反射ベクトル	$\boldsymbol{R} = (R_x, R_y, R_z)$
法線ベクトル	$\boldsymbol{N} = (n_x, n_y, n_z)$

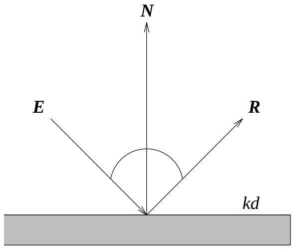

図 2.8. 鏡面反射光

$$\boldsymbol{R} = 2(-\boldsymbol{E} \cdot \boldsymbol{N})\boldsymbol{N} - \boldsymbol{E}$$
$$(\boldsymbol{E} \cdot \boldsymbol{N} = E_x n_x + E_y n_y + E_z n_z)$$

- 相互反射光/環境光 (Ambient Light)

  通常は，壁や床などの周囲の物体に依存する間接光 (相互反射光) が存在するため，真っ暗な陰は生じない．ここでは，簡便にその作用の効果をだすために直接光以外の一定の環境光による表面輝度でモデル化する．

環境光による拡散反射光	I_{ad}	$I_d + I_{ad} = k_d I_p(\boldsymbol{L} \cdot \boldsymbol{N}) + k_d I_a$
環境光の光の強さ	I_a	$I_{ad} = k_d I_a$

2.2.4 透過光と屈折

透過とは，屈折はさほど感じられなくとも，厚いガラスにように，その向こう側の光景がすけて見えることである．透過光 (Transmitted Light) は，その淡い色が透明物体の存在を示している透明感を誘う．透過係数を k_tとすると透過光 I_tは，次のようになる．

$$I_t = k_t I_B$$

屈折 (refraction) とは，光線が別の媒質に入射したときに，光線の方向が曲げられる現象であり，透明度を強調するものである．透過光で屈折を考慮した表面輝度，つまり目に飛び込んでくる光 Iは次のようになる．

$$\begin{aligned} I &= \text{拡散反射光 } (I_d) + \text{環境光 } (I_{ad}) + \text{鏡面反射光 } (I_s) + \text{透過光 } (I_t) \\ &= k_d I_{do} + k_d I_{ao} + k_s I_{so} + k_t I_{to} \\ &= k_d I_p(\boldsymbol{L} \cdot \boldsymbol{N}) + k_d I_a + k_s I_p(\boldsymbol{R} \cdot \boldsymbol{V})^n + k_t I_B \end{aligned} \tag{2.6}$$

屈折透過光 I_tを定めるために，入射した光線がどの方向に向かうのかを調べる必要がある．それに答えるのがスネルの法則 (Snell's law) である．物体 1 から物体 2 に入射すると

き，n_{12}を 1 に対する 2 の屈折率とよび，以後 n とする．また，物体 1 から物体 2 に入射するとき，2 に対する 1 の屈折率 n_{21} は，$\frac{1}{n}$となる．

$$n = n_{12} = \frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1}$$

$$n_1 sin\theta_1 = n_2 sin\theta_2$$

物体 1 における反射係数	k_{s1}
物体 2 における反射係数	k_{s2}
物体 1 から物体 2 へ透過するときの透過係数	k_{t1}
物体 2 から物体 1 へ透過するときの透過係数	k_{t2}
入射角	θ_1
屈折角	θ_2
物体 1 の絶対屈折率	n_1
物体 2 の絶対屈折率	n_2

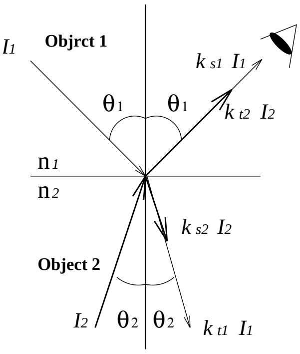

図 2.9. 光の反射と屈折

屈折によって目に飛び込んでくる光は，図 2.9に示すように，$k_{s1}I_1 + k_{t2}I_2$であり，鏡面反射光と透過光の 2 つの光である．この反射係数 k_{s1}, k_{s2}を決めるには，フレネルの公式 (Fresnel Formula) を用いる

物体 1 側の反射係数 k_{s1}フレネルの公式より次のようになる

$$k_{s1} = \frac{1}{2}\left\{\frac{sin^2(\theta_1 - \theta_2)}{sin^2(\theta_1 + \theta_2)} + \frac{tan^2(\theta_1 - \theta_2)}{tan^2(\theta_1 + \theta_2)}\right\} \tag{2.7}$$

また，θ_1とθ_2を交換してもよいことより，

$$k_{s1} = k_{s2},\ k_{t1} = k_{t2}$$

が得られる．k_{s1}は，加法定理を用いて次にように変形できる．

$$\begin{aligned} k_{s1} &= \frac{1}{2}\left\{\left(\frac{sin\theta_1 cos\theta_2 + cos\theta_1 sin\theta_2}{sin\theta_1 cos\theta_2 - cos\theta_1 sin\theta_2}\right)^2 + \left(\frac{sin\theta_1 cos\theta_2 + cos\theta_1 sin\theta_2}{sin\theta_1 cos\theta_2 - cos\theta_1 sin\theta_2}\right)^2\right\} \\ &= \frac{1}{2}\left\{\left(\frac{n \cdot cos\theta_2 - cos\theta_1}{n \cdot cos\theta_2 + cos\theta_1}\right)^2 + \left(\frac{n \cdot cos\theta_1 - cos\theta_2}{n \cdot cos\theta_1 + cos\theta_2}\right)^2\right\} \qquad (2.8) \\ & cos\theta_1 = -\boldsymbol{E} \cdot \boldsymbol{N},\ n = \frac{n_2}{n_1} \end{aligned}$$

また，

$$k_{s1} + k_{t1} = 1$$

$$k_{s2} + k_{t2} = 1$$

であるので，反射係数 k_{s1}を出しから，透過係数 k_{t1}, k_{t2}を求める．

屈折後の視線ベクトル $\boldsymbol{E'}$を求めるには，屈折する前の視線ベクトルを$\boldsymbol{E}$, 屈折した後の視線ベクトルを$\boldsymbol{E'}$とすると，

$$sin\theta_1 = |\boldsymbol{E} + \boldsymbol{N} cos\theta_1| \tag{2.9}$$

$$sin\theta_2 = |\boldsymbol{N} cos\theta_2 + \boldsymbol{E}| \tag{2.10}$$

となり，スネルの法則に代入して

$$n_1|\boldsymbol{E} + \boldsymbol{N} cos\theta_1| = n_2|\boldsymbol{N} cos\theta_1 + \boldsymbol{E'}|$$

$$\begin{aligned} \boldsymbol{E'} &= \frac{n_1}{n_2}\left(E + Ncos\theta_1 - \frac{n_2}{n_1}N\sqrt{1 - sin^2\theta_2}\right) \\ &= \frac{n_1}{n_2}\left\{E - N\frac{n_2}{n_1}\left(\sqrt{1 - sin^2\theta_2} - cos\theta_1\right)\right\} \\ & \quad (sin\theta_2 = sin\theta_1 = \frac{n_1}{n_2}\sqrt{1 - cos^2\theta_1}) \\ & \quad (cos\theta_2 = \sqrt{1 - sin^2\theta_2}) \\ &= \frac{n_1}{n_2}\left\{\boldsymbol{E} - \boldsymbol{N}\left(\sqrt{\left(\frac{n_2}{n_1}\right)^2 - 1 + cos^2\theta_1 - cos\theta_1}\right)\right\} \end{aligned} \tag{2.11}$$

2.2.5 ワールド座標系とオブジェクト座標系

2 次曲面と平面の基本立体は，すべて原点を中心としたもので，実際に空間に配置するためには，何らかの方法でそれらを拡大縮小，回転，平行移動し，配置する必要がある．これらの変換は行列により可能である．方法として，変換後の 2 次曲面の方程式を求めるか，実際に変換したのと同じ効果を実現することである. ここでは次の方法をとっている．基本立体が標準系の 2 次方程式で表されるような座標系を基本立体に張りつけておいて配置し，ワールド座標系で表現された視線をその座標系で読み直し，すなわち視線の方を変換して交点計算を行い，交点が見つかればそれをワールド座標系に直す．

基本立体の座標系をオブジェクト座標系とする．2 次曲面のパラメータは a, b, c とし，その 2 次曲面の中心が原点に来るように設定する．そして平行移動，回転のパラメータを設け，それらの設定により，空間に配置するようにする．

視線，あるいは交点を求める際に，座標変換をする必要がでてくる．オブジェクト座標系で表現された 2 次曲面と交点をワールド座標系で求めるために，ワールド座標系での視線をオブジェクト座標系に変換する必要がある．また，法線ベクトルはオブジェクト座標系によるベクトルなので，それもワールド座標系に変換する必要がある．オブジェクト座標系で得られた交点を，ワールド座標系に直すこともあろう．そこでワールド座標系⇔ オブジェクト座標系変換を求める．

まず，ワールド座標系の点を (x, y, z)，オブジェクト座標系の点を (x', y', z') とし，回転と平行移動の行列を求める．ワールド座標からオブジェクト座標への変換は次のようになる．

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \boldsymbol{M} \cdot \boldsymbol{R}_z(\boldsymbol{\theta}) \cdot \boldsymbol{R}_y(\boldsymbol{\theta}) \cdot \boldsymbol{R}_x(\boldsymbol{\theta}) \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{2.12}$$

- 座標軸回りの回転

  •x 軸回りの回転

$$\boldsymbol{R}_x(\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & -sin\theta & 0 \\ 0 & sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{R}_x(\theta)^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & cos\theta & sin\theta & 0 \\ 0 & -sin\theta & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  •y 軸回りの回転

$$\boldsymbol{R}_y(\theta) = \begin{pmatrix} cos\theta & 0 & sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{R}_y(\theta)^{-1} = \begin{pmatrix} cos\theta & 0 & -sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ sin\theta & 0 & cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

  •z 軸回りの回転

$$\boldsymbol{R}_z(\theta) = \begin{pmatrix} cos\theta & -sin\theta & 0 & 0 \\ sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{R}_z(\theta)^{-1} = \begin{pmatrix} cos\theta & sin\theta & 0 & 0 \\ -sin\theta & cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- 平行移動

  平行移動量を (m_x, m_y, m_z) とする

$$\boldsymbol{M} = \begin{pmatrix} 1 & 0 & 0 & m_x \\ 0 & 1 & 0 & m_y \\ 0 & 0 & 1 & m_z \\ 0 & 0 & 0 & 1 \end{pmatrix}, \boldsymbol{M}^{-1} = \begin{pmatrix} 1 & 0 & 0 & -m_x \\ 0 & 1 & 0 & -m_y \\ 0 & 0 & 1 & -m_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

オブジェクト座標系からワールド座標系への変換は，次のようになる．

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \boldsymbol{R}_x(\boldsymbol{\theta})^{-1} \cdot \boldsymbol{R}_y(\boldsymbol{\theta})^{-1} \cdot \boldsymbol{R}_z(\boldsymbol{\theta})^{-1} \cdot \boldsymbol{M}^{-1} \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} \tag{2.13}$$

2.2.6　テクスチャ

テクスチャとは，野球のボールの縫い目や板の木目など視覚効果を与えるもので，対象物の外観を修正し，場合によっては型を作りかえるモデリング技法である．例えばオレンジでは，表面ででこぼこのテクスチャをもつ球面と考えられる．

ピクチャマッピング

テクスチャで最も広く使われているのは，壁紙や写し絵を張るように，面の上に絵を直接描く方法である．この面に張ろうとしている絵のことを一般的にテクスチャマップ (texture map) と呼び，このテクスチャを張りつける過程をテクスチャマッピング (texture mapping) 呼ぶ．

バンプマッピング

バンプマッピング (bumb mapping) とは，テクスチャを面に適用して，実際に面の形を変えずに，変えたように見せる技法である. 基本的にはトリックであり，錯覚である．物体の形状を与えるものは，その物体の陰影づけの変化であり，それは面の法線が変化する結果として現れる．面の凹凸を与えるテクスチャをバンプマップあるいは摂動マップと呼ばれる [20]．

$Blinn$ の示した方法は次のような理論に基づいている．$\boldsymbol{P}(u,v)$ を，パラメータ $0 < \mathrm{u} < 1$，$0 < \mathrm{v} < 1$ で表される面，$\boldsymbol{F}(u,v)$ を凹凸を持たせるために定義する摂動関数とする．このとき面$\boldsymbol{P}$の遍微分$\boldsymbol{P}_u, \boldsymbol{P}_v$は接平面に含まれるベクトルになる．面$\boldsymbol{P}$の法線は$\boldsymbol{P}_u, \boldsymbol{P}_v$の外積によって表される．ここで面$\boldsymbol{P}$ に摂動関数$\boldsymbol{F}$を作用させてできた新しい面$\boldsymbol{P}'$を次のように定義する [21]．

$$\boldsymbol{P}'(u,v) = \boldsymbol{P}(u,v) + F(u,v)\frac{\boldsymbol{n}}{|\boldsymbol{n}|}$$

但し，$\boldsymbol{n} = \boldsymbol{P}_u \otimes \boldsymbol{P}_v$である．

$\boldsymbol{P}(u,v)$ の法線ベクトル$\boldsymbol{P}_u, \boldsymbol{P}_v$は，法線成分$\frac{\boldsymbol{n}}{|\boldsymbol{n}|}$の微分項の誤差を考えると，

$$\boldsymbol{P}'_u = \boldsymbol{P}_u + F_u\frac{\boldsymbol{n}}{|\boldsymbol{n}|}$$

$$\boldsymbol{P}'_v = \boldsymbol{P}_v + F_v\frac{\boldsymbol{n}}{|\boldsymbol{n}|}$$

となり，結局面$\boldsymbol{P}$の面法線$\boldsymbol{n}$は

$$\boldsymbol{n}' = \boldsymbol{n} + \boldsymbol{D}$$

ただし，

$$\boldsymbol{D} = F_u\frac{\boldsymbol{n} \otimes \boldsymbol{P}_u}{|\boldsymbol{n}|} - F_v\frac{\boldsymbol{n} \otimes \boldsymbol{P}_v}{|\boldsymbol{n}|}$$

ここで波紋のバンプマッピングを考えてみる．波の中心を (u_0, v_0)，波の 1 周期長を w_l，振幅の減衰係数を w_d，波紋の中心での波の振幅を w_hとする．波の振幅の式は $y = Asin\frac{2\pi}{T}t$ なので，振幅 lm は次のようになる．

$$lm = sin(\frac{2\pi}{w_l}lxz) \times \frac{1.0}{1.0 + \frac{lxz}{wd}}w_h$$

lxzは，波紋の中心から距離である．この振幅を摂動関数とし，法線ベクトルを求めれば波紋ができる．

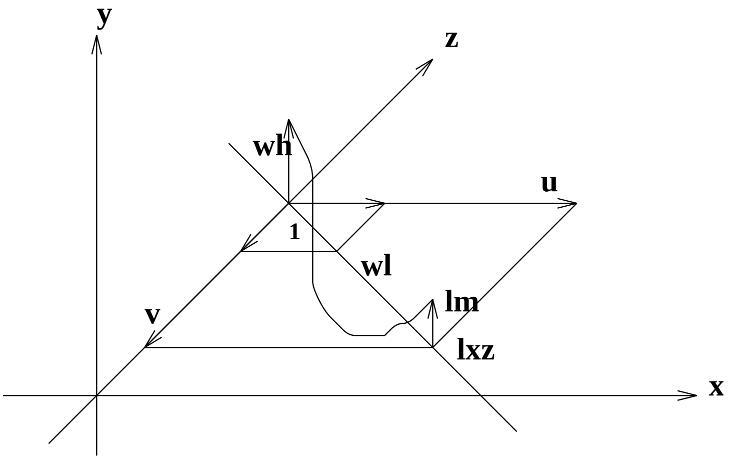

図 2.10. 波紋のバンプマッピング

2.3 レイトレーシング法の高速化技法

レイトレーシングの約 90%以上が，物体との交差判定による計算時間で占められている．レイトレーシングの処理時間が膨大なのは以下の要因による [1]．

- 処理対象となる光線の数が多い．1 画面の光線数は，例えば 1000x1000 とすると 100 万本になる．また，光線が物体と交差した場合，さらに反射光線，透過光線及び影による光線が生成される．
- 1 本の光線は全物体と交差判定をするので，物体数に比例して交差判定の数が増える．この交点計算には，視線や交点に対する座標変換が伴い，計算時間をさらに長くする．
- ほぼすべての処理が浮動小数点演算である．

全体の交差判定の数は光線数×物体数 (2 次光線も含む) となり，この交差判定の数を減らすことが課題となっている．このため，2 つの方法が考えられる．

1. 光線の数を減らす方法
    - 寄与率 :反射光線，透過光線，影光線の生成を物体の輝度に影響を与える度合いに応じて調整する．
    - 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 [2]，空間等分割法がある．
   - 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．

第 3 章　マルチトランスピュータシステム

3.1　トランスピュータのアーキテクチャ

トランスピュータ (TP) とは，イギリスの INMOS 社が開発した並列計算機用プロセッサである．32bitCMOS のマイクロプロセッサ，64bit のオンチップ浮動小数点演算ユニット T805(25MHz) を使用している．チップ上には高速の SRAM 4KB を内蔵している．T805 が 25MHz ピーク時 25MIPS の命令スピードとなり，プロセッサと平行して浮動小数点演算を実行でき，2.8MFLOPS の性能を引き出す．

各々のトランスピュータは，他のトランスピュータと接続する最大 20Mbps の通信速度をもつ 4 つのシリアルリンク (ハードウェアリンク) を備えている．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．並列プログラムを複数のトランスピュータで行う場合と，単一のトランスピュータで行う場合があり，両者ともタスク間の通信はチャンネルを介して行なわれ，前者の場合を外部チャンネル (シリアルリンク)，後者の場合を内部チャンネルという．

3.2　トランスピュータの相互接続網

マルチプロセッサにおいては，プロセッサ間の結合方式が非常に重要である．結合形態すなわち結合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．

現在，研究室に設置されている並列計算機の可能な接続網は，64 台によるトーラス結合とハイパーキューブ結合である．トーラス結合とは，リング結合を発展させたもので，図 3.2に示すような結合網である．ユーザーにとって扱いやすい結合網であり，本研究でもこの結合を利用している．ハイパーキューブ結合 (2 進 n－キューブ，超立方体) 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結合して，プロセッサ間の平均距離を短縮させたものである．2 進 n－キューブは，$N(=2^n)$ 個のプロセッサからなり，各プロセッサには n 本のリンクがある．つまり，あるプロセッサの番号を 2 進数で表すと，そのプロセッサはそれとハミング距離が 1 だけ異なるプロセッサに直接結合している．ハイパーキューブ結合は，リング結合，木結合，格子結合などを包含し，汎用性が高い．これらはすべて物理的接続網である．図 3.1は研究室内のハイパーキューブ結合である． 2 進 5－キューブになると，1 つのプロセッサは他の 5 つのプロセッサと結合しなければならない．1 つのプロセッサは 4 つのシリアルリンクしかもたないため，各頂点に 2 つのプロセッサをおくことによりハイパーキューブ結合を可能にしている．

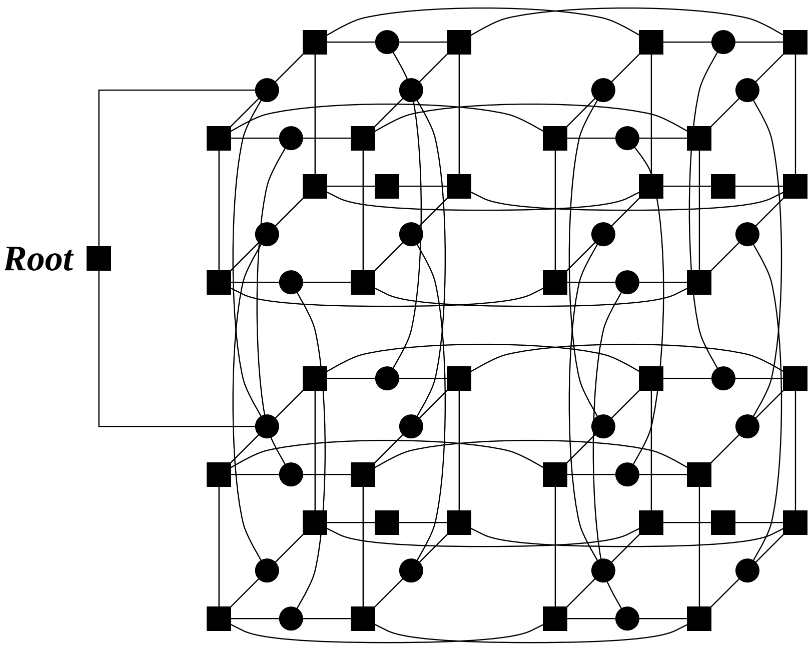

図 3.1. ハイパーキューブ結合 (2 進 5 - キューブ)

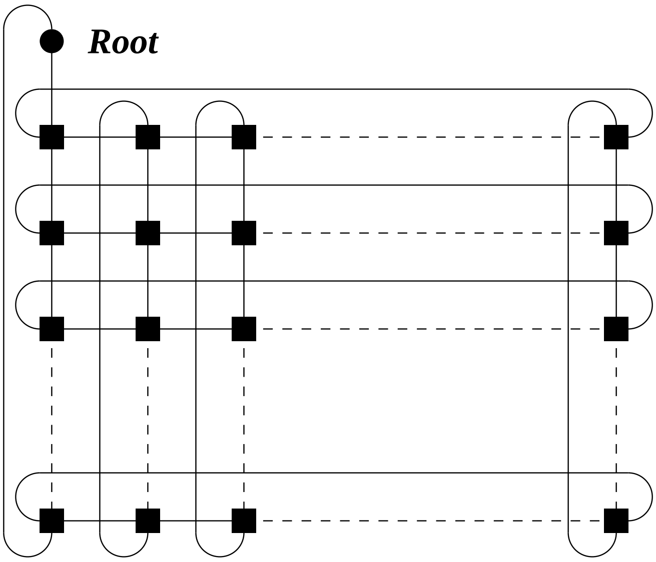

図 3.2. トーラス結合

3.3 Parallel C

Paralell C は，英国スリーエル社が開発したマルチタスクプロセッシングが可能なトランスピュータ用の C コンパイラである． Paralell C の並列処理におけるタスク間通信は，専用のチャンネルを介して行なわれる．1 つのタスクは複数の入出力チャンネルを持つことができ，タスク内の 1 つのチャンネルは他のタスクの 1 つのチャンネルのみと結合する 1 対 1 の関係でなければならない．1 つのチャンネルはあるタスクから他のタスクへの一方向しか許されない．両方向の通信が必要な場合は，その 2 つのタスク間で入出力用のチャンネルを用意する必要がある．タスク間の通信は出力チャンネル要求と入力チャンネルが一致した時のみ通信が成立する．それまで最初に入力または出力要求を出したタスクは通信が成立するまでウエイト状態になる．

言語自体は，並列処理に関する文法以外は一般のＣ言語とほぼ同じであるが，並列処理をするにあたってさらに，Configration File を記述する必要がある．これは，実際に使用するプロセッサやその結合網を決定するものである．このファイルによってソフトウェア的に接続網が決められる．リング結合を用いるとき，このファイルにより物理的接続網のハイパーキューブ結合，トーラス結合のどちらでも可能になる．本研究では，トーラス結合を Configration File によりリング結合として用いている．

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．タスクどうしは独立に処理が行われるため，タスク間のデータ通信はチャネルを介して行われることになる．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なければない時に，非常に有効な手段となる．入出力などを複数のスレッドで実行する場合は，セマフォを使って排他制御しなければならない．図 3.3にタスクとスレッドの関係を示す．1 つのプロセッサ内に 2 つのタスクがあり，さらに各タスクに 2 つのスレッドがあることを示している．

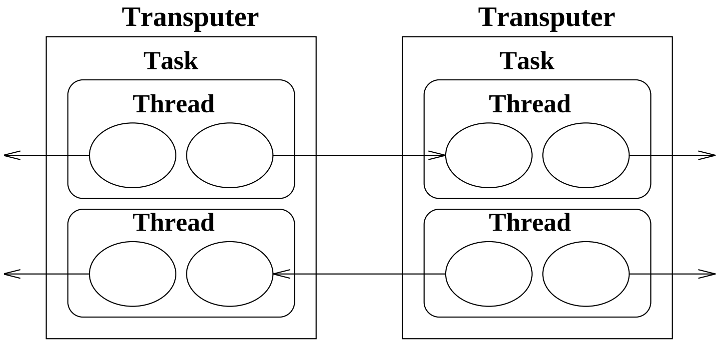

図 3.3. タスクとスレッドの関係

第 4 章　画面分割による高速化

4.1　負荷均衡アルゴリズム

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，プロセッサで並列処理をすることで高速化を図る．従来画面等分割をする実験を行ってきた [11]．画面を等分割すると各領域の負荷に偏りが生じるという問題が残されていた．この問題を解決するために，画面の負荷を求め，どのように分割すべきかを検討した．

画面上のどの辺りにどの物体が描かれるか，どの程度の負荷がかかるのかを知ることができれば，各領域の負荷の偏りを減らすことができる．レイトレーシングを行うとき，3 次元空間における物体の定義が記述された物体記述データを読み込み，それから光線追跡が行われる．そこで，その物体記述データから，物体が描画される位置と負荷を予測し，各領域の負荷が均等になるように分割する方法を考えた．

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，実際にそれが画面に描かれる領域は，物体の回転，移動，視点の位置などにより，知ることは容易ではない．レイトレーシングを行えば，各領域の負荷，位置を正確に知ることができるが，2 度も視線追跡を行うことになる．レイトレーシングを行うことで得られるそのデータを事例として残し，別のレイトレーシングを行うとき，物体の位置など類似性のある事例を参照し，画面を分割する方法も考えられるが，その類似性をどのようにして検索するかといった問題がある．ここでは，レイトレーシングを行う前の負荷の予測を，実際の計算時間に比べてできるだけ小さくし，できる限り正確に負荷の予測をすることを目標としている．この負荷の予測アルゴリズムを負荷均衡アルゴリズム (Load Balancing Algorithm(LBA)) と呼び，それによる分割を負荷均衡分割と呼ぶことにする．

物体記述データのフォーマット及び，物体形状の定義から現段階では球 (楕円も含む) のみを想定している．

4.1.1　画面上の物体の位置

レイトレーシングでは，物体が透視投影法によって画面上に映し出される．よって実際の物体の大きさよりも映し出される物体は小さく，また透視投影法の性質上画面の端ほど物体は歪んでしまう．球の場合，実際に映し出されるのは球の外形，つまり円であるが，その半径も歪みによって一定ではない．レイトレーシングを用いず，できるだけレイトレーシングで描かれた円と等しい円を描く方法を考える．

まず，実際に映し出される円の中心と，x, y方向について中心から円の外周までの距離を

求める．求めたいのは図 4.1の $y1, y2$ である．球の中心座標を (r_x, r_y, r_z) ，半径を rとすると球の方程式は，

$$(x-r_x)^2+(y-r_y)^2+(z-r_z)^2 = r^2 \tag{4.1}$$

となる．図 4.1のように x, y平面上の点 $P(x_1, y_1)$ を通る接線を傾き m, 切片 n として $y = mx+n$ を考える．

$$\begin{cases} (x-r_x)^2+(y-r_y)^2=r^2 \\ y=mx+n \\ \text{接線は点 } P(x_1,y_1) \text{ を通る} \end{cases} \tag{4.2}$$

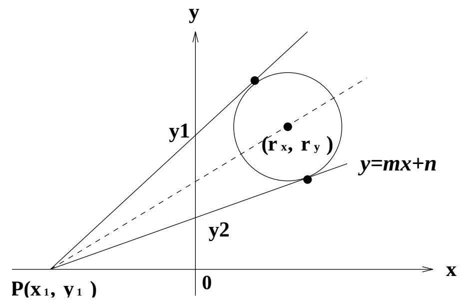

図 4.1. 円の半径を求める

接線 $y = mx+n$ は点 (x_1, y_1) を通るので，式 4.2から $y = m(x-x_1)+y_1 = mx-mx_1+y_1$ と変形できる．それを円の方程式に代入する．

$$\begin{aligned} &(x-r_x)^2+(y-r_y)^2-r^2 \\ =\ &(x^2-2r_xx+r_x^2)+(mx-mx_1+y_1-r_y)^2-r^2 \\ =\ &(1+m^2)x^2-2\{r_x+m(mx_1-y_1+r_y)\}x+(r_x^2-r^2+(mx_1-y_1+r_y)^2)=0 \end{aligned}$$

接することから判別式 Dを用いる

$$\begin{aligned} & b^2-4ac \\ D\ =\ & 4\{r_x+m(mx_1-y_1+r_y)\}^2-4(1+m^2)(r_x^2-r^2+(mx_1-y_1+r_y)^2) \\ =\ & 8r_xx_1m^2+(8r_xr_y-8r_xy_1)m+4r^2-4m^2r_x^2+4m^2r^2-4(mx_1-y_1+r_y)^2 \end{aligned}$$

また

$$\begin{aligned} (mx_1-y_1+r_y)^2 &= (mx_1-(y_1-r_y))^2 \\ &= m^2{x_1}^2-2mx_1(y_1-r_y)+(y_1-r_y)^2 \end{aligned}$$

より

$$= 4(2r_xx_1+r^2-r_x^2-{x_1}^2)m^2+8(r_xr_y-r_xy_1+x_1y_1-x_1r_y)m+4(r^2-(y_1-r_y)^2)$$

従って m を求める一般式が次式で導かれる．この式を解けば m が得られる．

$$(2r_x x_1 + r^2 - r_x^2 - {x_1}^2)m^2 + 2(r_x r_y - r_x y_1 + x_1 y_1 - x_1 r_y)m + (r^2 - (y_1 - r_y)^2) = 0 \quad (4.3)$$

図 4.2は x 軸方向から見たものであるが，視点が P，スクリーンが x, y平面である場合を考える．求める値は実際に映し出される円の y方向の半径である．式 4.3を用いて値を代入すると式 4.5式のようになり，m に関する 2 次方程式を解けば m が得られる．

$$\begin{cases} (z - r_z)^2 + (y - r_y)^2 = r^2 \\ y = mz + n \\ \text{接線は点 } P(v_z, 0) \text{ を通る} \end{cases} \quad (4.4)$$

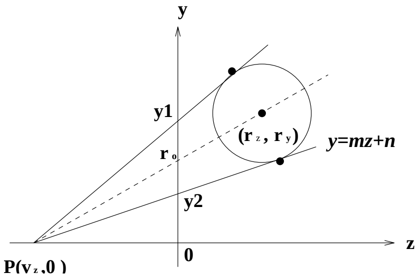

図 4.2. 画面上の円の中心と半径を求める

$$\begin{aligned} &(2r_z v_z + r^2 - r_z^2 - {v_z}^2)m^2 + 2r_y(r_z r_y - r_y v_z)m + r^2 - r_y^2 = \\ &\{v_z(2r_z - v_z) + (r - r_z)(r + r_z)\}m^2 + 2r_y(r_z - v_z)m + (r - r_y)(r + r_y) = 0 \quad (4.5) \end{aligned}$$

式 4.5から m を求め，図の y_1, y_2は $z = 0$ なので，以下のように求まる．

$$y_1 = y_2 = m(z - v_z) + 0 = -mv_z \; \{m = m_1, m_2\} \quad (4.6)$$

同様にスクリーン上の球の中心 (r_o) を求め，$y_1 - r_o, r_o - y_2$ が実際に表示される円の y 方向の半径となる．同様にして x 方向に対しても上述の一般式を用いて x_1, x_2を求める．

4.1.2 負荷の予測

物体のおおよその位置が分かった段階で，画面全体の負荷を予測する．画面を分割する方向に 1 ラインずつ負荷の値を決めていく．レイトレーシングの計算は，90 %以上が交差判定によるものであり，負荷というものはほぼ物体数に比例する．負荷を予測する指標としてこの物体数を用いることにする．物体が何も表示されていないピクセルは，全物体数だけ交差判定が行われるので，1 ピクセルの負荷を全物体数とし，物体が表示されている 1 ピクセルの負荷を以下の方法で求め，画面全体の負荷を求める．

何も表示されていない 1 ピクセルの負荷 = 全物体数

物体は，鏡面体，屈折体，通常物体 (鏡面でもなく屈折でもないもの) に分類する．

- 通常物体
  鏡面，透過屈折しない物体では，まず全物体数だけの交差判定が行われ，次にその交点が影になるかどうか，光源とその点の間に遮る物体がないかを調べる．物体がない場合は，全物体数回交差判定が行われ，ある場合で早いときは 1 回，最悪のとき全物体数回交差判定が行われる．よってその平均をとって全物体数 / 2 回とする．さらに，交差判定以外の処理として色の強度，ハイライト，マッピングなどの負荷も考慮する必要があるが，これに対する負荷の計算方法は検討中である．従って負荷は次式で表される．

$$\text{全物体数} + \frac{\text{全物体数}}{2}$$

- 鏡面物体
  鏡面物体は，通常物体による負荷と，その点に映し出される物体の検索の負荷がかかる．映し出される物体の検索に，通常物体と同様の処理が行われるため全物体数回の交差判定が行われる．さらに映り込んだ物体がさらに鏡面であると，さらに全物体数回交差判定が行われる．図 4.3のように，この映り込みによる交差判定は最大でも 5 回，最小で 2 回となっている．負荷としてはその平均をとり，全物体数 $\times 3.5$ とする．そして映り込む物体があれば，そこで影による交差判定が行われる．

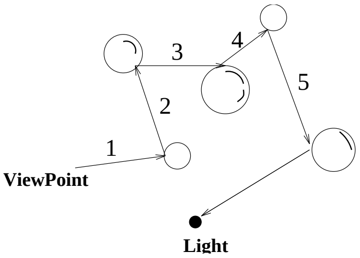

図 4.3. 鏡面による映り込み

以上のことから，鏡面に対する負荷は次式で表される．

$$\text{全物体数} \times 3.5 + \frac{\text{全物体数}}{2}$$

- 透過屈折物体
  屈折する物体は，交点において反射光線と透過光線の 2 つ光線を生み出し，さらに透過光線はその物体から出るときに，内部への反射光線と外へ出る透過光線を生み出す．それらを考えると少なくとも交差判定が 5 回繰り返される．ここでは，さらに調整用として 1.5 を追加している．

$$\text{全物体数} \times 5 + \frac{\text{全物体数}}{2} + 1.5$$

以上の方法で 1 ライン上の各ピクセルの負荷をだし，1 ライン全体の負荷をだす．図 4.4 にそれらの示す．拡大された部分は，1 ライン上の各ピクセルを表している．斜線の部分は物体 (鏡面) のあるピクセルで，空白は物体のないピクセルである．上述の方法に従って図の 1 ラインの負荷を求めると，975 となる．このようにして，全ラインの負荷をだし，画面全体の負荷をだす．

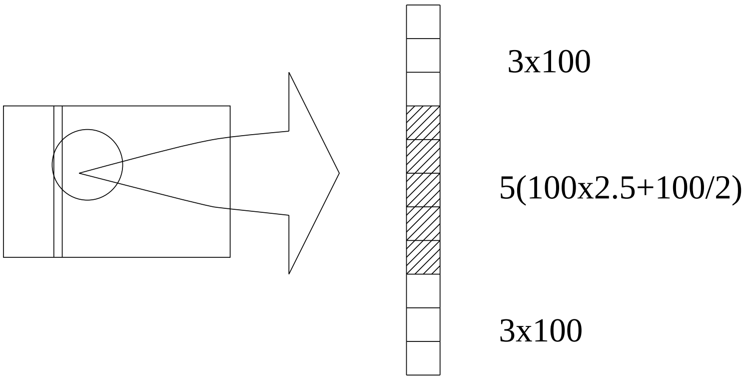

図 4.4. 各ラインの負荷の計算

4.1.3 負荷均衡分割

画面全体の負荷をもとに画面を分割する．分割方法として，まず全体の負荷を使用する全プロセッサ数で割り，平均の負荷をだす．分割後の各ブロックの負荷がその平均値に近くなるようにラインを割り当てていく．分割の種類は，メッシュ分割 (Mesh)，縦分割 (Column)，横分割 (Row) の 3 通りであり，メッシュの場合は，縦方向に分割した後，同様に横方向に分割している．分割の様子を図 4.5に示す．

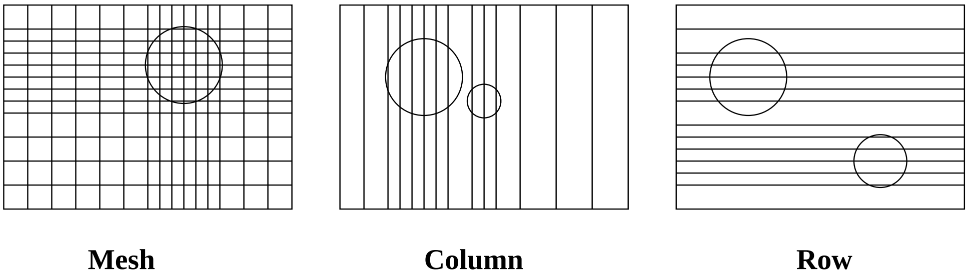

図 4.5. 分割の種類

各分割方法についてそれぞれ問題がある．縦 / 横分割では，各ブロックに割り当てるとき，分割方向に 1 ラインずつ割り当てていきそのラインの負荷の合計が最も平均値に近く

なったところ，正確にはその平均値を越えないところでラインの割り当てをやめる．そのため最後に割り当てるラインの負荷が非常に大きい場合，平均値とそのブロックの負荷の値との差が大きくなる．またメッシュ分割では，縦方向に分割後，横方向に分割を行うが，縦と横の分割に依存性はなく，どちらから分割しても同じ結果になる．そのため，各ブロックの負荷に偏りが生じる．それは図4.5で確認できる．各ブロックの負荷が均等になれば，何も物体が表示されないブロックの大きさは等しくなる．しかし，図4.5のように何も物体が表示されていないブロックの大きさに違いがあることが分かる．

これは，メッシュに分割するとき縦，横分割で依存性を持たせなかったためである．そこで，負荷の偏りを減少させるために次の分割方法を考えた．

1. 縦方向に今までの方法で分割する．この時点である程度の負荷の均衡がされている．
2. 各ブロックを横方向の分割数で分割する．このとき，1. で得られたブロックの負荷を横の分割数で割り，平均値をだす．この平均値に近くなるように各ブロックを横に分割する．

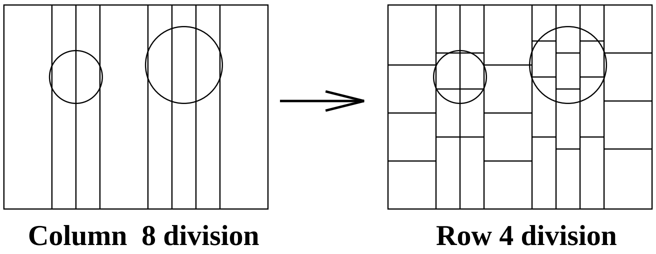

図4.6. メッシュ分割の改良版

この方法により，縦分割の段階で負荷の偏りが生じても，より負荷の均衡のとれた領域に分割できると思われる．

4.2 並列化

負荷均衡アルゴリズムによって分割されたブロックを，各プロセッサで計算させ並列化を行う．接続網はリング接続およびトーラス接続を使用する．図4.10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，与えられたブロックの計算を行う．リング接続は図4.10 (左) に示すような構成になる．データがTPからTP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．ライン単位ではなくブロック単位で処理を行っているため，非常に多くのメモリを要する．画面は 640×400 で，

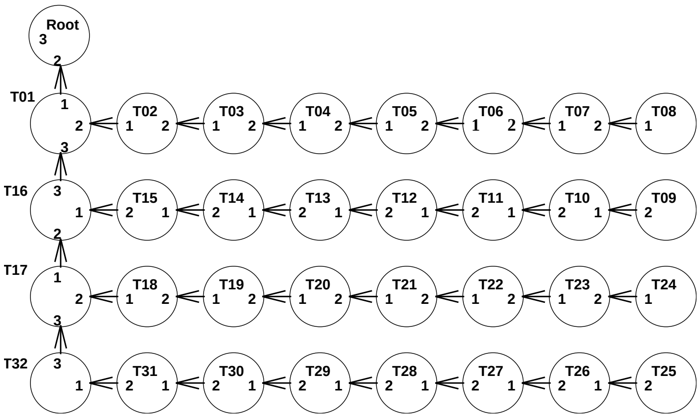

図 4.7. 回収方法 1(最大通信距離 11)

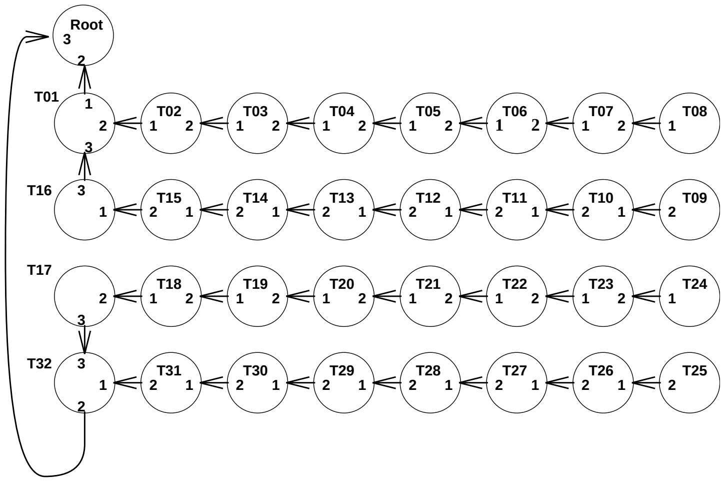

図 4.8. 回収方法 2(最大通信距離 9)

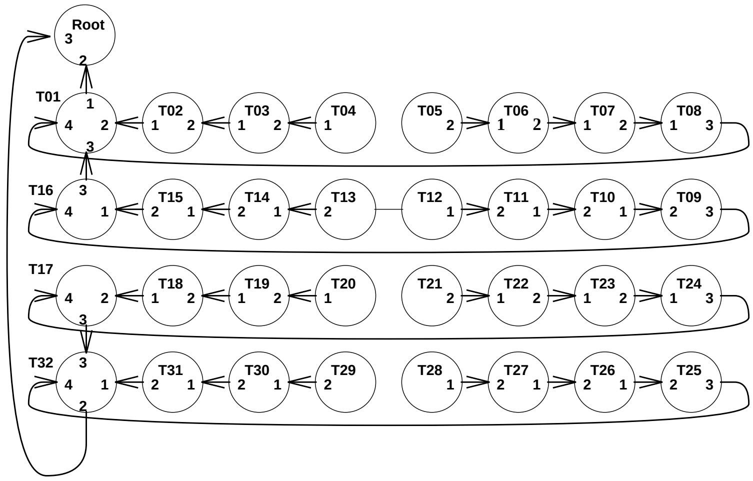

図 4.9. 回収方法 3(最大通信距離 6)

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，全データ領域として $640 \times 400 \times 4 = 1MB$ 必要となる．各プロセッサには分割されたブロックの計算結果の格納領域が必要である．64 等分割で 16KB は必要であり，4 等分割の時には 256K バイトも要する．このシステムでは 1 度そのデータをメモリに入れ，再び取り出して出力しなければならないので，送られてきたデータを通過させるためのメモリも消費する．

等分割では，各ブロックのデータの大きさは同じなので，メモリ消費量も最も経済的に押さえられるが，負荷均衡分割の場合，負荷を調整するために各ブロックのサイズが異なる．そのためデータを通過させるためには，各プロセッサで，分割された最大ブロックのサイズのメモリを動的に取る必要がある．

通信によるオーバヘッドを極力なくすために，トーラス接続を用いる．データをルートに戻す方法として 3 通りを考えた．1 つめは，図 4.7に示すような最大通信距離が 11(32 台)，15(64 台) となる接続である．ルートから縦方向に接続されるプロセッサは，図 4.11の左図のように通信部のスレッドが 1 つ増える．2 つめは，図 4.8に示すように最大通信距離が 9(32 台), 11(64 台) となる．3 つめは，最も通信距離を短くしたものであり，最大通信距離は 6(32),8(64) となる．同様にルートから縦方向に接続されるプロセッサは，図 4.11の右図に示すようにさらに通信部のスレッドが増え，計算部 1 つと通信部 3 つからなる．図は 32 台によるトーラス接続のものであり，64 台の場合も同様となる．

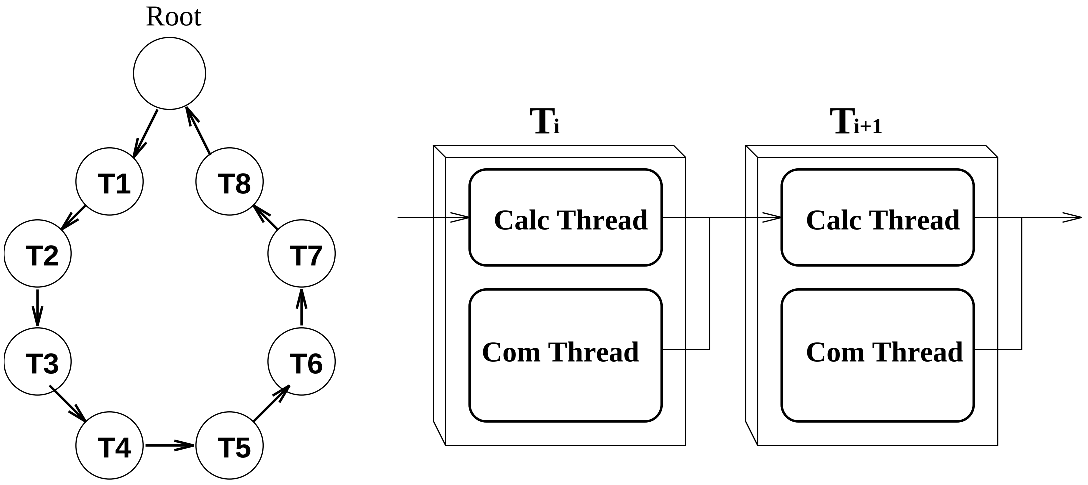

図 4.10. リング接続 (左) と各プロセッサ内のスレッドの様子 (右)

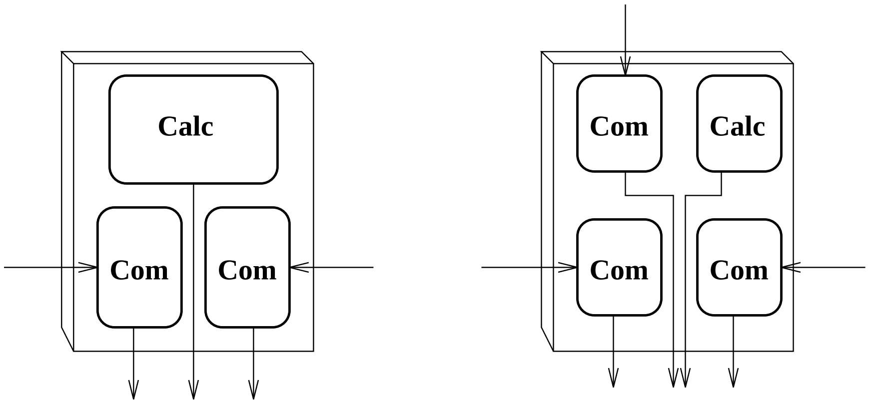

図 4.11. 回収方法 1,2,3 のスレッド (1,2(左)，3(右))

第 5 章　空間分割による高速化

5.1　空間分割

空間分割 (Space Subdivision) とは，その名の通り空間を分割するアルゴリズムである．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．まず，ある空間に物体を配置し，適当な部分空間に分割する．空間の分割方法も様々あり，オクトリ (Octree) による適応型空間分割，ボクセル (voxel) とよばれる一定の大きさに分割する空間等分割法などがある．分割後，その各空間に属する物体をリストとして得る．ここまでが前処理である．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，視線ごとに各物体と交差するかを計算してきた．例えば，物体が100個あった場合，その視線と明らかに交差しなくても，100個の物体すべてについて交差判定を行ってきた．しかし，空間分割の場合，視線がある空間に入ると，その空間に属する物体に対してのみ交差判定を行う．つまり，全体の空間に物体が100個あっても，今扱っているその部分空間に物体が1つもない場合は全く交差判定を行わないのである．このことによって高速化が図れる．

この空間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．もし，交点が存在しない，あるいはその空間に物体が1つも属していない場合は，その視線が貫く次の空間に視線を移し，再び上に述べたような判定を行う．反射の場合も同じで，反射する方向にある空間で判定を行う．その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．

5.2　マルチプロセッサによる空間分割

この空間分割をマルチプロセッサに適用し，並列処理を行う．本研究では空間等分割を行う．物体の存在する3次元空間を同じ大きさの部分空間であるボクセルに分割し，それを各プロセッサに割り当て，プロセッサ間で光線情報パケットをやりとりすることで画像を生成する．空間を設定し，図5.1のように最大で $4 \times 4 \times 4$ の分割を行う．分割された各ボクセルには，ボクセル番号 (ijk) が割り当てられており，x, y, z 軸とは別に i, j, k なる座標系を定め，ボクセル番号を決めている．

5.2.1　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．等分割による空間分割を行うので 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

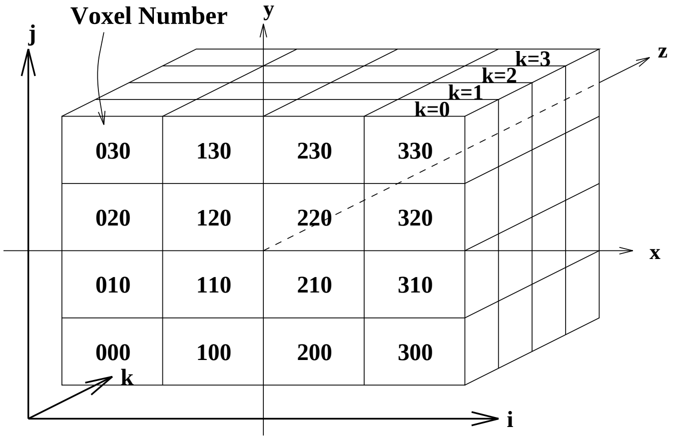

図 5.1. 空間分割 (Space Subdivision)

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 2DDDA，それを拡張した 3 次元版を 3DDDA と呼び，DDA によってできた線分をデジタル直線と呼ぶ．

ここで必要とするデジタル直線は，ディスプレイに直線を表示する場合に使用する通常のデジタル直線とは少し異なる．すなわち，空間分割用のデジタル直線は，元の (実数) 直線が通過するすべてのボクセル (2 次元であればピクセル) 列として得られなければならないのに対して，ディスプレイ用のデジタル直線は，ある座標軸 (配列のあるインデックス) について同じ座標をもつピクセルは，唯一存在するというものが用いられる [3]．図 5.2 にその 2 つの直線を示す．

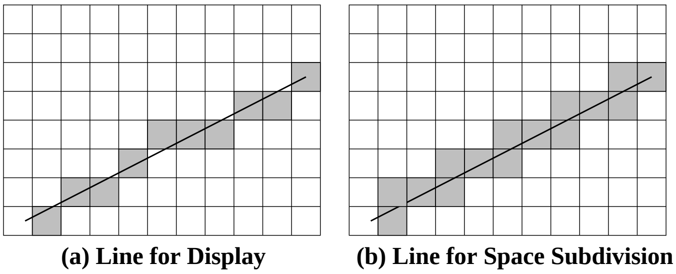

図 5.2. デジタル直線

実際の 3DDDA を図示するのは困難なので，図 5.3は 2 次元的にしてある．3DDDA は 2DDDA に z軸方向に拡張したものと考えればよい．

まず図 5.3に示す各変数を以下のように決める．Vx, Vy, Vzは 1 つのボクセルの大きさを表している

- (i, j, k) :現在のボクセルの x, y, z方向のインデックス．

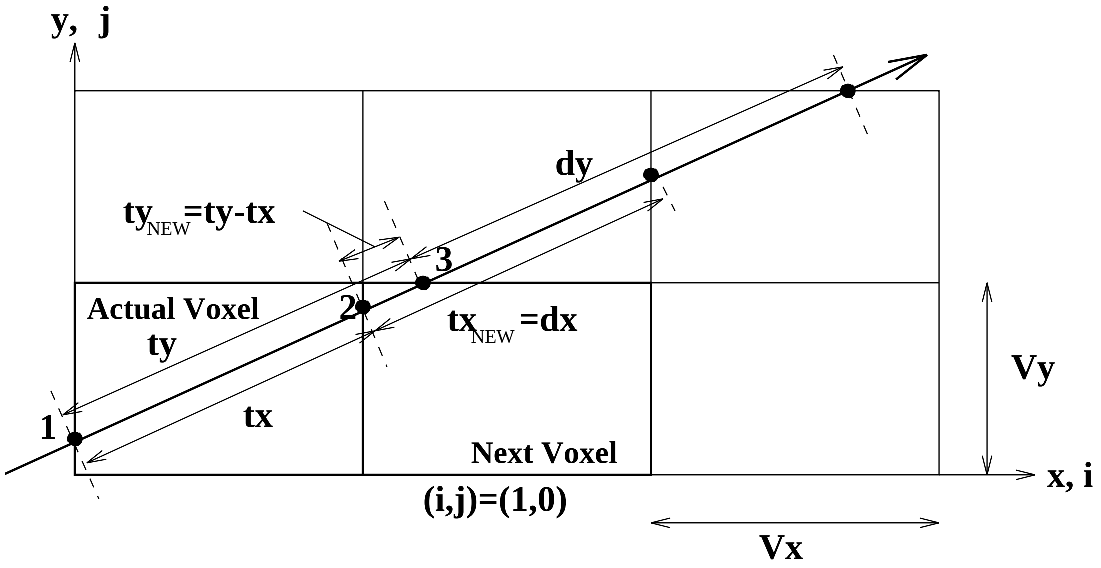

図 5.3. 3DDDA の様子 (2 次元)

- (dx, dy, dz) ：直線が x, y, z方向それぞれについて，1 ボクセル分だけ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．

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 5.3 のように，$tx < ty$なので，$(i+1, j) = (1, 0)$ を通らなければならないことが分かる．次に，点 2 において新たに t_xと t_yを比較し，$ty_{NEW} < tx_{NEW}$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，インデックスの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．この例では光線が $(0, 0) \Rightarrow (1, 0) \Rightarrow (1, 1) \Rightarrow (2, 1)$ を通る．

5.2.2 3 次元空間における直線の分類

3DDDA により，光線の進むべき方向とボクセルを決めることができる．それには，前述の tx, ty, tz，dx, dy, dzと光線の傾きを利用して求める必要がある．2 次元 (2DDDA) の場合では，視覚的に分かりやすく，比較的容易に出せるが，3 次元の場合そう簡単には出せない．そこで，傾きも含めてこれらの値を求めやすくするために 3 次元における直線の分類を行った．

まず，点 $P(p_x, p_y, p_z)$, 点 $Q(q_x, q_y, q_z)$ を通る直線を考える．$x - y$平面における傾きを m_{yx}

,$y-z$平面の傾きを m_{yz} ,$z-x$ 平面の傾きを m_{zx}とすると

$$\begin{array}{ll} \delta_x = q_x - p_x & m_{yx} = \delta_y/\delta_x \\ \delta_y = q_y - p_y & m_{yz} = \delta_y/\delta_z \\ \delta_z = q_z - p_z & m_{zx} = \delta_z/\delta_x \end{array}$$

となる．直線はすべて，図 5.4に示すように 7 つのいずれかに分類される．1,2,3 は各軸に平行な直線であり，4 は yz平面に平行な，5 は xy 平面に平行な，6 は xz平面に平行な直線である．それ以外の直線が 7 で表される．

表 5.1. 直線の分類

分類番号	直線の傾きの状態
1	$\delta_x = 0$ ***and*** $\delta_z = 0$
2	$\delta_x = 0$ ***and*** $\delta_y = 0$
3	$\delta_z = 0$ ***and*** $\delta_y = 0$
4	$\delta_x = 0$ (但し ,$1, 2$ を除く)
5	$\delta_z = 0$ (但し ,$1, 3$ を除く)
6	$\delta_y = 0$ (但し ,$2, 3$ を除く)
7	$(\delta_x, \delta_y, \delta_z) \neq 0$

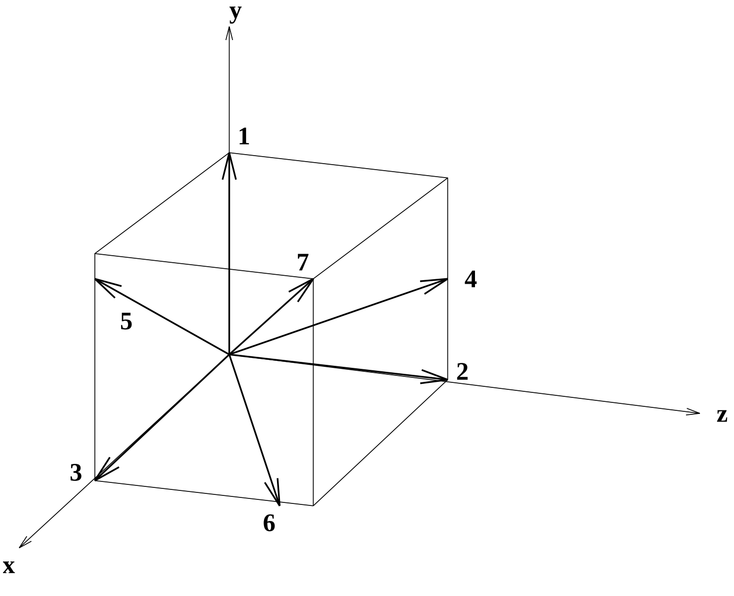

図 5.4. 3 次元空間での直線の分類

5.2.3 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) とは，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．現在，パケットは表 5.2のように定義している．

表 5.2. パケットの定義

変数名	バイト数 (データ型)	内容
$Source$	$4(int)$	レイ発生ボクセル番号 (i, j, k)
$Destination$	$4(int)$	行先ボクセル番号 (i, j, k)
$pixel\_xy$	$4(int)$	レイ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
Ir, Ig, Ib	$4(float) \times 3$	計算された交点の輝度

図 5.5のように，変数 $kind$ の $0, 1bit$ と $4, 5bit$ を使って種類を分けている．パケットの種類は Terminate(終了用)， RayPacket(レイパケット)，ReturnPacket(リターンパケット)，ShadowRay (影を検索するためのレイパケット)，ReflectRay(鏡面による 2 次レイの用のパケット) の 6 種類のパケットがある．

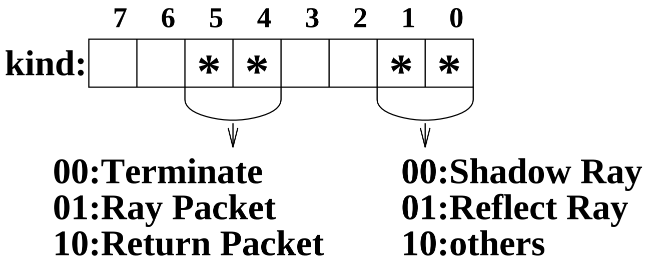

図 5.5. パケットの種類

5.2.4　各プロセッサのタスクとスレッド

図5.6に示すように，各プロセッサは，メインタスクと通信タスクからなり，さらに通信タスクには，各ポートからの入出力用のスレッド(IN,OUT)が10個存在し，入出力の監視を行っている．

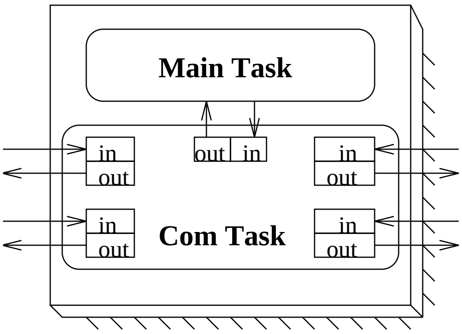

図5.6. 各プロセッサのタスクとスレッド

図5.7にスレッドで行われる処理の流れ図を示す．IN スレッドでは，入力データがあるかどうか調べ，もしあればパケットの種類により処理が分かれる．OUT スレッドでは，出力データがあれば出力し，なければ待ち続ける．関数レイパケットでは，目的ボクセルに到着したかどうか判定し，到着したならメインタスクへパケットを出力し光線追跡を行う．さもなければ，ボクセル間移動処理をし出力する．関数リターンパケットは，視線発生源に到着したかどうか調べ，到着したならば関数$Arrival$で結果データをメモリに格納する．さもなければ，ボクセル間移動アルゴリズムで出力ポートを決定し出力する．関数ターミネイトは，すべてのプロセッサの処理が終了したときに送られてくるパケットによって起動し，すべてのスレッドを終了させる．メインタスクでは，そのレイパケットの情報に基づいて光線追跡を行う．

5.2.5　ボクセル間移動アルゴリズム

空間分割では，各プロセッサ間で光線(パケット)が飛び交う．トランスピュータシステムでは，あるプロセッサにデータを送るとき，データの経路を指定して送らなければならない．そこで，3DDDA により次のボクセル(i,j,k)を指定することで，その番号のプロセッサに送る移動アルゴリズムを考えた．

図5.1に示したように，各プロセッサにはボクセル番号が割り当てられている．番号はインデックス(i,j,k)からとられ，$(i,j,k)=(1,0,0)=(ijk)=(100)$のように認識される．実際に64個のプロセッサを使用するとき時の，各プロセッサの番号割当の様子を図5.12に示す．ボクセル間の移動は，移動先のkによって距離が異なるので，kで場合分けを行った．

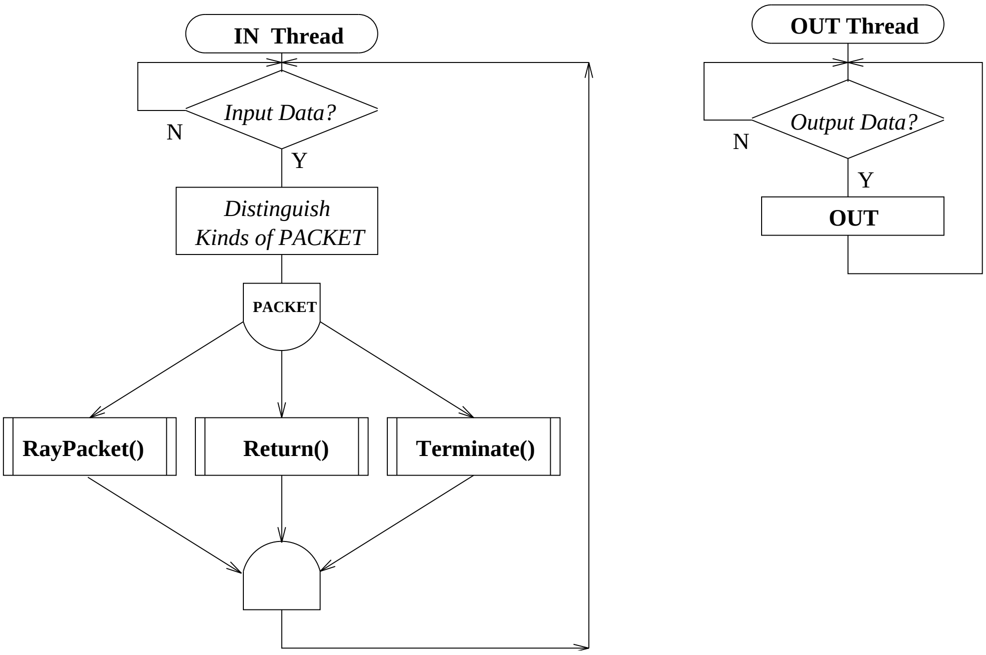

図 5.7. IN スレッドと OUT スレッドの流れ図

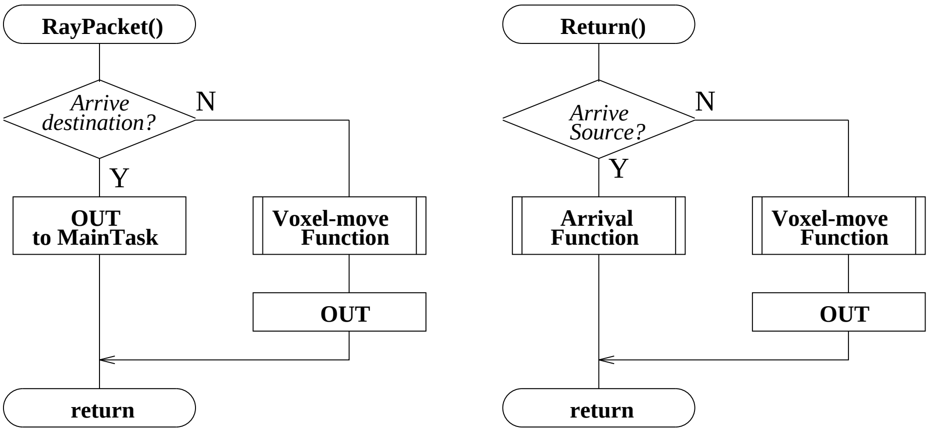

図 5.8. レイパケットとリターンパケットの流れ図

1. 同じk内での移動で，隣接する4つのプロセッサにあるいは斜め方向に移動する場合．

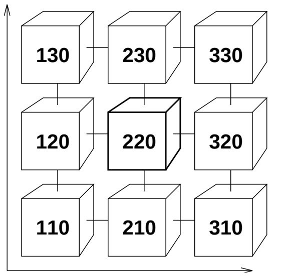

図 5.9. 同じk内で移動

- 220 から 320 へ移動する場合
  $(3-2, 2-2, 0-0) = (1, 0, 0)\{i$ 方向に 1 移動 $\}$
- 220 から 130 へ移動する場合
  $(1-2, 3-2, 0-0) = (-1, 1, 0)\{i$ 方向に -1 ,j方向に 1 移動 $\}$

2. 異なるkへの移動で，z軸方向に移動する場合．

   (a) 単にz軸方向に移動する場合 (i, jは同じでkが変わる場合 (図 5.10))
   $k=0 \Leftrightarrow k=1$ は，パケットが $k=0$ のボクセルから $k=1$ のボクセルへ，あるいはその逆へ移動する場合をさしている．以下に示すkに関する移動を行った後に，上述 1. の移動を行う．

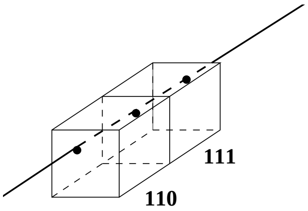

図 5.10. 異なるkへ移動

(i) $k=0 \Leftrightarrow k=1$: i 方向に ± 4 移動する．

(ii) $k=2 \Leftrightarrow k=3$: i 方向に ∓ 4 移動する．

(iii) $k=0 \Leftrightarrow k=3$: j方向に ± 4 移動する．

(iv) $k=1 \Leftrightarrow k=2$: j方向に ± 4 移動する．

(v) $k=0 \Rightarrow k=2$: i 方向に $+4$, j方向に $+4$ 移動する
$k=2 \Rightarrow k=0$: i 方向に -4, j方向に -4 移動する

(vi) $k=1 \Rightarrow k=3$: i 方向に -4, j方向に $+4$ 移動する
$k=3 \Rightarrow k=1$: i 方向に $+4$, j方向に -4 移動する

(b) それ以外の場合つまり，ボクセルの境界を通る場合である．図 5.11に示す 3 通りであり，1. の方法で (i, j) 方向に移動してから，2. の (a) で z軸に移動をすればよい．

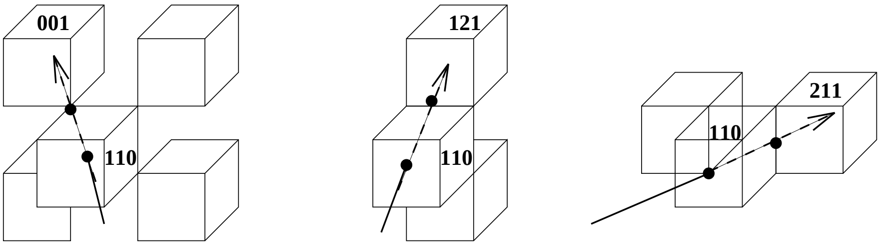

図 5.11. 境界を通り，異なる kに移動

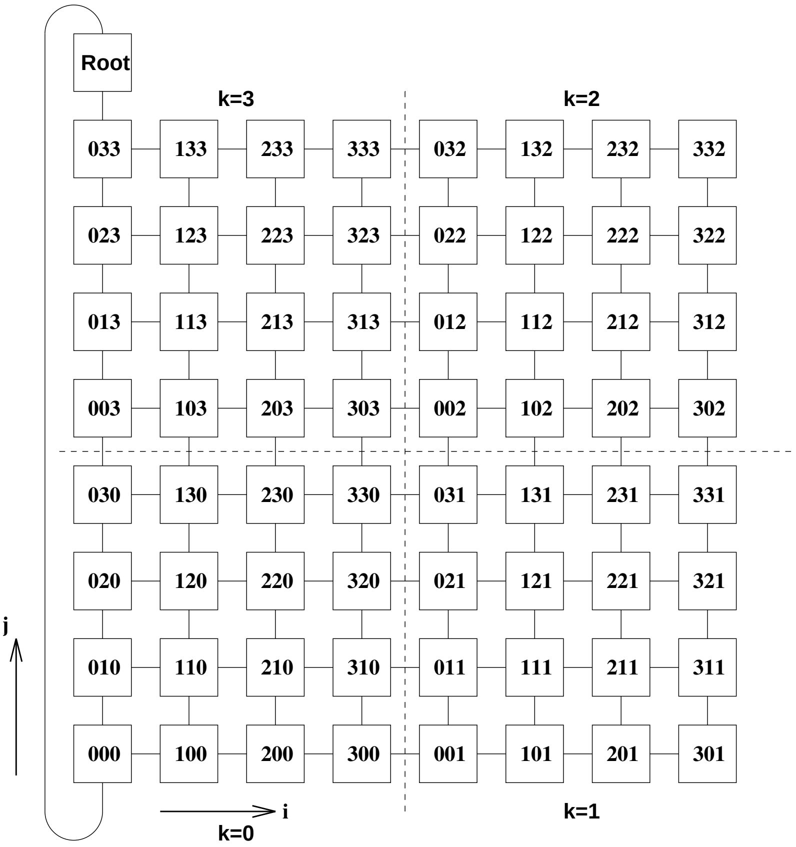

図 5.12. 各プロセッサとボクセルの位置関係

5.2.6 リターンパケットの回収

以上のアルゴリズムを基に実際にパケットのやりとりが行われるわけであるが，それらが空間内でどのように振る舞うかを考えてみる．リターンパケットの回収方法として，図 5.13，5.14の 2 通りを考えた．実線はレイパケット，点線はリターンパケットを示している．物体 B は鏡面反射物体とする．番号はボクセルの番号であり，y軸正の方向から負の方向へ見ている図である．

図 5.13は，ボクセル番号 200 を通る 1 視線について示している．レイパケット (1) は，200 で物体がないので 201 へと移動する．ここでは交差判定の結果．物体 A と交差することが判明する．そこで，その交点における輝度を求め，その点が影になるかどうかを調べるために光源方向に，レイパケット (4) を送り出す．さらにその交点に映る物体を求めるために，反射方向にレイパケット (2) を送り出す．それは 202 を通り， 102 のボクセルに入る．そこで物体 B と交差することが分かりその交点の輝度を求めるために，光源へ向けてレイパケット (3) を出す．次にこれらの結果であるリターンパケットを回収する．交点 a からリターンパケット (1’) を送り出すためには，リターンパケット (2’) と (4’) を待つ必要がある，同様に，交点 b からリターンパケット (2’) に送り出すにはリターンパケット (3’) を待たなければならない．すべてのリターンパケットが集められ，リターンパケット (1’) としてプロセッサ 200 に戻される．このようにして各視線のレイパケットの授受が行われる．

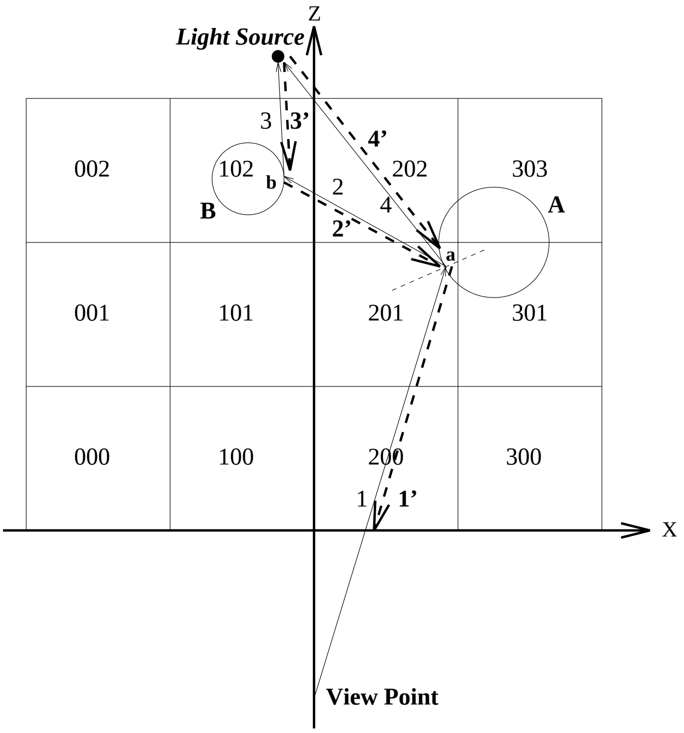

図 5.13. リターンパケット回収方法 1

図 5.14は先ほど少し異なる．レイパケットが各交点及び，光源において，リターンパケットが即座に送り出される．つまり，交点 a では，物体 B と光源に向けてレイパケットを出し，それらのリターンパケットを待たずに交点の輝度情報をもったリターンパケット (1’) を 200 に送り出す．同様に交点 b からもリターンパケット (2’) を送り出し，光源からは物体 A，B の影に関する情報がリターンパケット (3’)(4’) として 200 に送り出される．

前者の方法では，交点ごとにリターンパケットを待つ必要があり，待っている間，別の処理が行われないという事態が考えられる．最悪の場合，デッドロックが起こりうる．また各交点で，リターンパケットを保持するためのメモリを要する．こういった欠点がある反面，逐次に行う一般的なレイトレーシングの光線追跡と同じ過程を行っているので正確にリターンパケットの情報を求めることができる．後者の方法では，各交点でリターンパケットをまつ必要がなく，リターンパケット用のメモリも必要ない．しかし，図からもわかるように，200 のプロセッサで各リターンパケットの情報を格納するメモリを用意する必要がある．さらに，戻ってきた複数のリターンパケットをどのように識別し，各リターンパケットの内容を具体的にどのようなものにするかが問題となる．戻ってきた複数の情報を正しく 1 つのピクセルの情報に復元しなければならない．言い換えれば，復元できるようなリターンパケットを戻さなければならない．

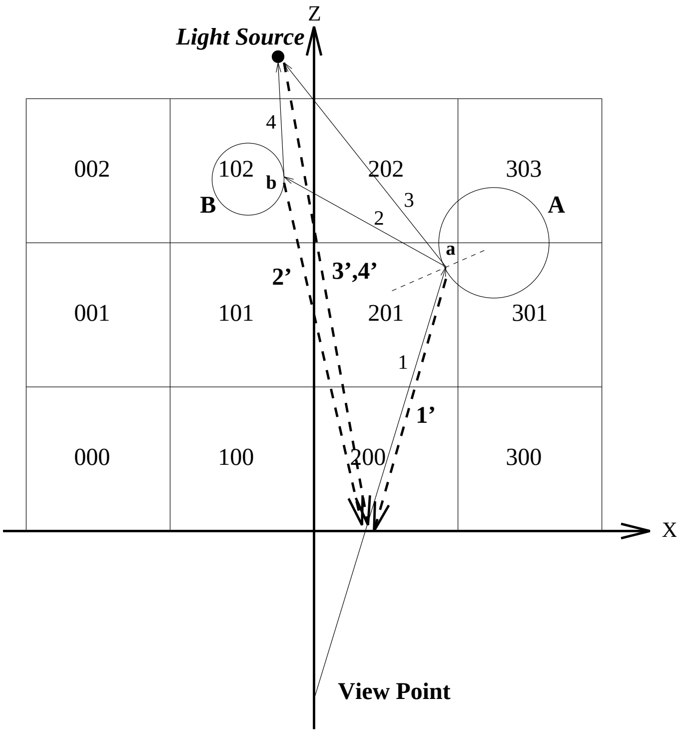

図 5.14. リターンパケット回収方法 2

どちらの方法も一長一短であるが，本研究では後者の方を採用することにし，話を進める．さきほども述べたように，リターンパケットの内容をどのようにするかが問題となる．表面輝度 I(r 成分) というものは，次式のようになる．

$$\begin{aligned}
I &= k_{dr}I_{pr}(\boldsymbol{L}\cdot\boldsymbol{N}) + k_{sr}I_{pr}(\boldsymbol{R}\cdot\boldsymbol{V})^n \\
&= \text{拡散反射光} + \text{鏡面反射光} \\
&= I_{pr}(k_{dr}(\boldsymbol{L}\cdot\boldsymbol{N}) + k_{sr}(\boldsymbol{R}\cdot\boldsymbol{V})^n) \qquad (5.1) \\
&= \text{影に依存する輝度} \qquad (5.2) \\
I_{pr} &= i_{pr}S_r \\
&= \text{直接光} \times \text{影かどうかのスイッチ}
\end{aligned}$$

表面輝度は，式を変形することで式 5.1のように，影に依存する輝度になることが分かる．I_{pr}は影の関数 S_rにより，その表面が影になるならば 0 となり，それ以外は正の値をもつことになる．このようにすることで図 5.14のリターンパケット 1,2 は，式 5.1の括弧の内容を持たせ，3,4 は I_{pr}の値を持たすことにする．

図 5.14の場合，この視点から出た視線に関するリターンパケットは 4 つである．しかし，どこかで光線が派生して，リターンパケットがいくつ帰ってくるかを知ることは不可能である．そこで生成されたリターンパケットの合計を知るために，光線に重みをつけることにした．図 5.15のように，最初のレイパケット (1) に重み 1 を与える．次にそれが交差し，2 次光線を発する時点で，重みを分散する．この図の場合，a で交差する時点で 3 つの光線に分かれるのがわかる．つまり，リターンパケット (1')，レイパケット (2)，レイパケット (4) の 3 つなので，各パケットの重みとして 1/3 を与える．同様に，レイパケット (2) は交点 b において光線が 2 つ分かれるので，各パケットに，その重み 1/3 を 2 で割った 1/6 を与える．光源へ向かったそれぞれのレイパケットは，そのままの重みでリターンパケットが帰される．最終的にこれらのパケットの重みを合計すれば，最初のレイパケットの重みと等しくなり，すべてのリターンパケットが戻ってきたかを知ることができる．この場合は次のようになる．

$$\frac{1}{3}(1') + \frac{1}{6}(2') + \frac{1}{6}(3') + \frac{1}{6}(4') = 1(\text{最初のパケットの重み})$$

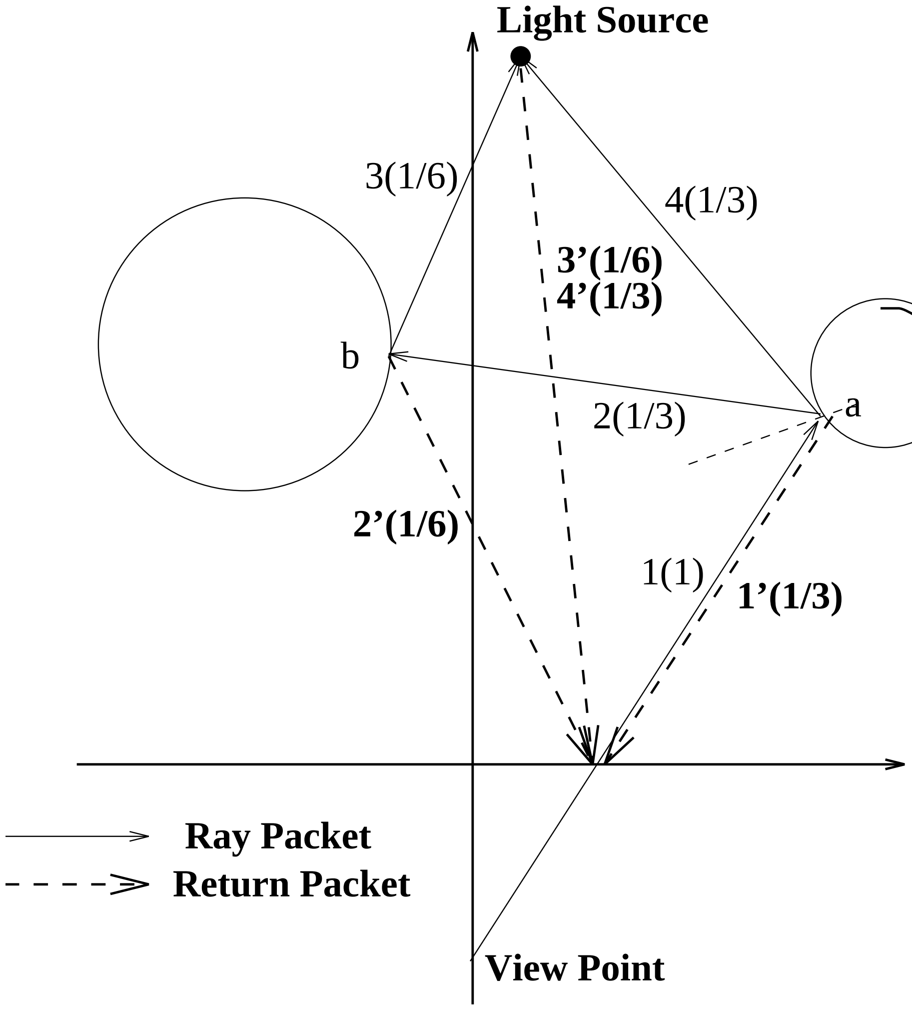

図 5.15. パケットの重み

第 6 章　実験と評価

6.1　実験方法

画面分割では，画面等分割と負荷均衡分割の実行時間と相対速度を求める．リング接続を用いて，トランスピュータ 1 台及び，4 台から 64 台までの偶数台で実験を行う．さらに，トーラス接続によるデータ回収時間短縮の実験を行う．これは，物理的な接続網の制約から 32 台と 64 台のみで行う．実験に用いたデータは，表 6.1に示す 7 データを用いた． 負荷分散アルゴリズムは球のみに対応しているので物体形状はすべて球体としている．また相対速度とは，式 6.1に示すように，1 台の実行時間に対する速度向上比のことである．

表 6.1. 画像データの内容

データ	内容
1	通常物体 100 個を規則的に画面全体に配置
2	通常物体 200 個を規則的に画面全体に配置
3	通常物体と鏡面物体が交互に縦方向に並ぶ．合計で 100 個の物体
4	通常物体と鏡面物体が交互に横方向に並ぶ．合計で 100 個の物体
5	通常物体 13 個，透過屈折物体 13，鏡面物体 (床) を配置
6	通常物体 (上下平面)，透過屈折物体 1 個，鏡面物体を 18 個配置
7	通常物体 2 個，鏡面物体 3 個，透過屈折物体 1 個，床 (鏡面 + バンプマップ)

空間分割は，3DDDA を用いた 1 台による空間分割と空間分割を行わない通常の方法と比べ，その効果を実験する．物体空間を $2 \times 2 \times 1$ の空間とし，物体を 25,50,75,100 個を配置した場合の実行時間を求める．

$$相対速度 = \frac{1\,台のトランスピュータによる実行時間}{n\,台のトランスピュータによる実行時間} \tag{6.1}$$

6.2　実験結果と考察

図 6.1 ~ 6.7に，画面分割 (リング接続) による各画像の相対速度グラフを示す．横軸はトランスピュータの台数で，縦軸は相対速度を表している．各グラフは，等分割による結果 (Col,Row,Mesh) と負荷均衡分割による結果 (LBA-Col,LBA-Row,LBA-Mesh) の相対速度を示している．また，各データの画像を図 6.9 ~ 6.15に示す．

6.2.1　等分割と負荷均衡分割の結果

データ 1 は，画面全体に球が配置されているので，全体的にリニアな結果が得られている．等分割による結果では，縦，メッシュ，横分割の順に結果がよくなっている．横分割が 3 つの中で最もよく縦分割が悪い理由は，この画像の負荷の偏りが横方向に小さく，縦方向に大きいでからある．画像 1 を見ても分かるように，画像左右の端に物体のない領域がある．縦分割した場合，必ず，物体の含まれるブロックと含まれないブロックができ，等分割のため物体のあるブロックの計算方が時間がかかってしまう．しかし，横分割の場合，分割されるブロックをみると，どの領域も含まれる物体数が同じで，負荷が均等になっているので結果がよくなっている．負荷均衡分割はすべて，等分割よりもよい結果となっており，負荷の予測がより正確に行われていると考えられる．

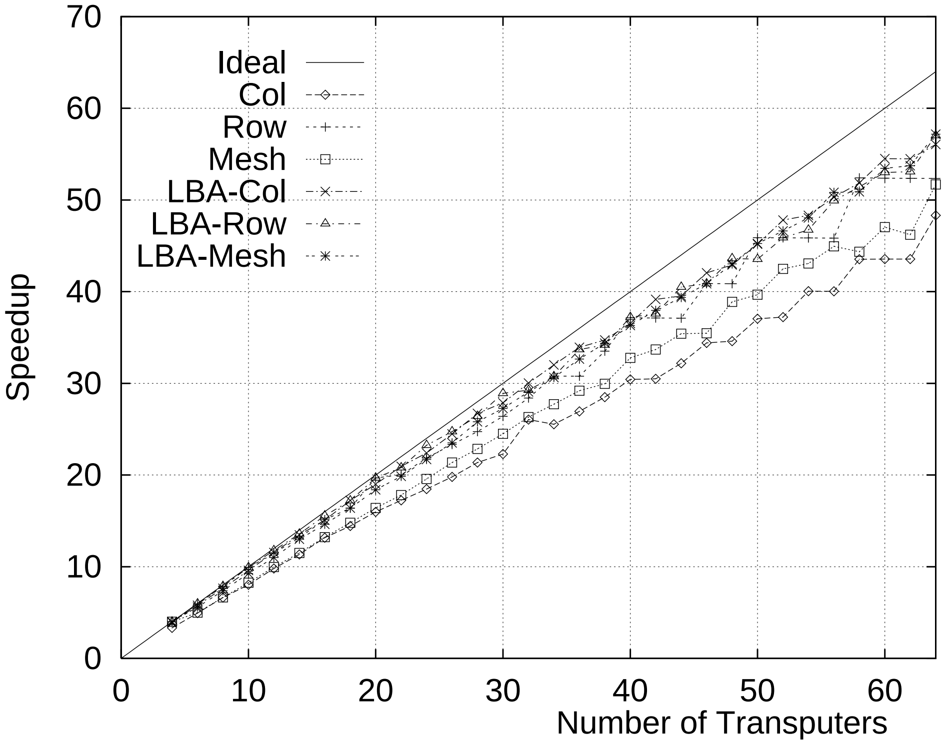

図 6.1. データ 1 による結果

データ 2 は，物体数が 200 個となったため，全実行時間にしめる通信時間の割合が減り，全体的に相対速度も上がり，データ 1 よりさらにリニアな結果となっている．等分割による結果は，データ 1 の場合とは逆で，画像の上下に物体のない領域があるため，横等分割の結果が悪くなっている．それ以外は良好な結果であり，7 データのうち最も良い結果となっている．

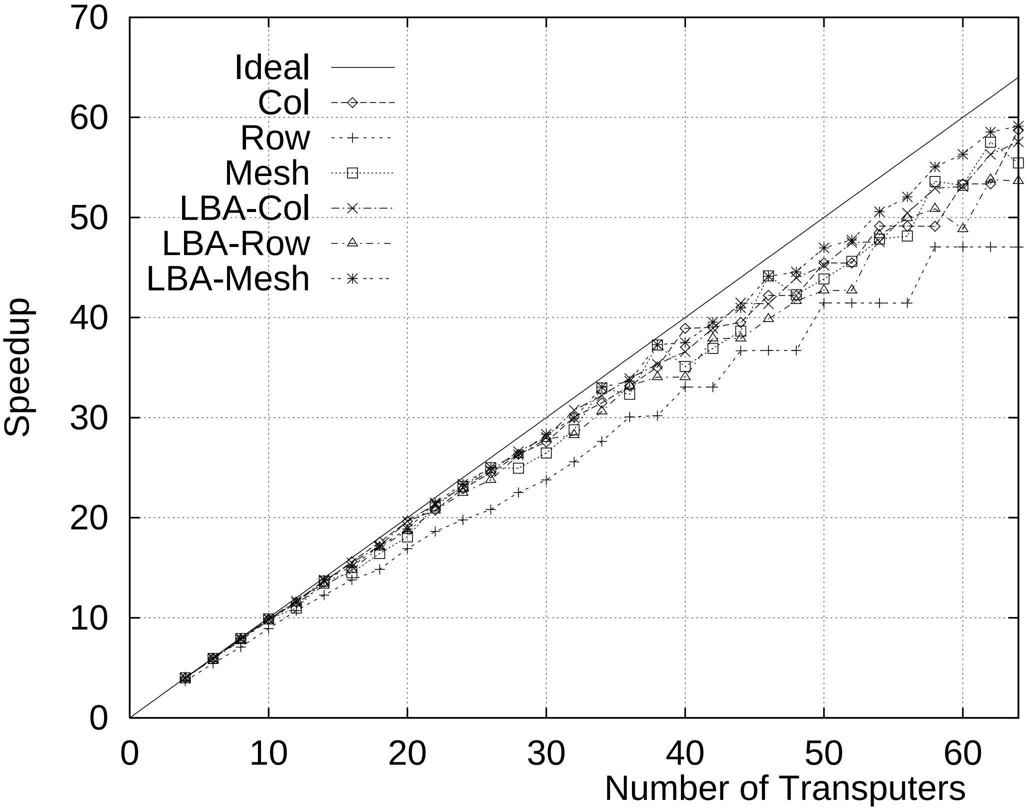

図 6.2. データ 2 による結果

データ 3 は，縦方向に負荷の偏りが大きいデータである．等分割では，それが原因で縦，横分割による結果の差が大きくでている．横分割が最もよい結果となっている．メッシュ分割は，その縦，横分割の中間に位置しているが，負荷の偏りが大きいことと，メッシュの x, y方向の分割数により，分割数が 2 違うだけで相対速度が大きい場合で± 10 も変化している．負荷均衡分割は，縦分割を除いてある程度のリニアな結果が得られている．鏡面に対する負荷の予測がうまく行われていると考えられる．縦分割は，縦方向に負荷の差が大きいため，縦等分割よりよい結果が得られているものの，うまく負荷を予測できていないと思われる．

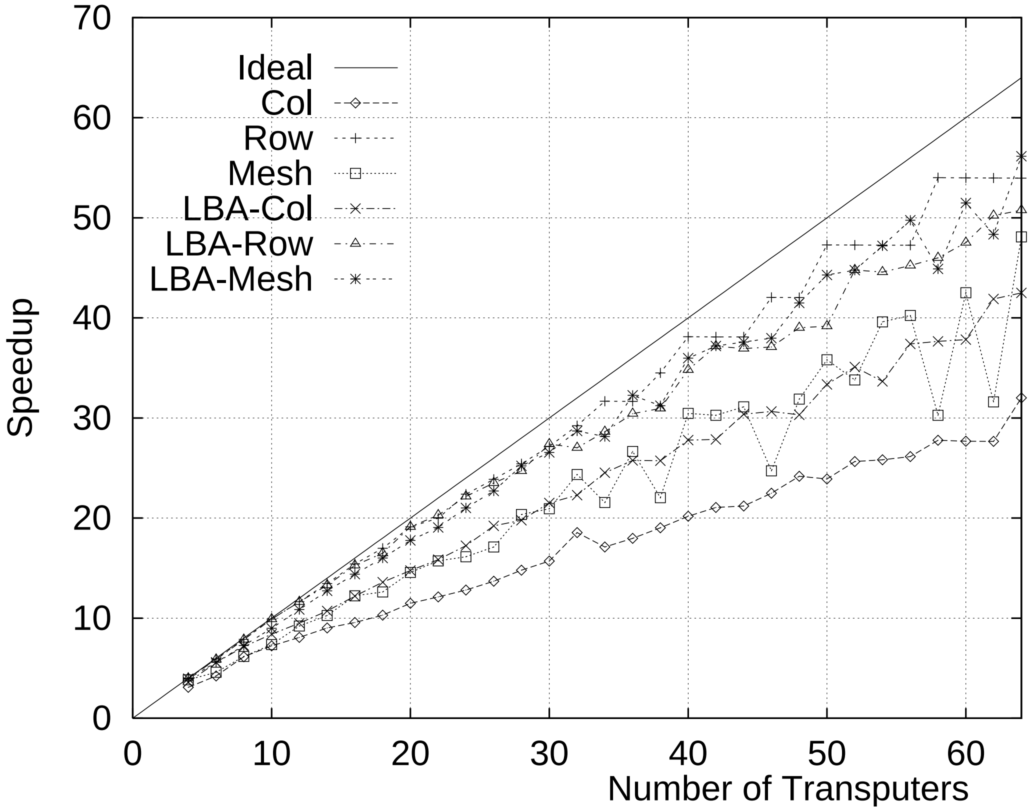

図 6.3. データ 3 による結果

データ 4 は，横方向に負荷の偏りが大きいデータである．等分割では，縦，横分割の差は，データ 3 ほど大きく現れなかったが，横分割が最も悪く，メッシュと縦分割はほぼ同じ結果になっている．負荷均衡分割は，メッシュ分割が最もよく，続いて縦分割，横分割となっている．データ 3 と比べて全体的に相対速度がよく，各分割の結果の差も小さい．これは，画面の大きさの構成が原因と考えられる．画面の構成比が 640×400 となっているため，縦と横の方向に負荷の偏りがある場合を比べると，縦方向に負荷が大きい場合の方が，縦分割時に負荷を分散しにくいためである．もし，画面が 640×640 の場合，データ 3 とデータ 4 の結果は全く同じになるはずである．

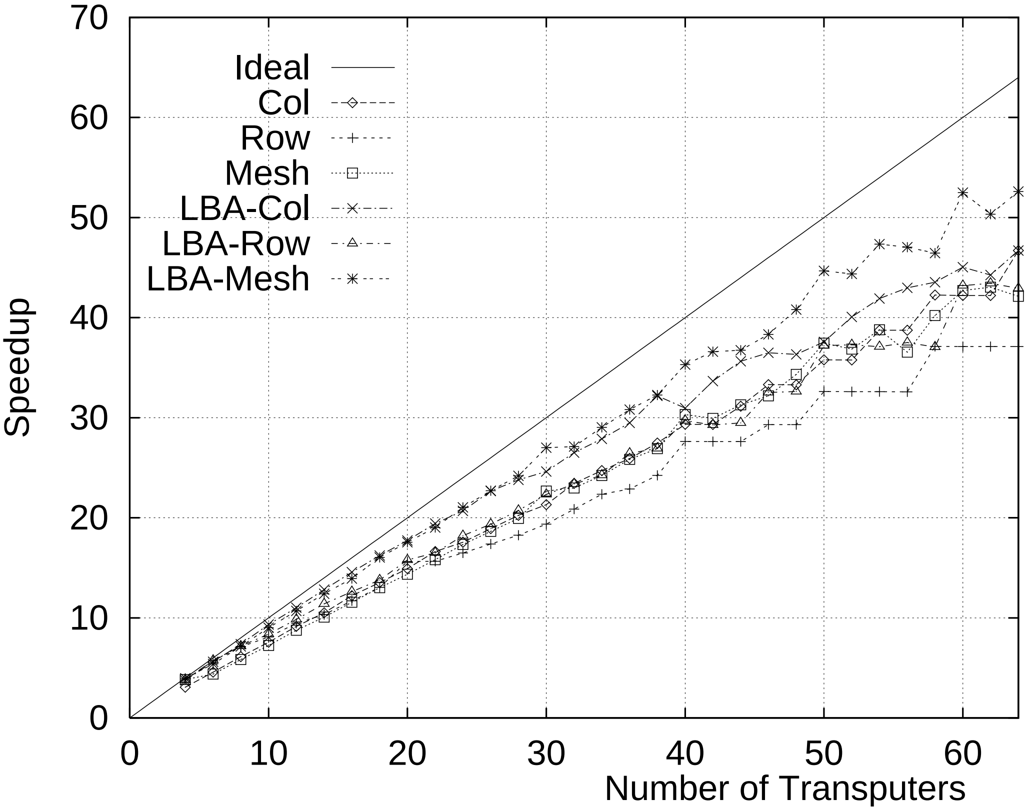

図 6.4. データ 4 による結果

以上の結果から，規則的な画像に対しては，負荷の予測がうまく行われている．また，同じ分割では，負荷均衡分割のほうがよくなっていることがわかる．

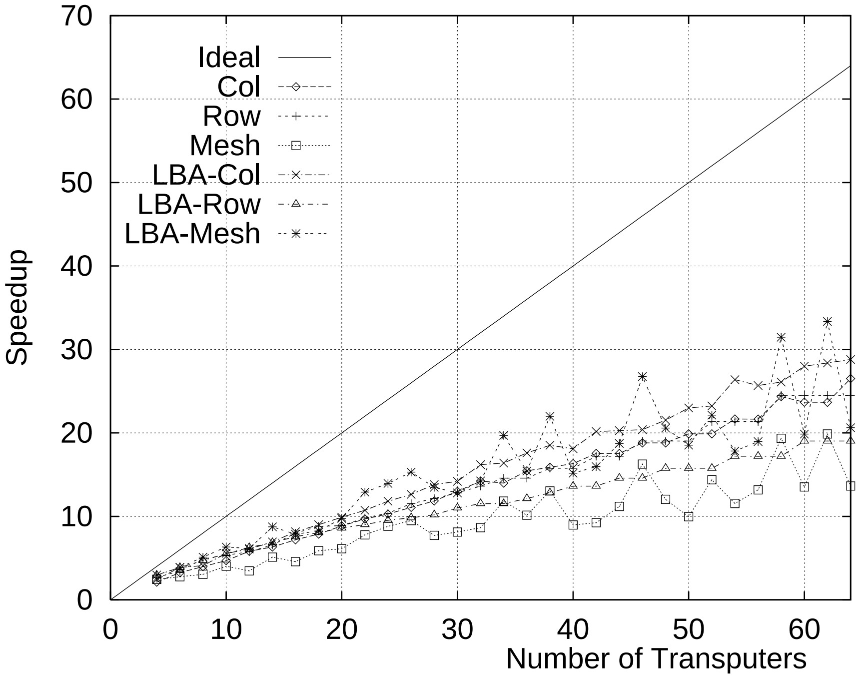

図 6.5. データ 5 による結果

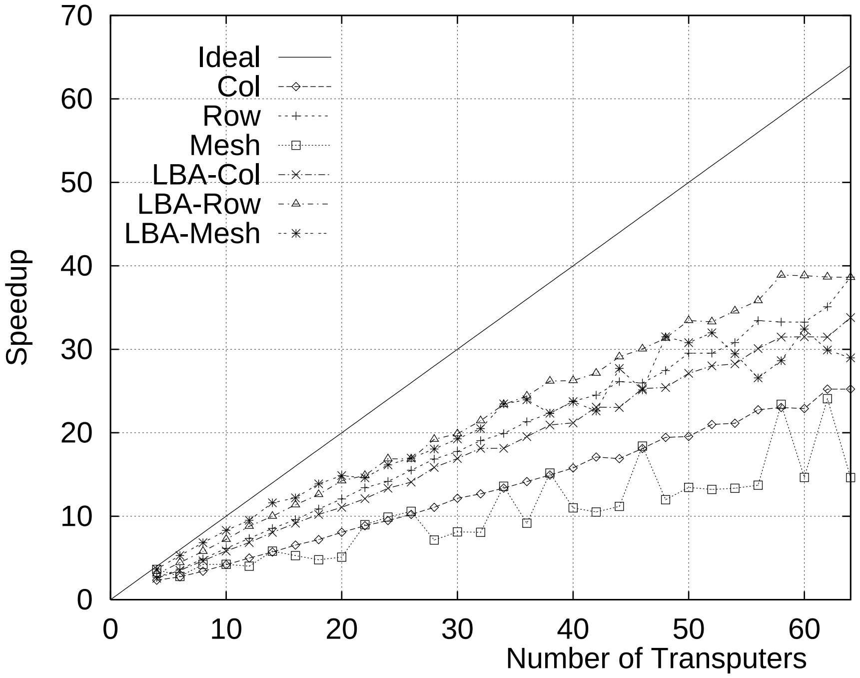

図 6.6. データ 6 による結果

データ5の結果は，全体的に相対速度が悪くなっている．等分割では，縦，横分割はほぼ同じ結果で，メッシュが最も悪くなっている．負荷均衡では，横分割よりも縦分割の方が良くなっている．おそらく画像の横方向に負荷の偏りがあると思われる．

データ6,7の結果は，同じような形のグラフとなっている．等分割，負荷均衡分割ともに，横分割が最もよくなっている．縦方向に負荷の偏りがあると思われる．

データ5,6,7の結果において，メッシュ等分割は最も悪い結果となっており，不規則な画像においては，適していないと思われる．負荷均衡分割は，不規則な画像においても効果があることが分かる．4章で述べた負荷均衡メッシュ分割の改良版で行えば，さらに良い結果が得られると思われる．負荷均衡分割が等分割と比べてそれほどよくない理由は，負荷均衡アルゴリズムが床に対する処理と，画面上で物体が重なった場合の処理が現在のアルゴリズムに含まれていないからである．

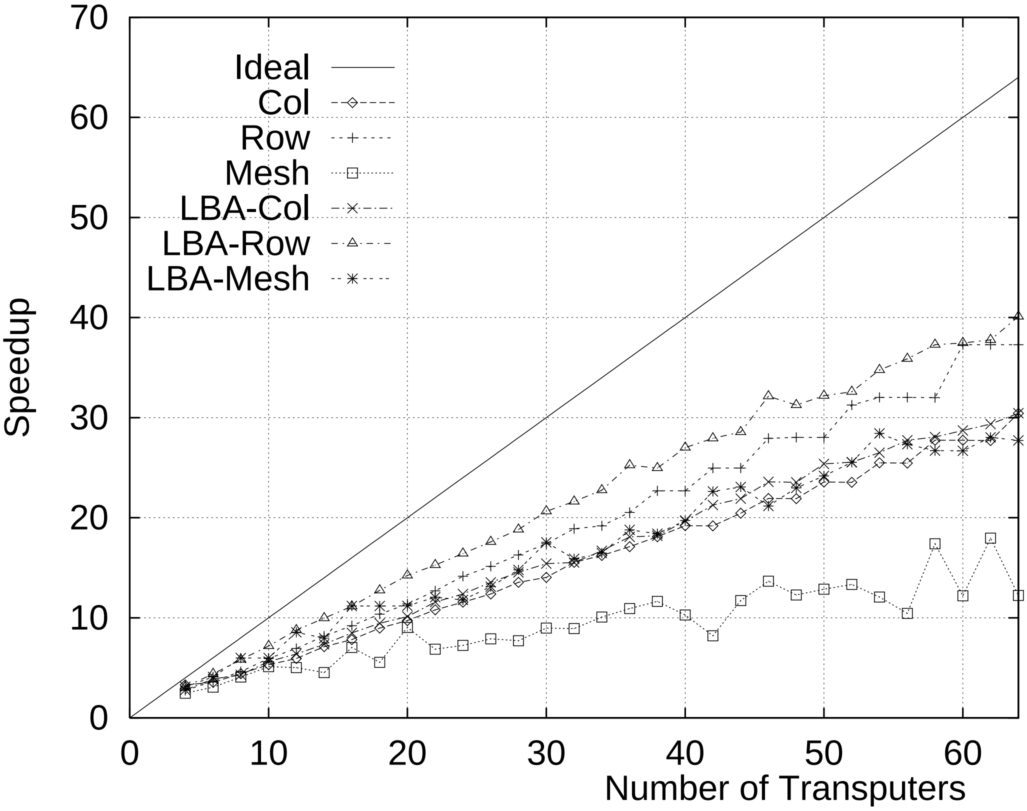

図 6.7. データ7による結果

6.2.2 リング接続とトーラス接続の結果

表 6.2，6.3，6.4，6.5に，トーラス接続による結果を示す．これは，リング接続とトーラス接続 (回収方法 1～3) の実行時間と相対速度を示した表である．

表 6.2は，等分割におけるリング 32 台とトーラス 32 台を比較した表である．確かに，実行時間でみると，リング接続と比べて回収方法 1～3 は速くなっていることが分かる．しかし，ほとんどの場合 1 秒も変わらず，相対速度でみても 1 以上速くなる，つまり，リングのそれと比べて値が 1 以上大きくなるものはほとんどない．表 6.3は負荷均衡分割における結果である．この場合，回収方法 1～3 の結果がよくなっているものが多い．特にデータ 5,6 の縦分割のスピードアップは顕著である．16.2 倍から 23.7 倍， 18.1 倍から 24.1 倍に相対速度がよくなっている．

表 6.4は等分割におけるリング 64 台とトーラス 64 台を比較した表である．32 台のときと同様，回収方法によって相対速度はほとんどかわらない．表 6.5は負荷均衡分割における結果であり，これも 32 台の結果同様，回収方法 1～3 の結果がよくなっている．特にデータ 5,6 の縦分割のスピードアップは大きい．28.8 倍から 43.4 倍，38.6 倍から 49.4 倍になっている．

等分割では，負荷均衡がなされていないため，ルートに各ブロックのデータが戻ってくる時間が同時ではなく，まちまちである．そのため，ルートとつながるプロセッサの付近のデータの混雑があまりないため，リングの実行時間とトーラスによる実行時間にほとんど差がでなかったと考えられる．負荷均衡分割では，各プロセッサの負荷がほぼ均等になっているので，各プロセッサの結果データがほぼ同時に，ルートに送られることになる．そのため，1 方通行でデータを回収するリング接続において，ルートの付近でデータの混雑がおきる．しかし，トーラス接続では，通信距離を短くし，ルートにデータを送る時，2 方向からデータを送り通信の待ち状態を少なくしているため，リング接続と比べ相対速度がよくなったと考えられる．

このシステムでは，ルートは 2 つのプロセッサとしか接続されないので，どの回収方法にしてもこの 2 つのプロセッサにデータが集中することは避けられず，このデータの集中が通信時間のネックとなっていると考えられる．根本的に解決するためには，ルートの残りのリンクを，他のリンクの余っているプロセッサとつなぎ，ルートに返す経路を増やし，データの集中を減らすことであろう．

表 6.2. リング接続とトーラス接続の比較 (等分割,32 台). 単位:(s)

		リング接続		回収方法 1		回収方法 2		回収方法 3	
データ	分割	計算時間	比	計算時間	比	計算時間	比	計算時間	比
1	メッシュ	227.922	**26.3**	227.792	**26.3**	227.686	**26.3**	227.614	**26.3**
	縦	230.301	**26.0**	230.172	**26.1**	230.066	**26.1**	229.927	**26.1**
	横	211.209	**28.4**	209.971	**28.6**	209.971	**28.6**	209.964	**28.6**
2	メッシュ	455.771	**28.8**	455.065	**28.8**	455.040	**28.8**	450.040	**28.8**
	縦	436.474	**30.0**	435.338	**30.1**	435.338	**30.1**	435.332	**30.1**
	横	512.713	**25.6**	511.647	**25.6**	511.647	**25.6**	511.642	**25.6**
3	メッシュ	318.864	**24.3**	318.048	**24.4**	318.049	**24.4**	317.977	**24.4**
	縦	418.426	**18.5**	417.523	**18.6**	417.523	**18.6**	417.384	**18.6**
	横	265.583	**29.2**	264.369	**29.3**	265.368	**29.3**	264.362	**29.3**
4	メッシュ	337.994	**22.9**	337.161	**23.0**	337.161	**23.0**	337.089	**23.0**
	縦	331.282	**23.4**	330.388	**23.5**	330.388	**23.5**	330.180	**23.5**
	横	372.188	**20.8**	370.814	**20.9**	370.814	**20.9**	370.808	**20.9**
5	メッシュ	380.354	**8.6**	378.834	**8.7**	378.799	**8.7**	378.793	**8.7**
	縦	231.287	**14.2**	230.022	**14.3**	230.022	**14.3**	229.882	**14.3**
	横	241.394	**13.6**	240.742	**13.7**	240.652	**13.7**	240.651	**13.7**
6	メッシュ	506.029	**8.1**	503.830	**8.1**	503.794	**8.1**	503.788	**8.1**
	縦	322.233	**12.7**	320.549	**12.7**	320.525	**12.7**	320.526	**12.7**
	横	214.334	**19.1**	212.948	**19.2**	212.948	**19.2**	212.874	**19.2**
7	メッシュ	124.170	**8.9**	121.361	**9.1**	121.256	**9.1**	121.116	**9.1**
	縦	71.317	**15.5**	68.722	**16.1**	68.723	**16.1**	68.567	**16.1**
	横	58.608	**18.9**	56.312	**19.6**	56.212	**19.7**	56.077	**19.7**

表 6.3. リング接続とトーラス接続の比較 (負荷均衡分割,32 台). 単位:(s)

		リング接続		回収方法 1		回収方法 2		回収方法 3	
データ	分割	計算時間	比	計算時間	比	計算時間	比	計算時間	比
1	メッシュ	206.372	**29.1**	205.695	**29.2**	205.694	**29.2**	205.689	**29.2**
	縦	199.837	**30.0**	212.815	**28.2**	212.815	**28.2**	212.810	**28.2**
	横	205.024	**29.3**	199.298	**30.1**	199.928	**30.1**	199.292	**30.1**
2	メッシュ	437.878	**30.0**	436.972	**30.0**	436.972	**30.0**	436.902	**30.0**
	縦	426.571	**30.7**	425.820	**30.8**	425.722	**30.8**	425.723	**30.8**
	横	464.295	**28.2**	463.563	**28.3**	463.538	**28.3**	463.539	**28.3**
3	メッシュ	270.424	**28.7**	269.851	**28.8**	269.822	**28.8**	269.816	**28.8**
	縦	348.264	**22.3**	303.091	**25.6**	303.091	**25.6**	302.968	**25.6**
	横	287.337	**27.0**	266.696	**29.1**	266.695	**29.1**	266.540	**29.1**
4	メッシュ	286.329	**27.1**	285.630	**27.2**	285.630	**27.2**	285.514	**27.2**
	縦	292.856	**26.5**	295.079	**26.3**	295.078	**26.3**	294.946	**26.3**
	横	332.227	**23.4**	294.006	**26.4**	293.922	**26.4**	293.916	**26.4**
5	メッシュ	231.022	**14.2**	229.880	**14.3**	229.855	**14.3**	229.854	**14.3**
	縦	203.096	**16.2**	138.843	**23.7**	138.843	**23.7**	138.841	**23.7**
	横	285.277	**11.5**	372.621	**8.8**	372.433	**8.8**	372.433	**8.8**
6	メッシュ	199.273	**20.5**	195.427	**20.9**	195.427	**20.9**	195.423	**20.9**
	縦	225.246	**18.1**	169.687	**24.1**	169.560	**24.1**	169.557	**24.1**
	横	190.471	**21.4**	164.220	**24.9**	163.905	**24.9**	163.905	**24.9**
7	メッシュ	69.699	**15.9**	66.823	**16.6**	66.822	**16.6**	66.819	**16.6**
	縦	71.388	**15.5**	53.970	**20.5**	53.969	**20.5**	53.967	**20.5**
	横	51.268	**21.6**	53.627	**20.6**	53.627	**20.6**	53.620	**20.6**

表 6.4. リング接続とトーラス接続の比較 (等分割,64 台). 単位:(s)

		リング接続		回収方法 1		回収方法 2		回収方法 3	
データ	分割	計算時間	比	計算時間	比	計算時間	比	計算時間	比
1	メッシュ	116.064	**51.7**	115.325	**52.0**	115.236	**52.1**	115.233	**52.1**
	縦	124.140	**48.3**	123.487	**48.6**	123.354	**48.6**	123.321	**48.6**
	横	114.595	**52.3**	113.521	**52.8**	113.522	**52.8**	113.441	**52.9**
2	メッシュ	236.488	**55.5**	235.596	**55.7**	235.570	**55.7**	235.571	**55.7**
	縦	223.322	**58.7**	222.541	**58.9**	222.457	**59.0**	222.454	**59.0**
	横	278.765	**47.0**	277.709	**47.2**	277.709	**47.2**	277.705	**47.2**
3	メッシュ	161.406	**48.1**	160.789	**48.3**	160.700	**48.3**	160.690	**48.3**
	縦	242.566	**32.0**	241.625	**32.1**	241.626	**32.1**	241.623	**32.1**
	横	143.859	**54.0**	142.799	**54.4**	142.799	**54.4**	142.684	**54.4**
4	メッシュ	184.305	**42.1**	183.616	**42.3**	183.616	**42.3**	180.530	**43.0**
	縦	166.257	**46.7**	165.279	**47.0**	165.279	**47.0**	165.276	**47.0**
	横	209.252	**38.4**	208.137	**37.3**	208.137	**37.3**	208.133	**37.3**
5	メッシュ	241.231	**13.6**	240.055	**13.7**	240.038	**13.7**	240.035	**13.7**
	縦	124.125	**26.5**	123.071	**26.7**	123.070	**26.7**	123.068	**26.7**
	横	134.279	**24.5**	133.956	**24.6**	133.855	**24.6**	133.856	**24.6**
6	メッシュ	279.223	**14.6**	277.936	**14.7**	277.883	**14.7**	277.881	**14.7**
	縦	161.817	**25.2**	160.727	**25.4**	160.704	**25.4**	160.703	**25.4**
	横	105.672	**38.6**	104.951	**38.9**	104.933	**38.9**	104.899	**38.9**
7	メッシュ	90.548	**12.2**	89.302	**12.4**	89.214	**12.4**	89.211	**12.4**
	縦	36.407	**30.4**	34.524	**32.1**	34.523	**32.1**	34.520	**32.1**
	横	29.715	**37.3**	28.209	**39.2**	28.124	**39.4**	28.121	**39.4**

表 6.5. リング接続とトーラス接続の比較 (負荷均衡分割,64 台). 単位:(s)

		リング接続		回収方法 1		回収方法 2		回収方法 3	
データ	分割	計算時間	比	計算時間	比	計算時間	比	計算時間	比
1	メッシュ	104.912	**57.2**	104.120	**57.6**	104.120	**57.6**	104.116	**57.6**
	縦	107.006	**56.0**	112.085	**53.5**	112.060	**53.5**	112.060	**53.5**
	横	105.187	**57.0**	103.732	**57.8**	103.732	**57.8**	103.691	**57.9**
2	メッシュ	221.816	**59.1**	220.908	**59.3**	220.908	**59.3**	220.906	**59.4**
	縦	227.923	**57.5**	225.597	**58.1**	225.477	**58.2**	225.477	**58.2**
	横	244.496	**53.6**	237.908	**55.1**	237.900	**55.1**	237.987	**55.1**
3	メッシュ	138.260	**56.1**	137.804	**56.3**	137.789	**56.3**	137.786	**56.3**
	縦	182.677	**42.5**	157.105	**49.4**	157.105	**49.4**	157.044	**49.4**
	横	152.921	**50.8**	147.142	**52.8**	146.984	**52.5**	146.895	**52.5**
4	メッシュ	147.575	**52.6**	146.932	**52.8**	146.932	**52.8**	146.902	**52.9**
	縦	166.254	**46.7**	163.046	**47.6**	163.047	**47.6**	163.044	**47.6**
	横	180.868	**42.9**	150.216	**51.7**	150.218	**51.7**	150.215	**51.7**
5	メッシュ	159.461	**20.6**	159.167	**20.7**	158.924	**20.6**	158.635	**20.6**
	縦	114.235	**28.8**	75.782	**43.4**	75.782	**43.4**	75.733	**43.4**
	横	172.895	**19.0**	240.875	**13.7**	240.653	**13.7**	240.653	**13.7**
6	メッシュ	140.970	**29.0**	136.010	**30.0**	136.011	**30.0**	135.476	**30,1**
	縦	120.852	**33.8**	100.847	**40.5**	100.847	**40.5**	100.672	**40.6**
	横	105.757	**38.6**	82.637	**49.4**	82.262	**49.6**	82.258	**49.6**
7	メッシュ	39.970	**27.7**	39.031	**28.3**	39.031	**28.3**	39.027	**28.3**
	縦	36.399	**30.4**	30.254	**36.6**	30.254	**36.6**	30.078	**36.8**
	横	27.626	**40.1**	27.280	**40.6**	27.280	**40.6**	27.145	**40.8**

6.2.3 3DDDA による空間分割の結果

表 6.6に通常のレイトレーシングと空間分割によるレイトレーシングの実行時間と速度向上比を示す．速度向上比とはこの場合，”通常の光線追跡による実行時間 / 空間分割による実行時間”のことである．図 6.8は，実際に行った分割で，物体を 50 個配置した場合を表している．物体が 25 個の場合 010 のみに物体を登録し ,50 個の時は 010, 110 に各 25 個登録し ,75 の時は ,010, 000, 110 に各 25 個の物体を登録する .100 個の時は，各ボクセルに 25 ずつ登録する．

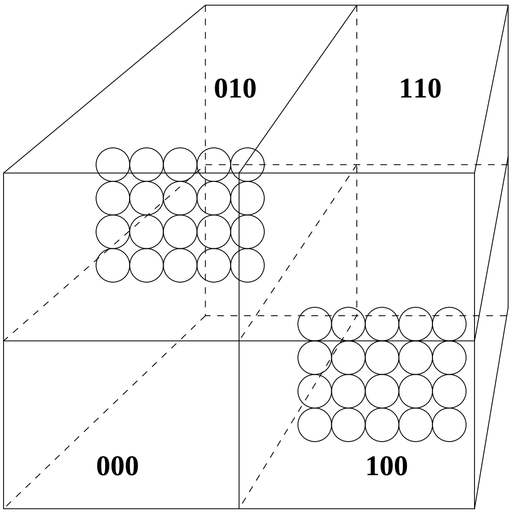

図 6.8. 2 × 2 × 1 による空間分割

表 6.6. 通常の光線追跡と空間分割による光線追跡の実行時間の比較 (単位: s)

物体数	通常の光線追跡	空間分割による光線追跡	速度向上比
25	111	53	2.1
50	248	106	2.3
75	409	163	2.5
100	601	218	2.8

明らかに，空間分割の結果が良くなっているのがわかる．3DDDA と空間分割により，無駄な交差判定を大幅に削減できたと思われる．最高で物体が 100 個の時 2.8 倍の速度向上が得られている．マルチプロセッサで行う場合，3DDDA によるオーバヘッド以外に，パケット通信によるオーバヘッドが生じると考えられるので，これらについて検討する必要がある．

図 6.9. データ 1 による画像

図 6.10. データ 2 による画像

図 6.11. データ 3 による画像

図 6.12. データ 4 による画像

図 6.13. データ 5 による画像

図 6.14. データ 6 による画像

図 6.15. データ 7 による画像

第 7 章　おわりに

本論文では，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た．画像の品質がよいだけに，それ相応の実行時間を要することは避けられないが，レイトレーシングのアルゴリズムは単純であり，並列化への適応度は非常に大きい．負荷均衡アルゴリズムにより最高で 59.4 倍の相対速度を得ることができた．

画面等分割と負荷均衡分割，リング接続とトーラス接続との比較，空間分割のトランスピュータへ実装方法，そして 3DDDA を用いた空間分割の効果を試した．その結果，負荷均衡アルゴリズムによる画面分割の有効性が認められ，さらにトーラス接続にすることにより実行時間を短縮することができた．規則的な画像に対しては相対速度がよくなり，不規則な画像に対しては，全体的に相対速度が落ちるものの，等分割よりもよいと結果なった．しかし，鏡面や屈折に対する負荷の予測の精度がまだよくなく，画面上で物体が重なる時の処理がないので，これらを解決すれば，特に不規則な画像に対して相対速度がさらによくなると思われる．空間分割では，3DDDA を用いて実行時間の短縮が認められ，最高で 2.8 倍の高速化が確認できた．しかし，物体が少ない場合，均等に配置されている場合，ボクセルの数が多い場合など，実行時間の短縮を阻む要素が多い．マルチプロセッサではさらに，通信のオーバヘッドが考えられるので，その点を考慮していく必要がある．

今後の課題として，画面分割では，球体だけでなく他の 2 次曲面の負荷の均衡と正確な予測，特に本論文で提案したメッシュの改良版による負荷均衡分割を行う．空間分割においては，今回行った 3DDDA では，ボクセルへの物体の登録はすべて手動で行っていた．物体データが変わるたびに，どの物体がどの空間に属するかを記述していては手間がかかる．すべての物体について，どのボクセルに属しているかを計算し，自動的に登録するアルゴリズムが必要である．物体が 2 つのボクセルにまたがって存在する場合なども考慮するべきであろう．また，分割方法を等分割としていたが，オクトリによる分割など，各ボクセルに含まれる物体の数が均等になるまで分割するなどといった適応型空間分割 [23] も，考えていかなければならない．これらを含めた，本システムを用いた空間分割の実現が課題である．

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，並びに得丸英勝教授，渡部透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，山崎恭啓氏に感謝いたします．さらに，共同研究者である吉谷崇史氏に感謝します．最後に，本研究に助言をして頂いた得丸・渡部・西川研究室の吉田一貴氏に感謝の意を表します．

参考文献

[1] 富田眞治，権　五鳳，明日英也，“ 並列グラフィックス計算機の研究開発動向”，PIXEL, no.131, pp.132-139, (1994).

[2] Andrew S.Glassner, “ Space Subdivision for Fast Ray Tracing”, *IEEE Computer Graphics and Applications*, vol.4, no.10, pp.16-22, (October 1987).

[3] 千葉則茂／村岡− 信，“ C による CG レイトレーシング”，サイエンス社, (1991-6).

[4] 村田誠治，權　五鳳，村上和彰，富田眞治“ 熱視線』:視線探索法を高速処理する専用並列レンダリング ・マシン-メモリ構成およびその評価”，情報処理学会研究報告，ARC-97-13, pp.97-104, (1992-12).

[5] 權　五鳳，村田誠治，村上和彰，“ 3MPRT(3-stage Macropipelined Ray Tracing) マシン-物体検索法の高速化に関する評価-”，電子情報通信学会研究報告，CPSY92-19, pp.79-86, (1992-8).

[6] John G.Cleary, Brain M.Wyvill, Graham M.Birtwistle, Reddy Vatti, “ Multiprocessor Ray Tracing”, *IEEE Computer Graphics and Applications*, vol.5, no.1, pp.3-12, (March 1986).

[7] 竹安俊幸，大西充久，河合利幸，　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, pp.2-501-503, (1991-10).

[8] M.D.J.McNeil, 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).

[9] 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).

[10] 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).

[11] 原田博之，山崎勝弘，得丸英勝，渡部透，“ トランスピュータによるレイトレーシング法の並列処理 ”，第 37 回システム制御情報学会研究発表講演会，pp325-326,(1993-6).

[12] 上嶋　明，“ トランスピュータによる画像生成の高速化 ”，立命館大学理工学研究科情報工学専攻修士論文, (1993-2).

[13] 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, (1993-1).

[14] 河合利幸，大西充久，　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 ”，電子通信情報学会論文誌 D-II, vol.J76-D-II, no.8, pp.1763-1771(1993-8).

[15] 大西充久，竹安俊幸，河合利幸，　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, pp.2-499-500(1991-10).

[16] Didier Badouel, Thierry Priol, “ An Efficient Parallel Ray Tracing Scheme for Highly Parallel Architectures”, *COMPUTER & GRAPHICS*, vol.15, no.4, pp.549-552, (1991).

[17] Hiroaki KOBAYASHI, Hideyuki KUBOTA, Susumu HORIGUCHI, Tadao NAKAMURA “ Load Balancing Based on Load Coherence between Continuous Images for an Object-Space Parallel Ray-Tracing System”, *IEICE TRANS.INF.&SYST*, vol.E76-D , no.12, pp.1490-1498,(December 1993).

[18] 原田博之，山崎勝弘，渡部透，得丸英勝，“ 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).

[19] 山田哲也，安浦寛人，“ FMPP によるレイトレーシングの高速化手法について”，情報処理学会研究報告，ARC-97-14，pp.105-112，(1992-12).

[20] Andrews. Glassner 著，白田耕作監訳，“3 次元コンピュータグラフィックス”, pp.137-138, (1991-4).

[21] 高桑昌男，“ CG レイトレ物語”, pp.64-67, (1990-8).

[22] Roni Yagel, Daniel Cohen, Arie Kaufman, “ Discrete Ray Tracing ” , *IEEE Computer Graphics and Applications*, pp.19-28, (September 1992).

[23] B.S.S.Pradhan, A.Mukhopadhyay, “ ADAPTIVE CELL DIVISION FOR RAY TRACING ” , *COMPUTER & GRAPHICS*, vol.15, no.4, pp.549-552,(1991).

[24] F.Van Reeth, W.Lamotte, E.Flerackers, “ RAY TRACING SPEED-UP TECHNIQUES USING AN MIMD ARCHITECTURE” , *Programing & Computer Software*, vol.18, no.4, pp.173-181,(1992).

[25] Hao-Ren Ke, Ruei-Chuan Chang, " An efficient hierachical-traversal algorithm for ray tracing" , *The Visual Computer*, vol.10, no.2, pp.79-87,(1993).

[26] G.Simiakakis, A.M.Day, " Five-dimensional Adaptive Subdivision for Ray Tracing" , *COMPUTER GRAPHICS forum*, vol.13, no.2, pp.133-140, (1994).