コンピュータビジョン 24－4
（1983. 5. 19）

大規模画像に対する細線化アルゴリズムのパイプライン方式による効率化

An Efficient Thinning Algorithm for Large Scale Images Based upon Pipe-line Structure

中山 晶　木村文隆　吉田雄二　福村晃夫
Akira NAKAYAMA　Fumitaka KIMURA　Yuuji YOSHIDA　Teruo FUKUMURA .
（名古屋大学工学部 ： Faculty of Engineering, Nagoya University ）

Abstract

In this paper, we propose an efficient thinning algorithm for large scale images based upon pipe-line structure. This algorithm is derived from conventional thinning algorithms with a few modifications. Then thinned image is equivalent to that of the original conventional algorithm.

The algorithm was applied to several images and compared its performance with those of conventional algorithms. From experimental results, it was shown that our algorithm is more efficient than conventional ones.

1 まえがき

大規模な画像データを細線化する場合、通常とられる方法として、画像を幾つかの領域に分割して各領域毎に処理を行い、その後でそれぞれの処理結果をはり合わせる方法[1]がとられたり、画像を格納するための2つの補助記憶ファイルを設けて、2つのファイル間に細線化処理を行うためのバッファ領域を設け、画像データをバッファを通してファイル間を往復させるという手法[2]がとられる。しかし、これらの手法では、処理手順が煩雑となったり、主記憶と補助記憶間の転送回数が多くなるため処理時間のオーバーヘッドが増大するなど有効な方法とは言い難い。

一方、このような難点を改良した大規模画像データに対する効率的細線化法にWoetzelの方法[3]やその他の方法[4]があるが、用いられる細線化アルゴリズムが特殊なため、処理に要する反復回数が多かったり、細線化結果が十分に優れたものではないなどの欠点が見られる。

本論文では、与えられた画像に含まれる図形の幅の上限があらかじめ知られている場合に、その幅の半分よりやや広い幅の範囲を処理対象として画像を一方向に一度走査することで完全な細線化図形が得られる方式を提案する。

この方式（以下では、処理対象となる部分を保持する領域をパイプとみてパイプライン方式と呼ぶ）は、従来の画像全体を何度も走査する種々の方法に比べて、処理時間が短縮されると共に必要な主記憶容量が大幅に削減される。従来の細線化アルゴリズムはある条件を満せば、この方式のアルゴリズムに変更することが容易に可能であり、適用範囲の広い手法となっている。

本方式の採用により、従来通常の手段では不可能とみられていたミニコンや最近の16bit マイクロ・プロセッサといった高機能小型計算機でも従来のアルゴリズムの性質を保存した形で手軽に大規模画像が扱えるようになる。特に、本方式は、ファクシミリなどラスター走査型装置から走査線データを逐次与えられて実時間で処理を行う装置には極めて有効な手法となっている。

以下、2では、本方式が適用可能な基本となる細線化アルゴリズムの性質

について述べ，次いで3で本方式のアルゴリズムとその正当性について述べる。4で本方式によるアルゴリズムのソフトウェア実現とそれに基づく従来のアルゴリズムとの比較実験の結果を示す。最後に，5で今後の課題を述べる。

2 細線化アルゴリズムの性質

逐次型で一定方向走査型の細線化アルゴリズムでは，一回の画面走査で図形境界を一画素分だけ全方向から消去するため，消去に要する走査回数（サイクル数）は，画像中の図形の（4連結の意味での）最大線幅をWとすると

$$R = \lfloor W/2 \rfloor$$

で与えられる。このような型のアルゴリズムとしてはHilditch のアルゴリズム[6]，鷲岡のアルゴリズム[1]などがある。これらのアルゴリズムでは，細線化に際して，3×3のマスクにより画像 $A = \{a_{ij}\}$ をラスター走査しマスクの中央点 a_{ij} をその8近傍状態によって分類しながら消去条件を満す画素を消去するという逐次型局所変換 $a_{ij} = f(a_{ij})$ を施すことで連結した心線を得ている。

この型のアルゴリズムの特徴は，逐次型の長所を最大限に活す配慮がアルゴリズムの構成でされている一方で，図形境界の判定には一回前の処理値を併用することで1パス全方向消去を可能にしている点にある。

この型のアルゴリズムでは，画像Aは全て主記憶上にあるとしている。寸法がN行L列の画像Aの第i行の画素列を

$$a_i = (a_{i1}, a_{i2}, \cdots\cdots a_{iL}) \quad ; (i=1,2,\cdots,N)$$

で表し，画素列 a_i に対する変換を $F(a_i)$ と表すことにすると画素列 a_i は最終的な細線化結果を得るために終了判定走査を含めて $(R+1)$ 回の変換下を要する。すなわち，これは画面走査の結果，全画面上で図形が全く変化しなくなったときにアルゴリズムが停止するとしているためである。この変換下は，a_{i-1} と a_{i+1} の値を参照しながら実行される。

3 パイプライン方式による細線化

3.1 アルゴリズム

細線化を行うため，画像Aに対してFIFO型のバッファ領域，

$$B = \{b_{ij} \mid 0 \le i \le R+1,\ 0 \le j \le L+1\}$$

を図1のように設定する。また，Bの各行を $b_0, b_1, \cdots\cdots b_{R+1}$ で表す。

以下に，ここで提案するパイプライン方式による細線化アルゴリズムを示す。

このアルゴリズムでは，画像Aは行単位で逐次バッファに読み込まれ，マスクの走査を伴うF変換をB内のみで原アルゴリズムとは逆方向に行いながら細線化結果を行単位でバッファから出力するという方式をとる。

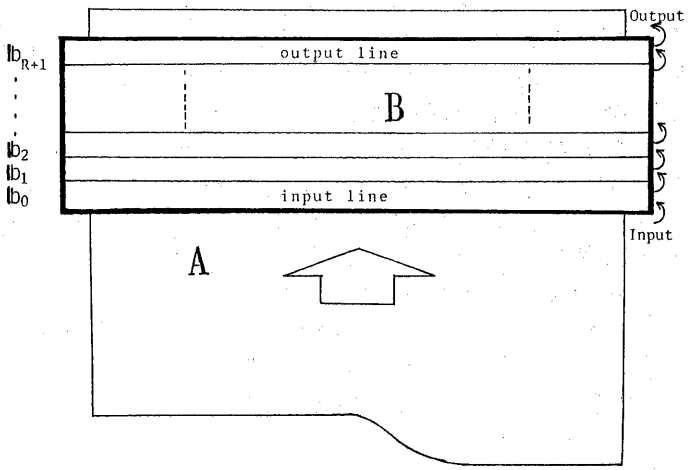

図1 バッファの設定

〔アルゴリズム〕

step 1（初期化）$r \leftarrow 1$とし，IB内の全ての要素を0とする。

step 2（データの入力）IB内の一番目の行IB_0へ画像の第r行目a_rを読み込む。このとき，Aの最終行a_Nが既に読み込み済みなら，IB内のデータを逐次掃き出すため$IB_0 = (0, 0, \cdots\cdots 0)$とする。

step 3（逐次的下変換）IB内における次の各行

$IB_1, IB_2, \cdots\cdots\cdots, IB_R$

に対して，この順に下変換を適用し，変換によって変化しない行があればそこで変換を終了する。

step 4（データの出力）IBの最上位の行IB_{R+1}の内容を細線化結果として出力する。このとき，IB_{R+1}の内容が，step 1における初期値の0要素列の場合は出力しない。

step 5（データの転送）IB内の各行の内容を

$IB_{i+1} \leftarrow IB_i \quad (i = 0, 1, 2, \cdots\cdots, R)$

として全体を一段上へシフトする。

step 6（終了の判定）Aの行データがIB内に存在しなければ，アルゴリズム終了。そうでなければ，次のデータを読み込むため，$r \leftarrow r+1$として再びstep 2以下を実行。 ■

このアルゴリズムでは，Aの全ての行要素列は，バッファIBの最下位行に入力され，順次上の行へシフトされて最上位行に達して出力されるまでに，最大R回の下変換を施され，最終的な細線化結果を得る。

原アルゴリズムが画面中で図形の変化がなくなるまで全ての行要素a_iは必ず（R+1）回の下変換を必要としたのに対し，本アルゴリズムでは，最大線幅をもつ部分が入力されたときのみIB内の範囲で最大R回の下変換が行われる。従って，入力されるデータの状態によって，バッファIB中のデータに対する下変換は最小1回，最大R回で済むことになる。

ここに，IBは連続したデータの系列

$(a_1, a_2, \cdots\cdots\cdots, a_N)$

に対して，IB内を通過する線幅に応じてR個のプロセッサが働く一種のパイプラインとみなすことができる。このアルゴリズムでは，線幅Wによってきまる遅れ時間で行単位の処理結果が得られる。この時間々隔が極めて小さければ，スキャナーなどの標本化時間を伴う待ち時間を有効に利用できるため実時間処理を効果的に行うことができると考えられる。

対象画像が線状画像であれば，画像の一辺の長さLに比べてRは十分小さい値となることから，バッファIBの大きさはLにのみ依存した線形増加となり，原アルゴリズムを直接に実行するときの主記憶容量に比べて大幅な記憶容量の削減が可能となっている。

3.2 アルゴリズムの正当性

ここでは，前節のアルゴリズムの正当性の概略について述べる。すなわち，本方式によるアルゴリズムの実行結果が基本となる原アルゴリズムの実行結果と同一となることを示す。

正当性の証明に先だって，次のような準備を行う。画像AとバッファIBは図2に示すような関係で，

$A = (a_1, a_2, \cdots\cdots\cdots, a_N)$

$IB = (IB_0, IB_1, \cdots\cdots\cdots, IB_{R+1})$

なる行単位の列として表わされているものとし，基本となるアルゴリズムでk回目の処理が終った後のi行（画素列）の内容を$a_i^{(k)}$で表す。また以下では，$a_i^{(0)} \triangleq a_i \quad (i = 1, 2, \cdots\cdots, N)$

とする。さらに，任意の行cが，その上の行b，その下の行dの状態で下変換された結果，aとなるとき，これを

$a = F(c \mid b, d)$

のように表す。この記法を用いれば，基本となるアルゴリズムの逐次処理の各段での処理結果について，次の性質が成り立つ。

(i) $a_i^{(n)} = F(a_i^{(n-1)} \mid a_{i-1}^{(n)}, a_{i+1}^{(n-1)})$

(ii) 任意の lb，dl について
$\mathcal{O} = F(\mathcal{O} \mid lb, dl)$

(iii) 画像 A に対するアルゴリズムのサイクル数が R のとき
$a_i^{(n+1)} = a_i^{(n)}$ $(n \geq R)$

ここに，$i = 1, 2, \cdots, N$ および $n = 1, 2, \cdots, R$ であり，lb, $\mathcal{C}$, dl は任意の画素列，$\mathcal{O}$ は
$(0, 0, \cdots\cdots, 0)$
なる0要素の画素列である。また，正当性の証明を簡潔にするため前節のアルゴリズムの Step 3 の下変換は $lb_1, lb_2, \cdots, lb_R$ なる全ての行に対して適用されるものとする。

〔証明の概略〕

初めに，バッファに画像が第1行（最下位行）から逐次読み込まれ，本アルゴリズムによる処理が進み，バッファの最上位行（lb_{R+1}）に画像の第1行に対する処理結果が入っていて，走査を終了し，その行を出力しようとしている時を考える。

この状態では，バッファの内容は，次のようになっていることが知られる。

$lb_{R+1} = a_1^{(R)}$, $lb_i = a_{R-i+2}^{(i)}$
$; (i = 0, 1, 2, \cdots\cdots, R)$ ----- (1)

すなわち，バッファの最上位行は，対応する画像の行の R 回の走査後の内容に一致し，バッファの各行 lb_i $(i = 0, 1, \cdots\cdots, R)$ の内容は，現在処理対象となっている画像の行に対する i 回の走査終了時の内容に一致している。このことは，バッファの内容はその初期状態（第1回目の走査を開始する直前）では，
$(a_2^{(1)}, a_1^{(0)}, \mathcal{O}, \mathcal{O}, \cdots\cdots, \mathcal{O})$
となっていて，第1回目の走査（下変換）で性質(i)により $a_1^{(0)}$ が上位行 $\mathcal{O}$，下位行 $a_2^{(0)}$ の状態で，
$(a_2^{(0)}, a_1^{(1)}, \mathcal{O}, \cdots\cdots, \mathcal{O})$
となる。次に B の内容の上位シフトと $a_3^{(0)}$ の lb_0 への読み込みにより
$(a_3^{(0)}, a_2^{(0)}, a_1^{(1)}, \mathcal{O}, \cdots\cdots, \mathcal{O})$
となり2回目の走査で，性質(i)から，
$(a_3^{(0)}, a_2^{(1)}, a_1^{(2)}, \mathcal{O}, \cdots\cdots, \mathcal{O})$
となる。以下同様に進めていけば，式(1)のようになることは容易に知られる。このことから，本アルゴリズムにより出力される第1行の値は，$a_1^{(R)}$ であることが知られる。

次に，処理が進み，バッファが画像の第 n 行から第 $n-R-1$ 行に対応す

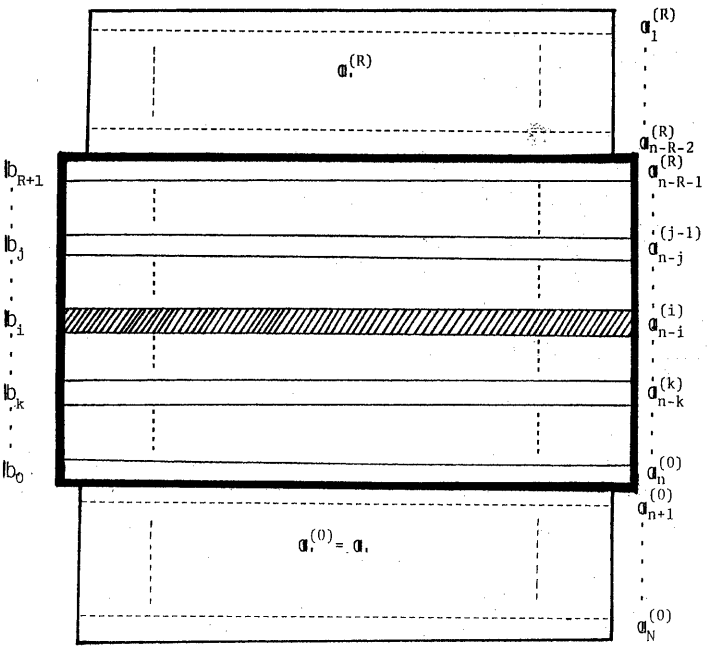

図2　バッファの位置と処理内容
（行 lb_i に対する走査が終了した直後の状態）

る内容を保持（図2の状態）していて，それらに対する走査（下変換）を終了した時点で，バッファの内容について

$$b_R = a^{(R)}_{n-R-1},\quad lb_i = a^{(i)}_{n-i}$$

$$;(i=0,1,2,\cdots\cdots,R) \cdots\cdots\cdots (2)$$

が成り立っているとする．すなわち，バッファの内容と，対応する画像の部分に対する処理内容とについて式(1)と同様の関係が成り立っているとする．

この時，次の第$n+1$行から第$n-R$行の処理に移ると，第$n-R-1$行の処理結果が出力され，内容が1行分シフトされて，第$n+1$行目が読み込まれるため，バッファの内容は，

$$lb_i = a^{(i-1)}_{n-i+1},\quad lb_0 = a^{(0)}_{n+1}$$

$$;(i=1,2,\cdots\cdots,R+1)$$

となる．この状態でバッファの各行（$lb_1, lb_2, \cdots\cdots, lb_R$）に対し下変換を施すとその内容は性質(i)から，

$$lb_{R+1} = a^{(R)}_{n-R},\quad lb_i = a^{(i)}_{n-i+1}$$

$$;(i=0,1,2,\cdots\cdots\cdots,R) \cdots\cdots (3)$$

となり，やはり同様の性質が保存されることが知られる．

以上の事から，本アルゴリズムにより出力される結果は，画像の全ての行について$a^{(R)}_i$ $(i=1,2,\cdots\cdots,N)$となって基本となるアルゴリズムの実行結果と同一となる．（証明終り）

なお，この証明にあたって，前節のアルゴリズムのstep 3における下変換を$lb_1, lb_2, \cdots\cdots, lb_R$なる全ての行に適用することを前提としたが，これは性質(iii)を忠実に適用したからである．実際には，ある段でのバッファ内の走査でバッファの第i行以上の各行

$(lb_i, lb_{i+1}, \cdots\cdots, lb_{R+1})$

が全く変化せず，かつバッファの内容の上位シフト後の走査で，第i行(lb_i)が変化しないならば，その段での$lb_{i+1}, \cdots\cdots, lb_{R+1}$もやはり変化しないことが言える．この性質を利用すれば，前段でのバッファの内容が変化しなくなった位置（上述のiの値）を記憶しておけば，走査をバッファ全体に対して適用しないで途中で打ち切ることが可能となる．3.1に述べたアルゴリズムはこの性質に基づいている．

また，厳密には，上述の証明に，バッファの内容が，画像の下位部分（$R+2$行未満）を含んでいる状態，すなわち，

$(0, \cdots\cdots, 0, a^{(0)}_N, a^{(1)}_{N-1}, \cdots\cdots a^{(k)}_{N-k})$

という状態にある場合についても触れる必要があるが，同様の方法で扱えるので証明を簡略にするため省略した．

4 アルゴリズムの実現と比較実験

前章の方式によるHilditchのアルゴリズムをFORTRAN言語でソフトウ

表1　比較実験の結果

データ＼項目	大きさ（最大線幅）	アルゴリズム（＊本方式による）	記憶容量（K Byte）	バッファラインの数	時間（Sec） CPU	時間（Sec） 1ライン当り
道路線図	1200 X 2048 (5)	Hilditch*	36	4	143.4	0.12
		Tsuruoka*	36	4	122.7	0.10
		Woetzel	63	8	903.7	0.75
気象図	500 X 500 (20)	Hilditch*	33	12	242.9	0.48
		Tsuruoka*	33	12	180.0	0.36
		Woetzel	53	23	1535.9	3.07

（注）使用計算機 FACOM 230-38

ェア実現した。実現にあたって，バッファBを図3に示すようなシリンダー状データ構造で表現することで，step5のデータ転送を不要にするよう構成した。実現したプログラムはステップ数約90行，プログラム領域約1.7K Byteで実現されている。実現したプログラムの例を付録に示す。

Hildizch アルゴリズムの他，逐次型の一定方向走査型アルゴリズムである鴫岡のアルゴリズムを本方式で同様に実現し，Woetzelのアルゴリズムと共に比較実験した結果を表1に示す。なお表での実行記憶容量はプログラム領域も含んだ容量である。

この結果から，本方式による結果がWoetzelの方法よりも実行記憶容量で約1.7倍，実行時間で約6.3倍の効率化が画られている。この結果は，Woetzelのアルゴリズムでは，Bの大きさ(L方向)が(W+3)の大きさを必要とし，細線化がマスク・パターンのマッチングを基本とする処理を行っているためと考えられる。

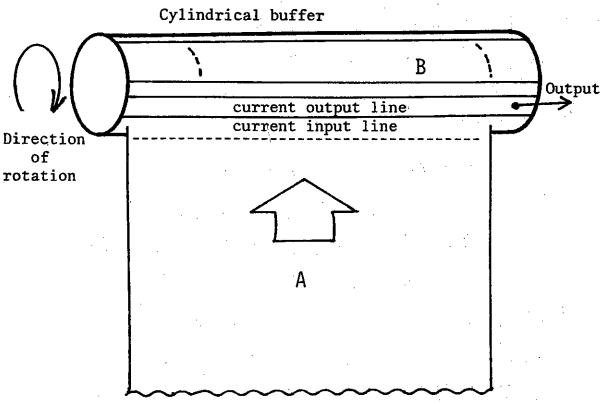

図3 シリンダー状データ構造

また，Hildizchのアルゴリズムよりも鴫岡のアルゴリズムの方が実行時間の点で優れているが，これは原アルゴリズムより周辺雑音消去の部分を省いて実現を行っているためであり，線幅が大きい場合は不要な枝成分が多数発生する。しかし，線幅が十分小さい場合は少なく最も効率的となる。

本方式による1ライン当りの平均処理時間は，Hildizch の場合 0.30(秒)，

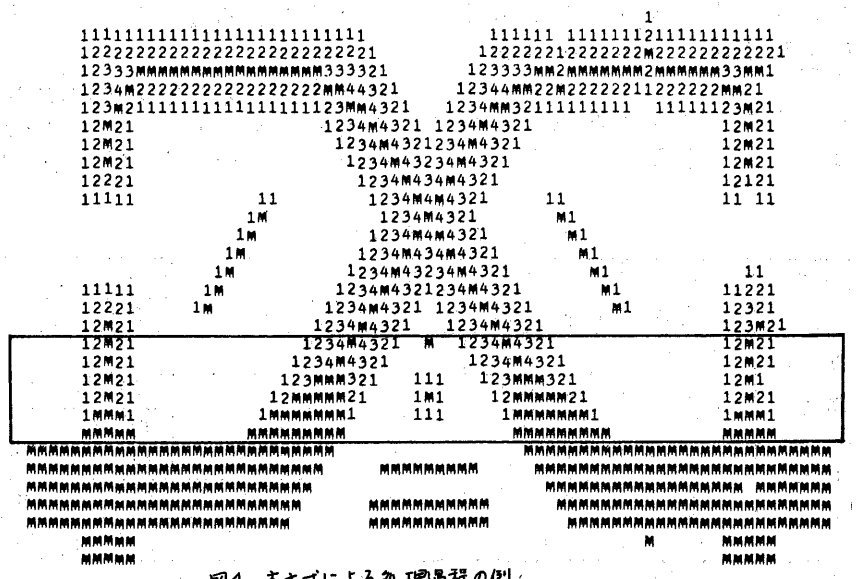

図4 本方式による処理過程の例

鶴岡の場合 0.23(秒)となりスキャナーなどの標本化時間を伴う待ち時間に十分追従可能な値となっている。この値は，本方式によるアルゴリズムの実現が行要素列の0，1パターンをそのまま走査するという方式をとっているためであり，1要素の位置の記憶とその数をカウントするIBと同じ大きさの領域を別に設けてIBと連動させれば，線状画像では1要素に比べて0要素が圧倒的に多い特徴をもつことから更に高速処理が可能であり，並列化を考慮したハードウェア化を含めて十分実用的な値になると考えられる。ちなみに，本方式による鶴岡のアルゴリズムを1要素のみにアクセスする方法で表1の道路線図に対して実行した所，処理時間110.8秒であり1ライン当り，0.09となり，表1の結果に比べて2.5倍の効率化となっている。

なお，本方式による原アルゴリズムの実行と修正を加えない原アルゴリズムとの実行を表1の気象図に対して比較した所，記憶容量で約17倍，処理時間で約1.5倍の効率化となっている。

一方，本方式によるHilditchアルゴリズムの処理過程の例を図4に示す。図では，直線で囲った部分がバッファの内部を表しており，この段階でのF交換が終了した時点を示している。ここに，画素に対応する数字は，その画素がバッファ内の何段目の走査で消去されたかを示す番号（labelと）を表しており，原アルゴリズムのサイクル数に直接対応している。この結果からもバッファより出力された細線化結果は修正を加えない原アルゴリズムの処理結果と等価な結果を与えていることが知られる。このような結果は，本方式による鶴岡のアルゴリズムでも得られており，本方式が原アルゴリズムの性質を保存した適用範囲の広い手法であることを示している。

5むすび

既存の細線化アルゴリズムをパイプライン方式に修正することにより効率よい処理が行われる手法を提案した。本方式による細線化アルゴリズムの実行は、従来提案された効率的細線化法であるWoetzelの方法よりも効率的で、また適用範囲の広い手法であることを幾つかの実験結果によって示した。

本方式による原アルゴリズムの実行は細線化処理ばかりでなく局所反復型処理を行う一般の画像処理アルゴリズム[8]にも適用可能であると考えられる。アルゴリズムの一般化を含めた理論的検討と本方式による適用ならびにその応用などは残された課題である。

謝辞　日頃，御指導賜る豊橋技術科学大学本多波雄教授ならびに本学福垣康喜教授および本稿をまとめるに当り有益な助言をいただいた本学横井茂樹助教授に深謝する。また，Woetzelのアルゴリズムに関する文献の提供をいただいた静岡大学鈴木智氏に感謝します。

（参考文献）

(1) Tamura, H: "Image database management for pattern information processing studies" Lecture Notes in Computer Science, Ed. Goos, G., Hartmain, J., PP. 198-227 (1980)

(2) 坂井，吹抜："パターン認識装置の基本設計"，信学会，オートマトンと自動制御研資（1961）

(3) Woetzel, G; "A Fast and Economic Scan-to-line Conversion Algorithm", SIGGRAPH, 12, 13, PP. 128-129 (1978)

(4) 岡田他："水平，垂直線要素の分類を用いた線順序形細線化法"，信学論 Vol. J64D-No5, PP.403-410 (1981)

(5) 田村："細線化法についての諸考察"信学研資，PRL75-66, PP.47-56 ('75)

(6) Hilditch, C. J.: "Linear skele-

ton from square curboad", in Machine Intelligence 6, B. Meltoger & D. Michine, Eds. Univ. Press Edinburgh, PP 403-420 (1967)

(7) 鶴岡他："ディジタル2値図形の一細線化法"，信学研資，PRL 78-47, PP 41-47 (1978)

(8) 鳥脇他："画像処理のアルゴリズム"，情報処理，Vol. 21, No 6, PP. 613-619 (1980)

(9) 大下他："ドラム型画像入力装置の性能評価実験"情報処理研資，コンピューター・ビジョン8-5, PP. 1-8 (1980)

(付録)

```
      SUBROUTINE SLIM8H(NR,NW,IBUF,LEND,K)
C ***************************************************************
C *                                                             *
C *                  **   PARAMETER CONTENTS   **               *
C *                                                             *
C *         NR ---------- IDENTICAL NUMBER OF INPUT FILE        *
C *         NW ---------- IDENTICAL NUMBER OF OUTPUT FILE       *
C *         IBUF ---------- RASTER BUFFER (SIZE = LEND*K)       *
C *         LEND ---- LENGTH OF LINE-DATA (WORD NUM.) + 2       *
C *         K -----  HALF VALUE OF MAXIMAM LINE WIDTH + 2       *
C *                                                             *
C ***************************************************************
      INTEGER IBUF(LEND,K),NA(9),NB(9),R
      DATA NPK,NUM,MAX/0,0,1/
C -------------------------------------------------------------
      NP=K-2
      NSP=K-1
      NSK=2*NSP-1
      LED=LEND-1
      DO 10 J@=1,K
      IBUF(1,J@)=0
   10 IBUF(LEND,J@)=0
      DO 20 I@=2,LED
   20 IBUF(I@,NSP)=0
C ============================
   30 NP=NP+1
      JP=MOD(NP,K)+1
      IF (NPK.NE.0) GO TO 40
C ############################
      READ(NR,END=40)
     +   (IBUF(IP,JP),IP=2,LED)
C ############################
      GO TO 60
   40 NPK=NPK+1
      DO 50 IP=2,LED
      IBUF(IP,JP)=0
   50 CONTINUE
   60 LMAX=0
      DO 160 R=1,MAX
      ID=0
      J=JP-R
      IF (J.LE.0) J=J+K
      JP1=MOD(J,K)+1
      JM1=J-1
      IF (JM1.EQ.0) JM1=K
C ----------------------------
      DO 150 I=2,LED
      IF (IBUF(I,J).LE.0)
     +                  GO TO 150
      NA(1)=IBUF(I+1,  J)
      NA(2)=IBUF(I+1,JM1)
      NA(3)=IBUF(I  ,JM1)
      NA(4)=IBUF(I-1,JM1)
      NA(5)=IBUF(I-1,  J)
      NA(6)=IBUF(I-1,JP1)
      NA(7)=IBUF(I  ,JP1)
      NA(8)=IBUF(I+1,JP1)
      DO 80 M=1,8
      IF (NA(M).NE.-R)
     +                  GO TO 70
      NA(M)=1
      NB(M)=0
      GO TO 80
   70 IF (NA(M).LT.0) NA(M)=0
      NB(M)= NA(M)
   80 CONTINUE
      NA(9)=NA(1)
      NB(9)=NB(1)
      N4=NA(1)+NA(3)+NA(5)+NA(7)
      IF (N4.EQ.4) GO TO 150
      NP8=0
      NS8=0
      DO 90 M=1,8
      IF (NA(M).EQ.1) NP8=NP8+1
      IF (NB(M).EQ.1) NS8=NS8+1
   90 CONTINUE
      IF (NP8.LE.1) GO TO 150
      IF (NS8.EQ.0) GO TO 150
      NC=0
      DO 100 M=1,4
      NC=NC+(1-NA(M*2-1))*
     +     MAX0(NA(M*2),NA(M*2+1))
  100 CONTINUE
      IF (NC.NE.1) GO TO 150
      IF (IBUF(I,JM1).NE.-R)
     +                  GO TO 120
      NA(3)=0
      NC=0
      DO 110 M=1,4
      NC=NC+(1-NA(M*2-1))*
     +     MAX0(NA(M*2),NA(M*2+1))
  110 CONTINUE
      IF (NC.NE.1) GO TO 150
      NA(3)=1
  120 IF (IBUF(I-1,J).NE.-R)
     +                  GO TO 140
      NA(5)=0
      NC=0
      DO 130 M=1,4
      NC=NC+(1-NA(M*2-1))*
     +     MAX0(NA(M*2),NA(M*2+1))
  130 CONTINUE
      IF (NC.NE.1) GO TO 150
  140 ID=1
      IBUF(I,J)=-R
  150 CONTINUE
C ----------------------------
      IF (ID.EQ.0) GO TO 160
      IF (R.GT.LMAX) LMAX=R
  160 CONTINUE
      MAX=LMAX+1
      IF (NP.LE.NSK) GO TO 30
      J@=MOD(JP,K)+1
C ################################
      WRITE(NW)
     +   (IBUF(I@,J@),J@=2,LED)
C ################################
      IF (NPK.EQ.NSP) RETURN
C ================================
      GO TO 30
      END
```

(注)
このプログラムでは，消去画素に対応する配列要素に負の値が入るため，必要であれば出力直前に負の値を0に変換する後処理を行う。なお，連結心線に対応する要素は全て1の値として保存される。