論　文

ユークリッド距離変換アルゴリズムの効率化

加藤　敏洋[†]　　平田　富夫[†]　　斉藤　豊文[†]　　吉瀬　謙二[†]

An Efficient Algorithm for the Euclidean Distance Transformation

Toshihiro KATO[†], Tomio HIRATA[†], Toyofumi SAITO[†], and Kenji KISE[†]

あらまし　本論文では，サイズが $N \times N$ の2値画像のユークリッド距離変換を $O(N^2)$ 時間で実行するアルゴリズムを与える．距離変換とは，入力として与えられた2値画像の各画素についてそこから最も近い0画素への距離を求める処理で，ディジタル画像処理における基本的な処理である．このアルゴリズムは，4近傍距離や8近傍距離などユークリッド距離以外の他の距離についてもアルゴリズム中の距離関数を置き換えるだけで距離変換が実行でき，その意味で一般的な距離変換アルゴリズムとなっている．また，$p(1 \leq p \leq N)$ 台のプロセッサを用意すれば $O(N^2/p)$ 時間の並列アルゴリズムが得られ，これまでの並列アルゴリズムより効率が良い．

キーワード　距離変換，画像処理，ユークリッド距離，並列アルゴリズム

1. まえがき

距離変換は，入力として与えられた2値画像の各画素についてそこから最も近い0画素への距離を求める処理で，ディジタル画像処理における基本的な処理である．スケルトン画像と組み合わせて図形の形状を抽出する処理やパターンマッチング，最近接点補間などに用いられる [9]．

距離変換の定義は Rosenfeld [11] らによって与えられた．そこでは距離として4近傍または8近傍による最小経路の長さ，すなわち4近傍距離または8近傍距離を用いている．ユークリッド距離ではなく4近傍距離や8近傍距離が用いられたのは，3×3 近傍に基づく局所処理の繰返しによって距離変換が容易に求まるからである．

4近傍距離や8近傍距離を用いた場合，1点からの距離が四角形状に広がるため，ユークリッド距離からのずれが大きい．そこで，4近傍と8近傍を交互に用いて8角形状に広げる8角形距離や，3×3 以上の近傍系を用いてユークリッド距離へのより精密な近似が図られてきたが，近傍を広げたことにより処理が複雑化するなどの問題も起こってきた．

一方，各画素に最も近い0画素との距離値を求めるだけでなく，最も近い0画素の座標値（ベクトル）も同時に伝搬することにより，ユークリッド距離を算出する方法も提案されている．山田 [14] は，ベクトルの伝搬を並列に行うとき8近傍による演算で誤差のないユークリッド距離変換が可能であることを示している．この方法によれば N^2 台のプロセッサを用いて $O(N)$ 時間で厳密なユークリッド距離変換を行う並列アルゴリズムが得られ，逐次アルゴリズムでは $O(N^3)$ 時間となる．但し，画像のサイズを $N \times N$ とする．また，ラスタースキャンを繰り返してベクトルを伝搬させることによりユークリッド距離変換を求める逐次アルゴリズムも提案されている [3], [8]．しかし，近似値しか求まらない画素が生じる場合があり，その意味で厳密なユークリッド距離は与えない．最近，Ragnemalm [10] は山田の方法を等高線スキャンと組み合わせて $O(N^2)$ 時間の逐次アルゴリズムを得たと報告している．等高線スキャンによる方法では，各0画素から同時に波を発生させ，その伝搬を追跡することで距離変換を行う．ただし，文献 [10] では実験により誤差がないことを主張しているが厳密な証明は与えられていない [1]．また，座標値の伝搬を行方向と列方向の処理に分解してハードウェア化と並列処理に適した形の実現法も提案されている [9]．

近年，斉藤，鳥脇 [12], [13] らは厳密なユークリッド距離変換を行うアルゴリズムを与えた．このアルゴリズムは入力画像によって時間計算量が変化し，とくに

† 名古屋大学工学部，名古屋市
Faculty of Engineering, Nagoya University, Nagoya-shi, 464-01 Japan

斜め線の入った画像の場合には，その計算量は$O(N^3)$となる．つまり，入力画像によっては，4近傍距離変換や8近傍距離変換の場合の計算量 $O(N^2)$ より効率が落ちる場合がある．基本的には文献[13]のアルゴリズムと同じであるが，Kolountzakis [7]らは，行方向の処理に分割統治法を導入し $O(N^2 \log N)$ のアルゴリズムを得ている．更に，Chen [2]らは行方向と列方向に交互に分割統治法を適用することによりこれを$O(N^2)$ 時間に改善している．また，Breu [1]らはボロノイ図に基づいた方法で $O(N^2)$ 時間のアルゴリズムを提案している．

本論文では，$O(N^2)$ 時間で厳密なユークリッド距離変換を行うアルゴリズムを与える．このアルゴリズムは，基本的には文献[2],[7],[12],[13]のアルゴリズムと同様に，行方向と列方向の処理に分解して距離変換を行う．本論文で提案するアルゴリズムは，列方向のスキャンは従来の方法と同じであるが，行方向のスキャンでは，N 個の関数の下側包絡線を求めることでその行の各画素に最も近い0画素への距離を知るという全く新しいアイデアを用いている．文献[1],[2],[10]のアルゴリズムに比べ計算量は同じであるが，入力画像のスキャン方向が行方向と列方向の2種類のため単純でプログラムとして実現するのが容易である．また並列処理にも適しており，共有メモリをもつ N 台のプロセッサで処理するとき，文献[2]のアルゴリズムは $O(N \log N)$ 時間を必要とするが，本論文で提案するアルゴリズムは $O(N)$ 時間で実行できる．更に，3次元画像の距離変換にも拡張が容易で，その時間計算量は $O(N^3)$ である．なお，このアルゴリズムは，4近傍距離や8近傍距離のようなユークリッド距離以外の距離についてもアルゴリズム中の距離関数を置き換えるだけで距離変換が実行でき，その意味で一般的な距離変換アルゴリズムとなっている．

以下，**2.**で本論文で必要となる定義を述べ，**3.**でユークリッド距離変換アルゴリズムを与える．また，3次元への拡張と並列化が容易であることを示す．**4.**では，このアルゴリズムが画像処理でよく用いられるその他の距離の場合にも（厳密な）距離変換を行うことを示す．**5.**はまとめである．

2. 準　　備

サイズが N 行 N 列(以下 $N \times N$ と略す)の2次元ディジタル画像において，第 i 行第 j 列の画素を (i,j) で表し，その濃度値が b_{ij} である画像を $\mathbf{B} = \{b_{ij}\}$ のように記述する．濃度値として0と1しかとらない画像を2値画像と呼び，値が0および1の画素をそれぞれ0画素，1画素と呼ぶ．

2値画像 $\mathbf{B} = \{b_{ij}\}$ のユークリッド距離変換画像 $\mathbf{D} = \{d_{ij}\}$ を次式で定義する．

$$d_{ij} = \min_{1 \leqq p,q \leqq N} \{\sqrt{(i-p)^2 + (j-q)^2} \mid b_{pq} = 0\} \tag{1}$$

すなわち，$\mathbf{D}$ は d_{ij} が入力画像 $\mathbf{B}$ の画素 (i,j) から0画素までの最小ユークリッド距離であるような画像である．$\mathbf{B}$ から $\mathbf{D}$ を求める処理を（ユークリッド）距離変換という．また，距離変換を施して得られる画像 $\mathbf{D}$ 自体も（$\mathbf{B}$ の）距離変換と呼ぶことがある．

3. 距離変換アルゴリズム

3.1 基本アルゴリズム

まず，本論文で与えるアルゴリズムの基本形を解説する．これは，文献[12]で提案されたアルゴリズムの2次元版である．$N \times N$ の入力2値画像 $\mathbf{B} = \{b_{ij}\}$ に対して次の二つの変換を順に実行する．

[変換1] $\mathbf{B} = \{b_{ij}\}$ に対し，各列 j ごとに列方向のみを参照して距離変換を行い画像 $\mathbf{G} = \{g_{ij}\}$ を求める．すなわち，

$$g_{ij} = \min_{1 \leqq p \leqq N} \{|i-p| \mid b_{pj} = 0\} \tag{2}$$

である．但し，0を含まない列 j については $g_{ij} = \infty (1 \leqq i \leqq N)$ とする．

[変換2] $\mathbf{G} = \{g_{ij}\}$ に対し，各行 i ごとに行方向のみを参照して画像 $\mathbf{D}' = \{d'_{ij}\}$ を求める．但し，

$$d'_{ij} = \min_{1 \leqq q \leqq N} \{(j-q)^2 + g_{iq}^2\} \tag{3}$$

である．

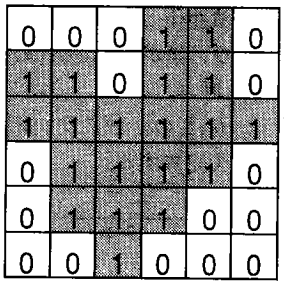
図1は入力画像 $\mathbf{B}$，変換1を実行した後の画像 $\mathbf{G}$，それに，画像 $\mathbf{D}'$ の例である．

次の補題が成り立つのは明らかである．

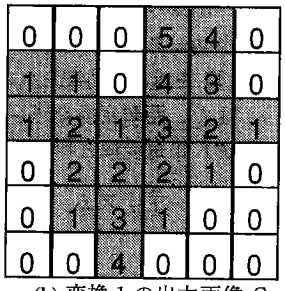
[補題1] 上で求められた画像 $\mathbf{D}'$ の各画素 $d'_{i,j}$ の値は，$\mathbf{B}$ において画素 (i,j) から最も近い0画素までのユークリッド距離を2乗した値である．

$\mathbf{D}'$ を2乗ユークリッド距離変換と呼ぶ．

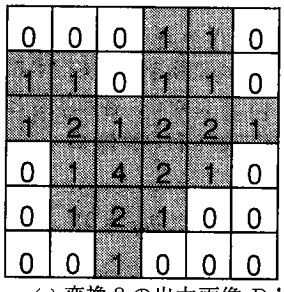
図2に $\mathbf{D}'$ を求めるアルゴリズムを示す．変換1を実行するのに，各列ごとに2回の走査を行っている．よって，変換1の時間計算量は $O(N^2)$ である．また，変換2では一つの d'_{ij} を求めるのに $O(N)$ 時間かかる



(a) 入力画像 B



(b) 変換 1 の出力画像 G



(c) 変換 2 の出力画像 D'

図1 ユークリッド距離変換の例
Fig. 1 An example of the Euclidean distance transform.

ので，時間計算量は $O(N^3)$ である．よって，アルゴリズム全体の時間計算量は $O(N^3)$ となる．なお，変換 2 を実行するには各画素ごとに次の処理を行ってもよい．

$$d'_{ij} \leftarrow \min_{-g_{ij} \le l \le g_{ij}} \{g^2_{i(j+l)} + l^2\}$$

但し，最小値の探索は l を順に変えることで行い，画像内 $(1 \le j+l \le N$ の範囲$)$ のみで探索する．このように，変換 2 は行方向の全画素を走査する必要はなく，(i,j) から距離 g_{ij} の範囲内のみを走査すればよいので，その時間計算量は $O(\mathbf{G}$ の画素値の平均値 $\times N^2)$ にできる．しかし，0 画素の数が少ない場合や画像中に図形がいくつも存在する場合には $\mathbf{G}$ の画素値の平均値が $O(N)$ になるため，最悪時間計算量はやはり $O(N^3)$ となる．

```
1: { 変換 1 }
2:   for j := 1 to n do
3:     begin
4:       temp := ∞;
5:       for i := 1 to n do
6:         begin
7:           if temp ≠ ∞ then temp := temp + 1;
8:           if b_ij = 0 then temp := 0;
9:           g_ij := temp;
10:        end;
11:      temp := ∞;
12:      for i := n downto 1 do
13:        begin
14:          if temp ≠ ∞ then temp := temp + 1;
15:          if b_ij = 0 then temp := 0;
16:          if temp < g_ij then g_ij := temp;
17:        end;
18:    end
19: { 変換 2 }
20:  for i := 1 to n do
21:    for j := 1 to n do
22:      begin
23:        d'_ij := ∞;
24:        for q := 1 to n do
25:          if g²_iq + (j − q)² < d'_ij
               then d'_ij := g²_iq + (j − q)²
26:      end
```

図2 ユークリッド距離変換アルゴリズム（基本形）
Fig. 2 A basic algorithm for the Euclidean distance transform.

文献 [13] ではこの基本アルゴリズムの改良版を提案している．これは，変換 2 の行走査における最小値の探索方法を改良することにより高速化を図ったものであるが，入力画像によっては行の走査の際に戻りがひんぱんに生じ，最悪時間計算量はやはり $O(N^3)$ となる．

3.2 変換 2 の新アルゴリズム

ここでは，変換 2 を効率良く実行する方法を与える．いま，i 行目の走査を考える．第 k 列に存在する 0 画素から (i,j) へのユークリッド距離の最小値は (i,k) に最も近い k 列の 0 画素 p から (i,j) への距離なので，$\sqrt{(j-k)^2+g^2_{ik}}$ である (図 3 参照)．この値の 2 乗を表す関数を j を変数として $f^i_k(j) = (j-k)^2 + g^2_{ik}$ と定義する．(図 3 において 2 次関数のグラフが $f^i_k(j)$ で，この関数の値は 0 画素 p から i 行目の各要素への距離の 2 乗を表している．) すると，すべての $k(1 \le k \le N)$ についての，この関数値の最小値

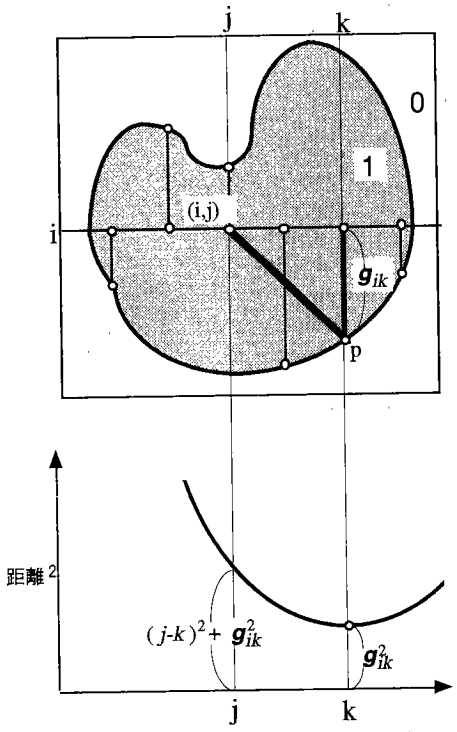


図3　0画素 p と i 行の各要素との距離
Fig. 3　Distance between a 0-pixel and pixels of the ith row.

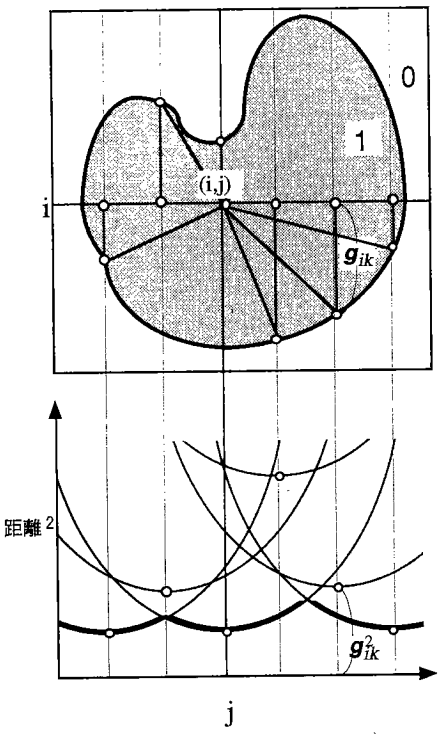


図4　下側包絡線
Fig. 4　A lower envelope.

が d'_{ij} である．つまり，$d'_{ij}=\min_{1\leqq k\leqq N} f_k^i(j)$ である．したがって，各行ごとに行う処理は，N 個の関数の集合 $F_N^i=\{f_k^i(x)\mid 1\leqq k\leqq N\}$ の下側包絡線 $\min_{1\leqq k\leqq N} f_k^i(x)$ を求めることに帰着する(図4)．以下では，文脈から明らかなときは $f_k^i(x)$ と F_N^i をそれぞれ $f_k(x)$ と F_N と書く．

$F_N=\{f_k(x)\mid 1\leqq k\leqq N\}$ に対し，その下側包絡線を与える関数の集合を $L_{\mathrm{env}}(F_N)$ と記述する．すなわち，

$$L_{\mathrm{env}}(F_N)=\{f_{i_1}(x),f_{i_2}(x),\cdots,f_{i_m}(x)\}$$

$$\min_{1\leqq k\leqq N} f_k(x)=\begin{cases} f_{i_1}(x) & (x<x_{i_1i_2}) \\ f_{i_2}(x) & (x_{i_1i_2}\leqq x<x_{i_2i_3}) \\ \quad\vdots & \\ f_{i_m}(x) & (x_{i_{m-1}i_m}\leqq x) \end{cases}$$

である．但し，x_{ij} は $f_i(x)$ と $f_j(x)$ の交点の x 座標である．次の補題が成立する．

[補題2] $i<j<k$ で $f_i(x)=(x-i)^2+b_i$，$f_j(x)=(x-j)^2+b_j$，$f_k(x)=(x-k)^2+b_k$ とする．このとき，(i) どの二つの関数もただ一箇所で交わる．(ii) 三つの関数の交点の x 座標を x_{ij},x_{jk},x_{ik} とすると，$x_{ij}\leqq x_{ik}\leqq x_{jk}$ または $x_{jk}\leqq x_{ik}\leqq x_{ij}$ である．

(証明) (i)は明らか．(ii)を示す．$f_i(x)$ と $f_k(x)$ の交点を (x_{ik},y_{ik}) とする．$f_j(x_{ik})\leqq y_{ij}$ のとき，$x_{ij}\leqq x_{ik}\leqq x_{jk}$ であり，$f_j(x_{ik})\geqq y_{ij}$ のとき，$x_{jk}\leqq x_{ik}\leqq x_{ij}$ である．□

補題2より，$f_1(x),f_2(x),f_3(x)\cdots$ の順に関数を加えながら下側包絡線を順次求めることができる．図5の例で示す．$f_1(x),f_2(x),f_3(x),f_4(x)$ の下側包絡線(を与える関数)が求まったとする(図5 (a))．次に，$f_5(x)$ を加えたとき，$f_4(x)$ と $f_5(x)$ の交点の x 座標 x_{45} が $f_3(x)$ と $f_4(x)$ の交点の x 座標 x_{34} の右にあれば $f_5(x)$ が下側包絡線を与える関数として加わるだけである．図5 (b)のように x_{45} が x_{34} の左にあるときには $f_4(x)$ は下側包絡線を与える関数ではなくなる．この例では，$f3$ も同様にして下側包絡線から除去され，結局 $\{f_1(x),f_2(x),f_5(x)\}$ が下側包絡線を与える関数となる．このことを次の補題3で述べておく．

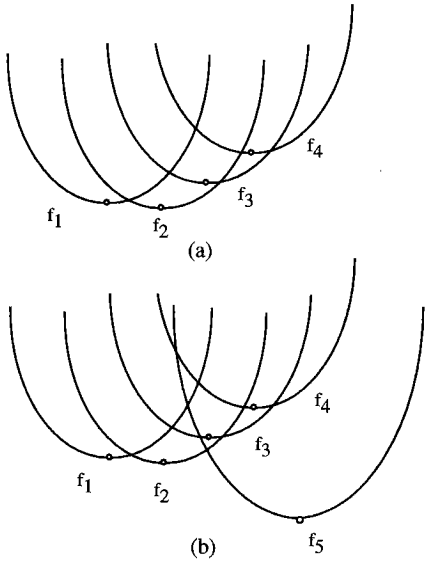


図5 下側包絡線の求め方
Fig. 5 Finding a lower envelope.

```
19: { 変換2 }
20: for i := 1 to N do
21:   begin
22:     空のスタック s を用意する;
23:     push(s,1); { スタック s に f1(x) を push}
24:     push(s,2); { スタック s に f2(x) を push}
25:     { 以下ではスタック s の先頭の2要素を
            t, t'(t' < t) とする }
26:     for j := 3 to N do
27:       if g_tj ≠ ∞ then
28:         begin
29:           while x_tj < x_t't do pop(s);
30:           push(s,j) { スタック s に f_j(x) を push}
31:         end;
32:     while N < x_t't do pop(s);
33:     for j := N downto 1 do
34:       begin
35:         if j < x_t't then pop(s);
36:         d_ij := f_i(j)
37:       end;
38:   end
```

図6 変換2の新アルゴリズム
Fig. 6 New algorithm for Transformation 2.

[補題3] $f_i(x) = (x-i)^2 + b_i, \quad (i = 1, 2, \cdots, N)$ とする．$F_k = \{f_i(x) \mid 1 \leqq i \leqq k\}$ の下側包絡線を与える関数の集合を $L_{\mathrm{env}}(F_k) = \{f_{i_1}(x), f_{i_2}(x), \cdots, f_{i_j}(x)\}(i_1 < i_2 < \cdots < i_j)$ とすると，$L_{\mathrm{env}}(F_{k+1})$ は次式で表される．

$$L_{\mathrm{env}}(F_{k+1}) = \begin{cases} L_{\mathrm{env}}(F_k) \cup \{f_{k+1}(x)\} & (x_{i_{(j-1)}i_j} < x_{i_j(k+1)}) \\ L_{\mathrm{env}}((L_{\mathrm{env}}(F_k) - \{f_{i_j}(x)\}) \cup \{f_{k+1}(x)\}) & (x_{i_{(j-1)}i_j} \geqq x_{i_j(k+1)}) \end{cases}$$

(証明) $x_{i_{(j-1)}i_j} < x_{i_j(k+1)}$ のときは明らか．$x_{i_{(j-1)}i_j} \geqq x_{i_j(k+1)}$ のとき補題2より，$x_{i_j(k+1)} \leqq x_{i_{(j-1)}(k+1)} \leqq x_{i_{(j-1)}i_j}$ である．よって，

$$f_{i_{(j-1)}}(x) \leqq f_{i_j}(x) \leqq f_{k+1}(x) \quad (x \leqq x_{i_j(k+1)})$$
$$f_{i_{(j-1)}}(x) \leqq f_{k+1}(x) \leqq f_{i_j}(x) \quad (x_{i_j(k+1)} \leqq x \leqq x_{i_{(j-1)}(k+1)})$$
$$f_{k+1}(x) \leqq f_{i_{(j-1)}}(x) \leqq f_{i_j}(x) \quad (x_{i_{(j-1)}(k+1)} \leqq x \leqq x_{i_{(j-1)}i_j})$$
$$f_{k+1}(x) \leqq f_{i_j}(x) \leqq f_{i_{(j-1)}}(x) \quad (x_{i_{(j-1)}i_j} \leqq x)$$

となる．従って，$f_{ij}(x)$ は F_{k+1} の下側包絡線を与える関数にはなりえない．よって，

$$L_{\mathrm{env}}(F_{k+1}) = L_{\mathrm{env}}(L_{\mathrm{env}}(F_k) \cup \{f_{k+1}(x)\})$$
$$= L_{\mathrm{env}}((L_{\mathrm{env}}(F_k) - \{f_{i_j}(x)\}) \cup \{f_{k+1}(x)\})$$

であることがわかる． □

補題3に基づいて変換2を効率よく実行するアルゴリズムを次のように構成する．先にも述べたように，変換2の各行 i ごとに行う処理は F_N の下側包絡線を求めることに帰着されるので，我々はまず $L_{\mathrm{env}}(F_N)$ を求める．初期設定を $L_{\mathrm{env}}(F_2) = \{f_1, f_2\}$ とし（簡単のため $g_{i1} \neq \infty, g_{i2} \neq \infty$ と仮定する．），$L_{\mathrm{env}}(F_3), L_{\mathrm{env}}(F_4), \cdots$ と順次求めていく．各時点で $L_{\mathrm{env}}(F_k)$ はスタックに格納される．それは，$f_{i_1}, f_{i_2}, \cdots, f_{i_j}$ の順にスタックに入るので，補題3で $x_{i_{(j-1)}i_j} \geqq x_{i_j(k+1)}$ のとき f_{i_j} を除去するのに，スタックの先頭を削除(pop)すればよいからである．$L_{\mathrm{env}}(F_N)$ がスタックに求まったら，スタック内の関数の値を各画素に代入する．図6にこのアルゴリズムを示す．

3.3 変換2の実行時間

上で提案した変換2のアルゴリズムは，画像の行ごとの処理に分解されている．各行ごとに，下側包絡線集合 $L_{\mathrm{env}}(F_N)$ を求め(図6の22～31行)，下側包絡線関数の値を各画素に代入している(32～37行)．$L_{\mathrm{env}}(F_N)$ を求めるときの時間計算量について考察する．スタックに対して施される演算回数に着目すると，

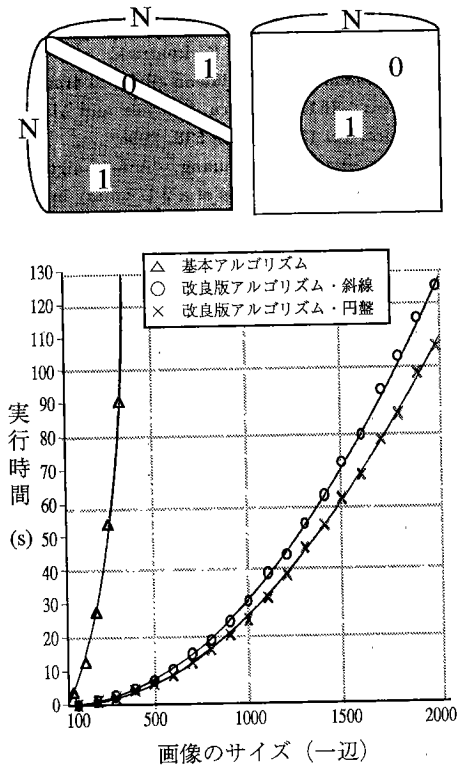


図7　斜線・円盤画像に対する実行時間
Fig. 7　Execution time for a slanting line and a disc.

pushはたかだか N 回なのでpopもたかだか N 回である．よって，各行ごとの処理の計算量は $O(N)$ で，変換2は $O(N^2)$ 時間でできることになる．従って，変換1が $O(N^2)$ でできることと合わせて，$O(N^2)$ 時間でユークリッド距離変換を求めることができることになる．

3.4　計算機実験

図2の基本アルゴリズムと本論文で提案したアルゴリズム（図2の19～26行を図6で置き換えたもの）を計算機上に実現し，ユークリッド距離変換の実行時間を測定した．入力画像は2種類で，一つは画像中に0画素による斜め線が1本存在する画像であり，もう一つは画像の中心に半径 $0.25N$ の円盤をおいた画像である．それぞれについて画像のサイズを 100×100 から 2000×2000 まで変えて計算時間を測定した．結果を図7に示す．なお，使用した計算機はSUN SS 10で，プログラムはC言語で作成した．

基本アルゴリズムはあきらかに $O(N^3)$ の曲線であり，本論文で提案するアルゴリズムは入力画像の内容によらず一定で $O(N^2)$ であることが確認できる．

3.5　3次元ディジタル画像の距離変換

上で与えたアルゴリズムは，3次元ディジタル画像のユークリッド距離変換にもすぐに拡張できる．入力の3次元ディジタル2値画像 $\mathbf{B}=\{b_{ijk}\}$ のサイズを $N\times N\times N$ とする．k を固定したときの2次元ディジタル画像 $\mathbf{B_k}=\{b_{ijk}\}$ に対し2次元ユークリッド距離変換を行った結果の画像を $\mathbf{D_k}=\{h^k_{ij}\}$ とする．これをすべての $k(1\leqq k\leqq N)$ について行うと，ここまでの処理は $O(N^2)\times N=O(N^3)$ 時間でできる．次に，i,j を固定してこれらの N 枚の距離変換画像を k 軸の方向にスキャンし，

$$d'_{ijk}=\min_{1\leqq q\leqq N}\{(k-q)^2+(h^q_{ij})^2\} \tag{4}$$

を求める．これは変換2の新アルゴリズム（図6の21～38行）を用いれば $O(N)$ 時間でできる．このスキャンをすべての $i,j(1\leqq i,j\leqq N)$ について行えば3次元画像のユークリッド距離変換が完了するので，このアルゴリズムの計算時間は $O(N^3)$ である．

3.6　並列アルゴリズム

本論文で与えたユークリッド距離変換アルゴリズムはスキャンの方向が行方向と列方向（3次元の場合はさらに k 軸方向）に固定しているため，等高線スキャンを用いた逐次アルゴリズムに比べ並列処理に向いている．すなわち，列方向の N 回のスキャンはそれぞれ独立しているので，$p(1\leqq p\leqq N)$ 台のプロセッサを用意して共有メモリに入力と結果の画像を書き込むことにすれば $O(N^2/p)$ 時間で実行できる．行方向も同様である．よって，2次元画像の場合，$p(1\leqq p\leqq N)$ 台のプロセッサで $O(N^2/p)$ 時間の並列アルゴリズムが構成できる．（3次元画像のときは $O(N^3/p)$ 時間の並列アルゴリズムが構成できる．）これに対しChen [2] らのアルゴリズムでは $O(N^2/p+N\log p)$ 時間を要する．つまり，プロセッサの台数が多いとき（$p\approx N$ のとき），文献[2]のアルゴリズムでは $O(N\log N)$ 時間であるが，本論文のアルゴリズムを並列化したものは $O(N)$ 時間で効率が良い．なお，ここでの並列計算モデルはEREW PRAMである．また，本論文のアルゴリズムは N^2 台で $O(\log N)$ 時間の並列化が可能なことが文献[4]で示された．

4.　他の距離への拡張

本論文で提案するアルゴリズムは，処理を列方向（変換1）と行方向（変換2）に分解して距離変換を

求めている．このような方法で正しく距離変換が求まるのは，ユークリッド距離の場合に限らない．

4近傍距離や8近傍距離など画像処理の分野で用いられる距離はほとんど軸方向の処理に分解して距離変換を行うことが可能である．

図6のアルゴリズムでは，2次関数の集合 $F_n = \{f_1(x), f_2(x), \cdots, f_n(x)\}$ の下側包絡線 $L_{\mathrm{env}}(F_n)$ と関数 $f_{n+1}(x)$ の交点を計算し $L_{\mathrm{env}}(F_{n+1})$ を求めている．このとき関数 $f_{n+1}(x)$ は他の関数 $f_k(x)$ $(k = 1, \cdots, n)$ のどれともたかだか1個しか交点を持たないことを利用している．4近傍距離と8近傍距離の場合も同じ性質をもっているので，本アルゴリズムによって距離変換を $O(N^2)$ 時間で求めることができる．距離関数を変更するだけでアルゴリズム自体を変更する必要がないので本論文で提案する距離変換アルゴリズムはきわめて汎用的である．具体的には，図6のアルゴリズムの2次関数 $f_k^i(j)$ を次のように置き換えればよい．

4近傍距離変換の場合 $\cdots$ $f_k^i(j) = |k-j| + g_{ik}$

8近傍距離変換の場合 $\cdots$ $f_k^i(j) = max(|k-j|, g_{ik})$

これらの関数について補題3が成立するのは容易に確かめられる．(4近傍距離や8近傍距離の場合に，$f_i(x)$ と $f_j(x)$ が一部で重なる場合があるが，そのときは一方をわずかにずらして交点を求めればよい．) このように，本論文で提案する距離変換アルゴリズムは，画像処理にあらわれるほとんどの距離関数に対し有効である．文献[5]ではそのような距離関数のクラスについて議論している．そのクラスは上で挙げた距離はもちろん，その他にチャムファー(chamfer)距離や8角形距離などをすべて含む極めて広いクラスである．

5. むすび

本論文では，$N \times N$ ディジタル2値画像のユークリッド距離変換を $O(N^2)$ で実行するアルゴリズムを提案した．このアルゴリズムは4近傍距離や8近傍距離のような他の距離にも適用可能であり汎用的なアルゴリズムである．また，容易に並列化でき，N 台のプロセッサで $O(N)$ 時間でユークリッド距離変換を計算できる．

謝辞 貴重なコメントを頂きました名古屋大学教授鳥脇純一郎先生に感謝致します．また，文献[2]と[7]を教えて頂いた奈良先端大の増沢利光先生と藤原さん，それに文献[1]を教えて頂いた大阪電通大の浅野哲夫先生に感謝いたします．

文献

[1] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman, "Linear time Euclidean distance transform algorithms," IEEE Pattern Analysis and Machine Inteligence, vol.17, pp. 529–533, 1995.

[2] L. Chen and H.Y.H.Chuang, "A fast algorithm for Euclidean distance maps of a 2-D binary image," Inf. Process. Lett., vol.51, pp.25–29, 1994.

[3] P.E. Danielsson, "Euclidean distance mapping," Comput. Graphics & Image Process., vol.14, pp. 227–248, 1980.

[4] A. Fujiwara, T. Masuzawa, and H. Fujiwara, "An optimal parallel algorithm for the Euclidean distance maps of 2-D binary images," 情処学アルゴリズム研報, AL-43, 1995.

[5] 平田富夫，加藤敏洋，"ユークリッド距離変換アルゴリズム，" 情処学アルゴリズム研報，AL-41, 1994.

[6] 加藤敏洋，斉藤豊文，平田富夫，"ユークリッド距離変換アルゴリズムの改良，" 平5東海連大，p. 369, 1993.

[7] M.N. Kolountzakis and K.N. Kutulakos, "Fast computation of Euclidean distance maps for binary images," Inf. Process. Lett., vol.43, pp. 181–184, 1992.

[8] F. Leymarie and M. D. Levine, "Fast raster scan distance propagation on the discrete rectangular lattice," CVGIP: Image Understanding, vol.55, pp. 84–94, 1992.

[9] D.W. Paglieroni, "Distance transforms: properties and machine vision applications," CVGIP: Graphical Models and Image Processing, vol.54, pp. 56–74, 1992.

[10] I. Ragnemalm, "Neighborhoods for distance transformations using ordered propagation," CVGIP: Image Understanding, 56, pp. 399–409, 1992.

[11] A. Rosenfeld and J. Pfalts, "Sequential operations in digital picture processing," J. Assoc. Comput. Mach., vol.13, pp. 471–494, 1966.

[12] 斉藤豊文，鳥脇純一郎，"3次元ディジタル画像に対するユークリッド距離変換，" 信学論(D-II), vol.J76-D-II, pp.445–453, 1993.

[13] T. Saito and J. Toriwaki, "Fast algorithms for n-dimensional Euclidean distance transformation," Proc. of 8th Scandinavian Conference on Image Analysis, vol. II, pp.747–754, 1993.

[14] H. Yamada, "Complete Euclidean distance transformation by parallel operation," Proc. of 7th Int. Conf. on Pattern Recognition, Montreal, vol.1, pp. 69–71, 1984.

(平成7年4月11日受付)

加藤　敏洋

平4名大・工・情報卒．平6同大大学院修士課程了．在学中，画像処理変換アルゴリズムの効率化に関する研究に従事．

平田　富夫　（正員）

昭51東北大・工・通信卒．昭56同大大学院博士課程了．工博．豊橋技科大助手を経て，現在，名大・工・電子工学科教授．昭59～60英国ウォーリック大学客員研究員（ブリティッシュカウンシルスカラー）．グラフアルゴリズム，データ構造の研究に従事．著書「アルゴリズムとデータ構造」（森北出版）．

斉藤　豊文　（正員）

昭61名大・工・電子卒．平3同大大学院博士課程了．同年4月名古屋大学工学部助手．平7同講師．パターン認識，画像処理に関する研究に従事．ME学会会員．

吉瀬　謙二

平6名大・工・電子卒．現在，東大大学院修士課程在学中．