数理解析研究所講究録 1362 巻 2004 年 71-78

自動微分法とシンプレクティック積分法を用いたグラフィックス描画法

独立行政法人 通信総合研究所 佐藤 哲 (Tetsu R. Satoh)

Communications Research Laboratory

1 はじめに

「計算機による計算結果には，人間が行う手計算のように単純なミスは混入しないが，なんらかの誤差が含まれる」というのが昔からの計算機利用者の認識であった．しかし近年では，計算機による計算の品質を高め，信頼できる結果を出力するための研究が盛んである．本論文では，そのような研究の応用として，信頼できる数値計算法を利用したコンピュータ・グラフィックス（CG）の描画手法を紹介する．

CG の描画手法として古くから知られている光線追跡法は，曲進する光線も扱えるように拡張されている．しかし，曲進する光線の軌道を計算するための数値計算手法の精度について，深く議論した研究は少なかった．そこで本論文では，光線の軌跡を表す方程式としてハミルトンの正準方程式を採用することで従来の非線形光線追跡法で用いられてきた多くの方程式を統一的に扱い，自動微分法により正準方程式を自動的に導出し，シンプレクティック積分法により信頼性のある正準方程式の解を求める手法について述べる．

2 非線形光線追跡法によるコンピュータ・グラフィックス

光線追跡法は，視線と配置オブジェクトの交点を検出し，交点の配置オブジェクトの色を描画することで CG を描画する手法である [1]．通常，光線は直進すると仮定されているが，非線形光線追跡法ではその仮定が取り払われている [2]．物理法則を忠実に計算し，非線形な光線追跡法に基づき CG を作成した研究は，重い天体により時空が歪む現象に対し適用されるものが代表的である [3] [4][5][6]．非線形光線追跡法も，概念的には古典的な光線追跡法と同様であり，以下のような処理で CG を描画する．

(1) 観測者の視点の座標と視野スクリーンを設定する．そして視野スクリーンの全ての画素に対し，以下の処理を繰り返す．
   (1) 視点と画素を結ぶ視線ベクトルを構成し，視線ベクトル方向に光線を発射する．そして光線がオブジェクトと交差するまで光線を延ばす．
   (2) オブジェクトと光線の交差点の色情報を，画素に与える．

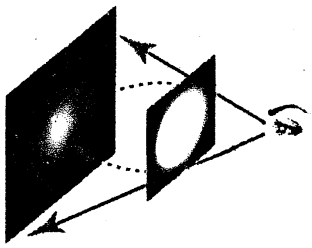


図 1: 曲がる光線により歪んだ映像が観測されることの概念図

光線が曲がる場合，観測者が観測している映像が歪むことは次のように説明される．図 1 のように，右端に観測者，左端に宇宙の銀河が存在するとする．もし観測者と銀河の間にブラックホールのような重い天体が存在すれば，銀河から出た光は点線のような軌道で観測者の視界に入る．しかし観測者は光線は図中の実線のように直進してきたと認知するので，結果的に中央のように銀河中心部の白い部分が大きく引き伸ばされた歪んだ光景を観測してしまう．

古典的な光線追跡法と非線形光線追跡法は，光線の軌道を表す方程式が異なる．古典的な光線追跡法で使用する光線の軌道を表す方程式は，直線である

Tetsu R. Satoh (tetu-s@crl.go.jp)
Communications Research Laboratory

ので一次関数である:

$$y = ax + b. \tag{1}$$

式 (1) をベクトルの媒介変数表示を用いて書き直すと，次のようになる：

$$\boldsymbol{r} = \boldsymbol{a} + t\boldsymbol{v}. \tag{2}$$

式 (2) は，$\boldsymbol{a}$ を基点として方向ベクトル $\boldsymbol{v}$ の方向に伸びる直線を表している．ところで式 (2) は，ベクトルの各成分のみを書くことにすると，次のように表すことができる．$\boldsymbol{r} = (x_1, x_2, x_3)$ とすると

$$x_i = a_i + tv_i. \tag{3}$$

そしてパラメータ t で 2 回微分すると，式 (3) の右辺は t を除いて定数なので消えてしまい，次式が得られる．

$$\frac{d^2x_i}{dt^2} = 0. \tag{4}$$

これが直線の微分方程式である．古典的光線追跡法は，数学的には式 (4) を基礎として光線の軌跡を計算する手法と言える．

ところで，地球上では光線が直進しない物理現象が多数存在する．蜃気楼や陽炎のように温度差によって媒質の屈折率が変化する場合が代表的な例で，その場合は直線の方程式は次のように自然に拡張される．

$$\frac{dn}{dt}\frac{dx_i}{dt} + n\frac{d^2x_i}{dt^2} = \frac{\partial n}{\partial x_i}, \tag{5}$$

ここで，n は屈折率を表す関数である．式 (5) は，蜃気楼の可視化に関連する文献 [7] や文献 [8] で用いられている式と形は違うが，数学的には同等である．式 (5) において，もし屈折率が一様なら n は定数であり，式 (4) に帰着することが分かる．

通常の空間幾何学（ユークリッド幾何学）を拡張したものが非ユークリッド幾何学であり，代表的なものの一つがリーマン幾何学である．そのリーマン幾何学の中で，ユークリッド幾何学の直線に相当する概念は測地線と呼ばれ，次式で表される [9].

$$\frac{d^2x_i}{dt^2} + \sum_{k,l}\Gamma^i_{kl}\frac{dx_k}{dt}\frac{dx_l}{dt} = 0. \tag{6}$$

ここで Γ^i_{kl} はクリストッフェル記号と呼ばれ，時空の歪み具合と関係している量である．時空が平坦なら Γ^i_{kl} はゼロとなり，やはり式 (6) は式 (4) に帰着し，測地線が直線の歪んだ時空に対する自然な拡張になっていることが分かる．

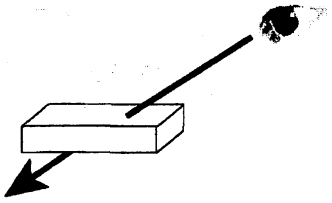


図 2: 古典的な光線追跡法

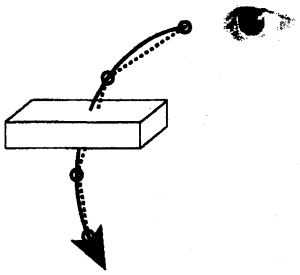


図 3: 非線形光線追跡法

統一的なアプローチとしては，ハミルトンの正準方程式 [10]

$$\begin{cases} \dfrac{dp}{dt} = -\dfrac{\partial H}{\partial q} \\ \dfrac{dq}{dt} = \dfrac{\partial H}{\partial p} \end{cases} \tag{7}$$

を用いることが考えられる．ここで，q は位置座標成分，p は q に対応した運動量である．H はハミルトニアンと呼ばれるスカラー関数である．このアプローチでは，式 (7) に対し適切はハミルトニアンを与えることで，式 (4) ～式 (6) を用いた場合と理論的に同値な計算が可能である．

光線とオブジェクトとの交差点の計算は，古典的な光線追跡法では，図 2 で示すように直線である視線とオブジェクトの交差点を，代数方程式を解くことにより求める．一方，非線形光線追跡法では光線の軌道が曲線になるので，図 3 で示すように局所的に直線近似をし，その線分とオブジェクトの交差点を，代数方程式を解くことにより求める．古典的な

光線追跡法では光線一本に対し代数方程式を計算すればよいのに対し，非線形光線追跡法では区分ごとの計算が必要となるので，非線形光線追跡法の方が大幅に計算コストが高くなる．

3 自動微分と数値積分

3.1 高速自動微分法

前節において，式 (7) を用いれば多くの現象を統一的に扱えることを紹介したが，プログラムとして実装するためにはハミルトニアンを準備し，ハミルトニアンを式 (7) に従って偏微分した方程式を書き下し，サブルーチンなどの形でプログラミングしなければならない．しかしそれでは理論的には形式的に統一的に扱えても，実装としては別々に扱わなければならず，真に統一的に扱えるとは言えない．そこで，偏微分して方程式を書き下すという部分を自動化するための技術が自動微分法 [11][12] である．自動微分法では，ある演算を実行する際に演算の微分も同時に計算し，レジスターに蓄積していく．例えば，乗算という演算を自動微分法として実装する場合，次のように乗算の計算と同時に乗算の微分であるライプニッツ則を計算する．C++言語では，例えば次のような実装になる．

```
autodiff autodiff::operator*(autodiff a)
{
  autodiff ret;
  ret.value = value * a.value;
  ret.diff = value*a.diff + diff*a.value;
  return(ret);
}
```

このような演算を再帰的に実行することで，出力レジスタには演算結果と共に導関数の値が格納される．乗算の実装例からも分かるように，微分の計算規則をプログラム中に内蔵させるので微分を正確に計算することになり，数値微分のような打切り誤差は発生しない．

自動微分法を導入することで，式 (7) を採用して非線形光線追跡法を実行するためには，ハミルトニアンの値を計算するサブルーチンを実装すれば，ハミルトニアンの導関数の値も自動的に計算され，式

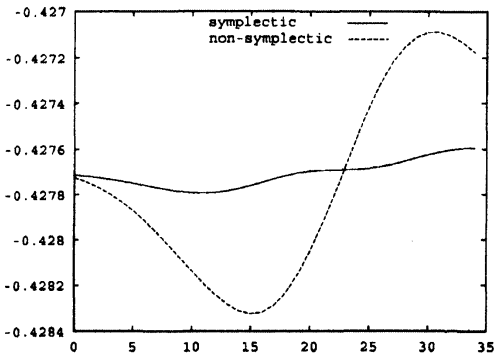


図 4: 蜃気楼が発生している状況で光線追跡を行った場合のエネルギー値

(7) の方程式が自動的に導かれる．従って自動微分法により局所的な光線の軌道を計算する方程式が自動的に導出されることになる．

3.2 シンプレクティック数値解法

自動導出された方程式は，解を求めなければならない．そのために数値積分法を用いれば，数値計算の誤差の範囲で微分方程式の解を人手による式変形を必要とせずに解を求めることができる．オイラー法や Runge-Kutta 法といった有名な数値積分法は，計算結果に打切り誤差を含むために誤差の蓄積に注意しなければならない．ここで，非線形光線追跡法においては，オブジェクトと光線の交差点を正確に求めることが大切であり，その過程の光線の軌道が正確かどうかは重要でないという点に着目する．表現を変えると，局所的な数値誤差よりも大域的な数値誤差を小さくすれば良いと言える．このような目的に対して開発された数値解析法の一つに，幾何的数値解析法 [13] がある．幾何的数値解析法の中で，ハミルトンの正準方程式を扱う場合に適するシンプレクティック数値積分法 [14] [15] がある．シンプレクティック数値積分法は，正確には次のように定義されるシンプレクティック性を保存量として持つ．

定義 1 $(p(t_0), q(t_0))$ を $t = t_0$ での光子の座標であるとする．$(p(t_0 + \Delta t), q(t_0 + \Delta t))$ は $t = t_0 + \Delta t$ での座標であるとする．任意の実数 t_0 に対し，写像 $(p(t_0), q(t_0)) \mapsto (p(t_0 + \Delta t), q(t_0 + \Delta t))$ がシンプレ

クティック性を保存するのは，

$$dq(t_0) \wedge dp(t_0) = dq(t_0 + \Delta t) \wedge dp(t_0 + \Delta t) \quad (8)$$

が成り立つ時およびその時に限る．ここで，dp，dq は各変数の *1*-形式であり，$\wedge$ は微分形式の外積を表している．

この手法は長期間の数値積分に強い特徴があり，安定性が高い．図 4 に，非線形光線追跡法を用いて蜃気楼のシミュレーションを実行した場合の，光線の軌道上でのエネルギーの値を示す．縦軸がエネルギーの値で，横軸が光線を発射してからの光線の経路長を表すパラメータである．破線が Runge-Kutta 法を用いて正準方程式を解いたものであり，実線がシンプレクティック法を用いたものである．打切り誤差の精度はどちらも 4 次である．数値計算の刻み幅も打切り誤差の精度も同じであるにも関わらず，シンプレクティック法の方がエネルギーを良好に保存していることが分かる．ここではシンプレクティック法として，半陰的オイラー法

$$\begin{cases} p_{k+1} &= p_k - \tau \dfrac{\partial H(p_{k+1}, q_k)}{\partial q_k} \\ q_{k+1} &= q_k + \tau \dfrac{\partial H(p_{k+1}, q_k)}{\partial p_{k+1}} \end{cases} \quad (9)$$

に対し対称分解法 [16][17] を用いて精度を高めたものを使った．具体的には，光線の運動に対するハミルトニアンは多くの場合に運動量成分について分離可能となるので，それを H_A，H_B とおく：

$$H(p, q) = H_A(p, q) + H_B(p, q) \quad (10)$$

そして刻み幅 τ で式 (9) を用いて H_A，H_B をハミルトニアンとする正準方程式を解くルーチンを $S_A(\tau)$，$S_B(\tau)$ とすると，2 次の精度を持つ解法が次のように構成される：

$$S_2(\tau) \equiv S_A(1/2\tau) S_B(\tau) S_A(1/2\tau) \quad (11)$$

同様に，

$$S_4(\tau) \equiv S_2(d_1\tau) S_2(d_2\tau) S_2(d_1\tau) \quad (12)$$

が 4 次の精度を持つ解法となる．ここで，d_1，d_2 は

$$\begin{cases} d_1 &= \dfrac{4 + 2^{2/3} + 2^{4/3}}{6} = 1.35121\cdots \\ d_2 &= -\dfrac{(1 + 2^{1/3})^2}{3} = -1.70241\cdots \end{cases} \quad (13)$$

と定義される定数である．ただし式 (13) を用いると，結果的に数値計算の刻み幅 τ が d_1 倍，d_2 倍されることになるため，τ として大き目の値を使いたい場合は解法 (12)，(13) では精度や安定度が落ちる場合がある．その場合は例えば

$$\begin{aligned} S_4(\tau) \equiv\ & S_2(d_1\tau) S_2(d_1\tau) S_2(d_1\tau) S_2(d_1\tau) S_2(d_2\tau) \\ & \times S_2(d_1\tau) S_2(d_1\tau) S_2(d_1\tau) S_2(d_1\tau) \end{aligned} \quad (14)$$

$$\begin{cases} d_1 &= \dfrac{1}{6} = 0.16666\cdots \\ d_2 &= -\dfrac{1}{3} = -0.33333\cdots \end{cases} \quad (15)$$

のようなスキームを構成する．ただし S_2 の評価回数が増えるので計算速度は低下する．また，陰的な解法なので処理中で連立方程式を解く必要がある．本研究では連立方程式の解法はニュートン法を用いている．

以上述べたように，自動微分法と数値積分法を用いることにより，スカラー関数としてハミルトニアンの計算ルーチンを与えることにより光線追跡が可能となるシステムが実現する．ハミルトン力学で扱えるいかなる現象もハミルトニアンを与えるだけで実行できるため，多数の現象を統一的に扱えるようになると言える．

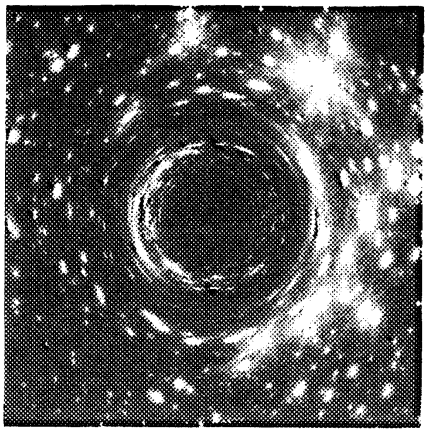
4 シンプレクティック・レイトレーシングの実装

4.1 画像生成例

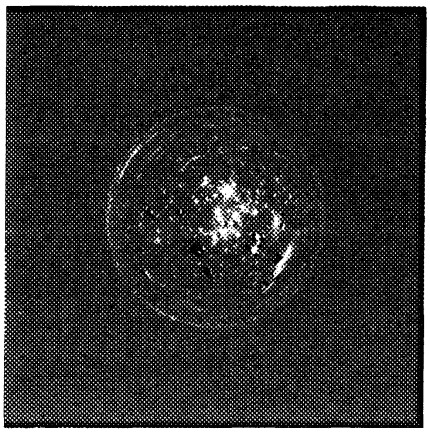
シンプレクティック・レイトレーシング [18] [19][20] とは，非線形光線追跡法に対し自動微分法とシンプレクティック積分法を導入したものである．本研究では，この手法を c++を用いて実装しており，Linux PC，NEC SX-6，SGI Onyx と多くのプラットフォーム上で動作している．以下，実行例を紹介する．

図 5 は球対称ブラックホール時空内で光線追跡をした例で，ハミルトニアンは次のように表される：

$$\begin{aligned} H =\ & -\frac{1}{2}\left(1 + \frac{r_g}{r}\right) p_t^2 + \frac{1}{2}\left\{\left(1 - \frac{r_g x^2}{r^3}\right) p_x^2 \right. \\ & \left. + \left(1 - \frac{r_g y^2}{r^3}\right) p_y^2 + \left(1 - \frac{r_g z^2}{r^3}\right) p_z^2 \right\} \\ & - \frac{r_g p_t}{r^2}\left(x p_x + y p_y + z p_z\right) \\ & - \frac{r_g}{r^3}\left(xy p_x p_y + xz p_x p_z + yz p_y p_z\right) \end{aligned} \quad (16)$$



(a)



(b)

図 5: (a) 球対称ブラックホールの可視化例．銀河の画像がブラックホールによる重力で歪んでいる．(b) 重力凹レンズ効果の可視化例．ブラックホールの近傍でブラックホールに背を向けると，このように宇宙全体が前方に集まって見える．

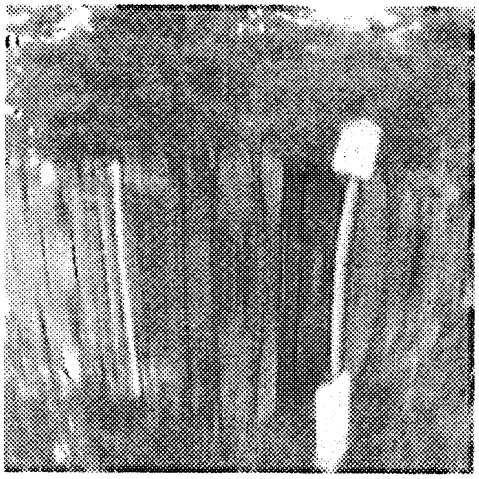


図 6: 蜃気楼現象としての逃げ水のシミュレーション例

ここで，(t, x, y, z) は 4 次元カーテシアン座標系の成分，(p_t, p_x, p_y, p_z) は対応する運動量成分，$r = \sqrt{x^2 + y^2 + z^2}$，$r_g$ はブラックホール半径と呼ばれるブラックホールの質量に対応する量である．図 5a と図 5b の違いは光線の発射方向，すなわち観測者の視線方向で，図 5a では観測者はブラックホールの方向を見ており，図 5b では観測者は逆にブラックホールに背を向けている．

図 6 は，公園の石畳が極端に熱された状況で光線追跡を実行した例である．温度の上昇に伴い地表面付近の空気の屈折率が変化し，逃げ水現象が発生している．この場合，ハミルトニアンは次のようになる．

$$H = \frac{1}{2} n(x, y, z)(p_x^2 + p_y^2 + p_z^2) \qquad (17)$$

ここで $n(x, y, z)$ は，座標 (x, y, z) での屈折率を与える関数で，図 6 の作成時は

$$n = 0.85 - 0.3z^3 + 0.03\cos(3\pi x) \qquad (18)$$

と定義されている．なお，高さ方向 z 軸の原点はカメラの位置に設定されているので，地表付近では $z < 0$ となる．

実際のプログラミング例は次のようになる．例として，ハミルトニアンとして式 (5) を用いる場合を考える．屈折率は少なくとも座標の関数であるようにモデル化するのが普通であるが，今回は簡単のためにランダムなゆらぎで与えられるとする．このハミルトニアンを C 言語のサブルーチンにすると，例えば次のようになる：

```
autodiff hamiltonian_A(double *x)
{  autodiff px(x[0]);
   return(1.0/2.0*n()*px*px);
}
autodiff hamiltonian_B(double *x)
{  autodiff py(x[1]);
   return(1.0/2.0*n()*py*py);
}
```

ただし，autodiff は 3 章で述べたような C++言語のクラスとして実装されている自動微分型を表す．この簡単なモデルでは通常のオイラー法を適用すればシンプレクティック数値解法になるので，例えば次のようなサブルーチンを用意すれば良い：

```
euler_A(double *x,double tau)
{  x[0] = x[0]+tau*hamiltonian_A(x).diff;
   x[1] = x[1]+tau*hamiltonian_A(x).diff;
}
euler_B(double *x,double tau)
{  x[0] = x[0]+tau*hamiltonian_B(x).diff;
   x[1] = x[1]+tau*hamiltonian_B(x).diff;
}
```

精度を 2 次に上げるには，次のようなサブルーチンを用いる：

```
#define   D1  0.5
second(double *x , double tau)
{
  euler_A(x, D1*tau);
  euler_B(x, tau);
  euler_A(x, D1*tau);
}
```

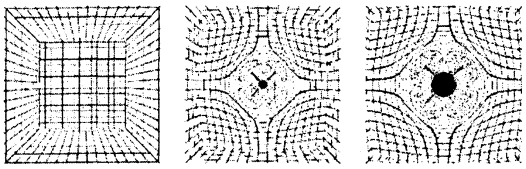
さらに，精度を 4 次に上げるには次のサブルーチンを使う：

```
#define     D2   1.351207191959657634
#define     D3    -1.702414383919315268
fourth(double *x, double tau)
{   second(x, D2*tau);
    second(x, D3*tau);
    second(x, D2*tau);
}
```

紙面の都合上，簡略化した部分もあるが，この例のようにそれほど複雑な実装によらずとも非線形現象のシミュレーションが可能である．

4.2 計算時間についての考察

アーキテクチャの異なるマシンでの計算時間を比較する．使用した計算機は，Intel Pentium III 800MHz のデュアル CPU マシン，Intel Xeon 2.8GHz のデュアル CPU マシン，MIPS R10000(250MHz) の SGI Onyx2，NEC SX-6 である．OS はいずれも UNIX 系で，コンパイラは Intel のプロセッサに対しては Intel C++，Onyx2 に対しては MIPSpro C++，SX-6 に対しては C++/SX を用いた．SX-6 では，横方向の光線の計算をベクトル化し，その計算を縦方向に繰り返すことで画像を生成した．生成画像は図 7 のようなもので，立方体状に壁で閉じられた空間を想定し，その中にブラックホールを発生させる状況を作った．ブラックホールの強さはパラメータ r_g で表され，(a) が $r_g = 0.00$，(b) が $r_g = 0.08$，(c) が $r_g = 0.12$ である．$r_g = 0$ の場合は光線は直進し，通常の光線追跡法と違いはない．$r_g > 0$ では光線が曲進し，演算回数が増加する．画像の大きさは 50 画素 ×50 画素と 100 画素 ×100 画素の 2 通りを作成した．表 1 の中で，PC A は Pentium III プロセッサのマシンを示し，PC B は Xeon プロセッサのマシンを意味する．Onyx2 は古い計算機であるが，ほぼ CPU の個数に比例した計算速度が得られており，SGI の共有メモリアーキテクチャの良い性能によるものと思われる．SX-6 を使用した場合，他の計算機と比べ優れている結果は得られていないが，処理のボトルネックとなる連立方程式の反復解法や光線とオブジェクトの交差判定の部分が十分にベクトル化されていず，プログラムのチューニングにより高速化可能と思われる．十分にベクトル化されていない原因は，チューニング済みのライブラリが自動微分法に対応していなかったからである．ベクトル型計算機への実装戦略としては図 8a のように光線の一本一本をベクトル演算によって計算していく手法も考えられるが，図 8b のように複数の光線の計算をベクトル演算によっ



(a) (b) (c)

図 7: 作成画像例

表 1: 計算時間測定結果（単位は分：秒）

$r_g = 0.00, 50 \times 50$				
CPU	PC A	PC B	SX-6	Onyx2
1	1:24	0:38	1:51	13:28
2	0:45	0:33	2:21	7:39
$r_g = 0.01, 50 \times 50$				
CPU	PC A	PC B	SX-6	Onyx2
1	5:42	2:08	13:42	57:01
2	3:05	1:28	5:58	28:19
$r_g = 0.00, 100 \times 100$				
CPU	PC A	PC B	SX-6	Onyx2
1	5:41	2:42	8:14	
2	3:08	2:17	11:45	
$r_g = 0.01, 100 \times 100$				
CPU	PC A	PC B	SX-6	Onyx2
1	22:51	8:48	34:38	
2	12:26	5:49	19:52	

て同時に実行する方が効率が良い．実際，ベクトル長を生成画像の画素数まで伸ばすことができ，チューニング不足とはいえベクトル化率は 99.6%を越えた．

現在のところ，生成画像中の各画素に対する光線追跡処理は，単純に左上から走査線状に下に向かって処理されているが，図 9 のように処理量は画素によって大きく異なり，順番に CPU を割り当てることは効率が悪い．図 9 は球対称ブラックホール時空で光線追跡をした場合，ブラックホールが存在する画面中央部では光線追跡期間が長くなり，数値誤差の蓄積が大きくなっていることを示している．このような場合，例えば横 1 ラインの単位で並列処理をすると，両端は計算が終了していても中央部が終了せず，同期処理のために次のラインの計算が始まるまで両端の処理をした CPU にアイドル状態が発生することが考えられる．従って，前処理として全画面から適当な分布で疎に光線追跡をし，その結果より画面中の画素の計算に必要な負荷の分布を予測して CPU を割り振る処理を検討している．

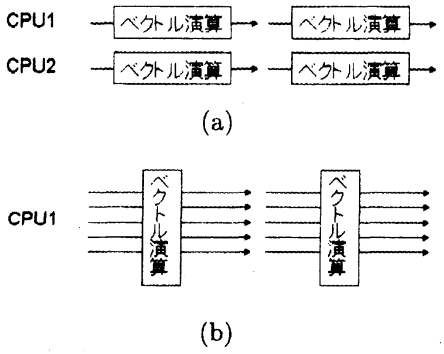


図 8: ベクトル型計算機への実装方法

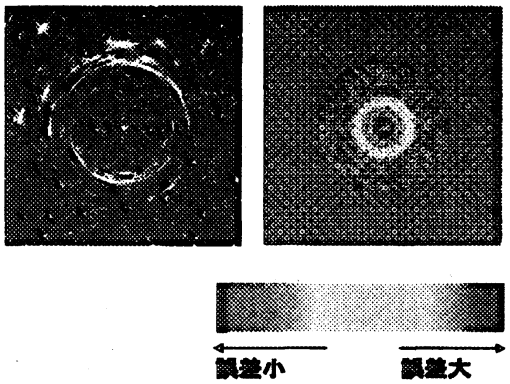


図 9: 誤差マップの作成

5 おわりに

本論分では，自動微分法とシンプレクティック積分法を用いて数値シミュレーションを行い，シミュレーション結果を用いて光線追跡法の概念に基づきコンピュータ・グラフィックスを生成する手法について述べた．現在のところ，適用対象が少ないこと，計算コストが高いことが問題として残されているので，適用できる現象や分野を検討し，また高精細な画像をリアルタイムに近い速度で生成することは可能かどうか検討する必要がある．

参考文献

[1] Whitted, T.: An Improved Illumination Model for Shaded Display, *Commun. ACM*, Vol. 23, No. 6, pp. 343–349 (1980).

[2] Gröller, E.: Nonlinear Ray Tracing: Visualizing Strange Worlds, *The Visual Computer*, Vol. 11, pp. 263–274 (1995).

[3] 山下義行: ブラック・ホールのコンピュータグラフィックス:光線追跡法の曲がった4次元時空への拡張, 情処学論, Vol. 30, No. 5, pp. 642–651 (1989).

[4] 福江純: ブラックホールを視る, 数学セミナー, Vol. 29, No. 11, pp. 44–48 (1990).

[5] Nollert, H. P., Kraus, U. and Ruder, H.: Visualization in Curved Spacetimes. I. Visualization of Objects via Four-Dimensional Ray-Tracing, *Relativity and Scientific Computing*, Springer-Verlag Berlin (1996).

[6] Weiskopf, D.: Four-Dimensional Non-Linear Ray Tracing as a Visualization Tool for Gravitational Physics, *Proc. IEEE Visualization 2000*, pp. 445–448 (2000).

[7] 斎藤泰, 牧野光則, 大石進一: レイトレーシング法を用いた異方性不均質透明体の表現, 信学論, Vol. J76-D-II, No. 8, pp. 1755–1762 (1993).

[8] Stam, J. and Languénou, E.: Ray Tracing in Non-Constant Media, *Proc. 7th Eurographics Workshop on Rendering*, pp. 225–234 (1996).

[9] 佐藤文隆, 小玉英雄: 一般相対性理論, 岩波書店, chapter 1: 多様体の力学, pp. 1–14 (1992).

[10] 原島鮮: 力学, 裳華房, chapter 14: ハミルトンの正準方程式, pp. 293–301 (1989).

[11] Iri, M., Tsuchiya, T. and Hoshi, M.: Automatic Computation of Partial Derivatives and Rounding Error Estimates with Applications to Large-Scale Systems of Nonlinear Equations, *J. Computational and Applied Mathematics*, Vol. 24, pp. 365–392 (1988).

[12] Griewank, A.: On Automatic Differentiation, *Mathematical Programming: Recent developments and Applications*, pp. 83–108 (1989).

[13] Hairer, E., Lubich, C. and Wanner, G.: *Geometric Numerical Integration–Structure-Preserving Algorithms for Ordinary Differential Equations*, Springer-Verlag Berlin Heidelberg (2002).

[14] Sanz-Serna, J.: Symplectic Intergrators for Hamiltonian Problems: An Overview, *Acta Numerica*, pp. 243–286 (1991).

[15] Yoshida, H.: Recent Progress in the Theory and Application of Symplectic Integrators, *Celestial Mechanics and Dynamical Astronomy*, Vol. 56, pp. 27–43 (1993).

[16] Suzuki, M.: Decomposition Formulas of Exponential Operators and Lie Exponentials with Some Applications to Quantum Mechanics and Statistical Physics, *J. Math. Phys.*, Vol. 26, No. 4, pp. 601–612 (1984).

[17] Yoshida, H.: Construction of Higher Order Symplectic Integrators, *Phys. Lett. A*, Vol. 150, pp. 262–268 (1990).

[18] 佐藤哲, 岩佐英彦, 竹村治雄, 横矢直和: シンプレクティック・レイトレーシング: ブラックホール 時空での光線追跡, 情処学論, Vol. 42, No. 3, pp. 456–464 (2001).

[19] 佐藤哲, 岩佐英彦, 竹村治雄, 横矢直和: 高速シンプレクティック・レイトレーシング: 入れ子宇宙の 可視化, 情処学論, Vol. 42, No. 10, pp. 2392–2402 (2001).

[20] Satoh, T. R.: Symplectic Ray Tracing: A new approach to non-linear ray tracing by using Hamiltonian dynamics, *Proc. SPIE-IS&T Electronic Imaging, Visualization and Data Analysis 2003*, Vol. 5009, pp. 277–285 (2003).