グラフィクスとCAD 60－11
(1992. 12. 17)

手書きインタフェースのための ペンの囲みによる対象判定アルゴリズムの実現と評価

佐藤 俊，村瀬敦史，中川正樹

東京農工大学工学研究科電子情報工学専攻

〒184 東京都小金井市中町 2-24-16 東京農工大学工学部電子情報工学科

あらまし 表示一体型タブレット上において，ある対象を指定する方法として対象をペンで囲むことに注目した．対象が囲まれているか否かは，対象の代表点（中点，全点など）と囲み線との包含判定を行うことにより決定する．本報では，対象の代表点が囲み線の内か否かの包含判定を行うアルゴリズムを提示する．また，このアルゴリズムには，人間の手書きによる多様な囲み方に対応し，かつ高速性が要求される．本報では，基本となるアルゴリズムを述べ，さらに高速化を図るために考案した，囲み線を直線補間によって処理するアルゴリズムとセグメント分割によって処理するアルゴリズムを述べる．また，それぞれのアルゴリズムについて，囲まれる対象の代表点数を変え処理時間の測定を行った．

和文キーワード 手書きインタフェース，表示一体型タブレット，対象包含判定，塗りつぶしアルゴリズム

The Implementation and Evaluation of Algorithms to Determine Pattern Inclusion within Pen Movement Enclosures for Handwriting Interface

Takashi Satou, Atsushi Murase, Masaki Nakagawa

Department of Computer Science, Faculty of Technology, Tokyo University of Agri. and Tech.

Department of Computer Science, Faculty of Technology, Tokyo University of Agri. and Tech., Naka-cho 2-24-16, Koganei-shi, Tokyo, 184

Abstract For the specification of patterns we have considered encircling patterns with a pen on a display integrated tablet. The determination of pattern inclusion is done with representation points (a center point, all points, etc) of each pattern. This paper presents algorithms to determine whether a representation point of a pattern is within a pen movement enclosure. These algorithms are required to deal with variable kinds of pen movement enclosures by handwriting and to respond quickly.

In order to be able to process more quickly than the basic algorithm, we devised two algorithms processing pen movement enclosures by straight line approximations and by segmentations. With each algorithm we measured processing speed according to variable representative points of patterns within pen movement enclosures.

英文 key words Handwriting interface, Display integrated tablet, Pattern inclusion, Coutour filling algorithm

1．はじめに

現在，計算機を用いた文書の作成において，文字の入力にはキーボード，図の入力にはマウスの使用が主流である．しかし，これらの方法は，慣れるまでにかなりの時間を必要とし，また，使い勝手がよいとはいえない．人は幼年時から，文字や図を書くときには紙と鉛筆を用いている．表示一体型タブレットは，紙と鉛筆の感覚で文字や図を手書き入力する環境を提供する．

現在，表示一体型タブレットの開発によって，ペンによる文字入力や作図システムの研究・開発が進められている．システム本体の設計同様，認識処理後の誤認識の訂正，および，削除・移動・複写などの編集のインタフェースは重要である．

現在のインタフェースでは，対象を含む矩形の左上と右下をポイントしたり，対象上を直接ポイントする方法が大部分を占めている．しかし，これらの方法は，ユーザが意識した対象を自然に指定できなかったり，一度に多くの対象を指定できないという欠点がある．そこで，我々は，上記の欠点がなく，かつ人間に自然な方法として，対象を囲むことに注目した（図1）．

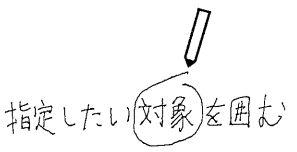


図1．対象の囲み

本報では，ある対象がペンによって書かれた囲み線の内か否かを高速に判定するアルゴリズムを提示する．

2．判定について

この章では，判定アルゴリズムに要求される事柄と，対象が囲み線の内か否かを判定する条件について述べる．

2．1 アルゴリズムへの要求

対象が囲み線の内か否かを判定するアルゴリズムに要求される事柄を次に述べる．

(1) 多様な囲み方に対応

手書きによる囲み動作は不安定で，多様な囲み方が存在する．例えば，囲み線が閉じない，囲む回転方向が一定しない，自己交差が起こる，何重にも書かれる，などが挙げられる（図2）．

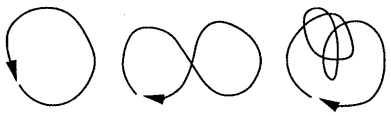


図2．多様な囲み方の例

(2) 処理の高速性

ヒューマンインタフェースの分野では，よくレスポンスタイムについての問題が取り上げられる．本アルゴリズムについても，ユーザにとってわずらわしさを感じさせない高速性が要求される．

2．2 対象が囲み線の内とは

ここでの対象とは，ペンダウンされてからペンアップされるまでの時系列で入力された離散的な筆点であるストロークを指している．したがって，対象が囲み線の内か否かを判定するには，対象を代表する点（代表点）の集合として囲み線の内か否かの判定を行う．代表点の集合には，対象を表現するのに十分な点の集合をとる．対象によっては，1点（例えば中点）の場合もあれば，全点の場合もある．

以下，“囲み線の内か否かの判定”を“包含判定”と表現する．また，囲みによって指定の対象となる部分を“領域”という．一般の塗りつぶしアルゴリズムによる領域と，ここで扱う領域とは異なるので，ここで扱う領域を図3に示す．

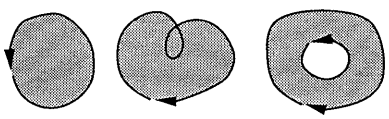


図3．領域

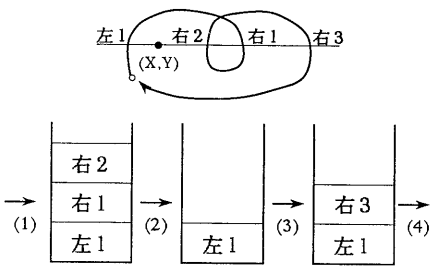
3．代表点の包含判定アルゴリズム

既存の塗りつぶしアルゴリズムでは，空間コヒーレンスが利用できるのに対し，オンライン入力された対象は離散的な筆点の集合なので，そのままオンラインの環境に応用することはできない．したがって，離散的なＹ座標値の水平スキャンが必要とされる．

この章では，上記の要求を満たすアルゴリズムと，それを考案する動機および基本となったアルゴリズムについて述べるとともに，それぞれの処理速度の比較を行う．

3．1 基本アルゴリズム

囲み線を始点から終点まで追跡し，それぞれの筆点間と対象の代表点（X，Y）のＹのスキャンラインとの交点を調べていく．交点を見つけた場合，交点のｘ座標値が代表点のＸ座標値の右（ｘ座標値の大きい方）にあるか左（ｘ座標値の小さい方）にあるかをスタックに push down する．同じ側が push down されたなら，その二つの状態を pop up する．最後にスタックが空ではなく，残りが偶数の場合には代表点が領域内であると判定する．この包含判定の動作手順について例を挙げて図４に示す．



(1) “左１” “右１” “右２” の push down で同じ側が連続.
(2) “右１” と “右２” を pop up.
(3) “右３” の push down．交差点の検出終了.
(4) スタックの残りが偶数 → 領域内.

図４．基本アルゴリズムの包含判定

3．2 直線補間によるアルゴリズム

前述したアルゴリズムは，代表点の包含判定を行うごとに囲み線の筆点間とＹのスキャンラインとの交点を検索しなければならない．初めに囲み線に何らかの処理を施すことにより，代表点の包含判定にかかる負担を著しく軽減できると考えた．

ここでは，囲み線の処理で直線補間を用いるアルゴリズムについて述べる．

3．2．1 判定の基本

表示一体型タブレットから入力された筆点座標は，整数値のタブレット座標（X_t，Y_t）であり，表示のために，整数値の表示座標（X_d，Y_d）に変換される．我々の環境での対応を次に示す．

$X_d = f \times X_t$
$Y_d = f \times Y_t$
$f = 1000/2048$

図５は，表示座標系で表された，ある対象の代表点（X，Y）に対する囲み線を表す．この代表点（X，Y）が領域内か否かを判定するには，囲み線とＹのスキャンラインとの交点座標（X_i，Y）と（X_j，Y）から，「$X_i \leqq X \leqq X_j$」であるかを調べればよい．

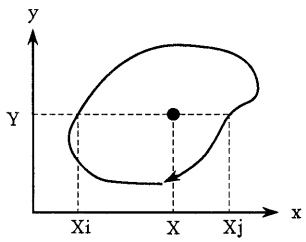


図５．判定の基本

これから述べるアルゴリズムは，オンラインの特性を利用し，かつ，これ以上の応用の効くものである．

以下，すべての座標値は表示座標系の値としている．

3．2．2 囲み線の処理

囲み線を示す離散的な筆点列（X_1，Y_1），（X_2，Y_2），…，（X_n，Y_n）が与えられたとする．閉領域を作成するために（X_n，Y_n）は（X_1，Y_1）に連結し，（X_0，Y_0）＝（X_n，Y_n）とする．この筆点列の隣合う筆点間において，直線補間し各y座標値に対応するx座標値を計算する．そして，スキャンテーブルのyにxの値をソートしながら登録し，同時に，運筆が上向きか下向きかの情報も登録する．直線補間による囲み線の状態を図6に，あるYのスキャンラインの例を図7に示す．

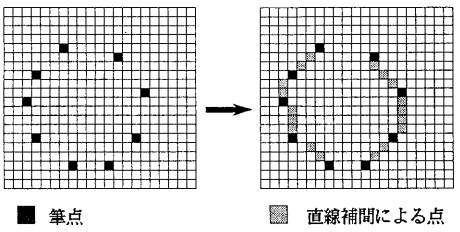


図6．直線補間による囲み線の状態

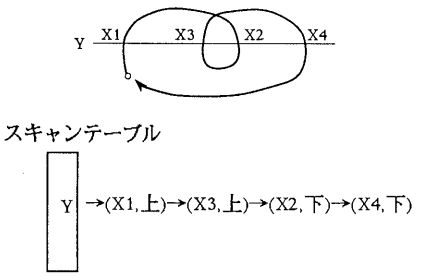


図7．あるYのスキャンラインの例

3．2．3 代表点の包含判定

対象の代表点（X，Y）に対し包含判定を行う場合，スキャンテーブルのYの情報でx座標値の小さい方から順にスタックにpush downする．連続して（X_i，上）と（X_j，下）または（X_i，下）と（X_j，上）がpush downされたなら，「$X_i \leqq X \leqq X_j$」を調べて，真なら領域内と判定する．偽ならその二つをpop upして，同様の処理を繰り返す．スタックにpush downする情報がなくなったなら領域外と判定する．上記のフローチャートを図8に示す．

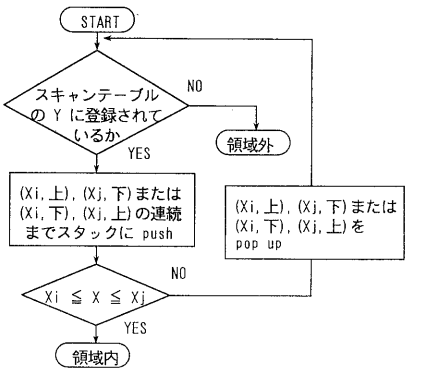
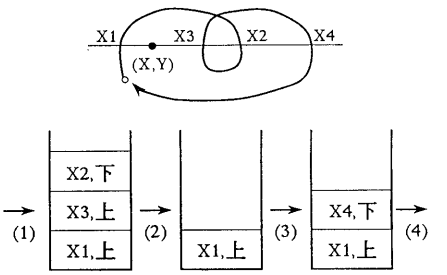


図8．包含判定のフローチャート

また，ある囲み線での判定処理を図9に示す．



(1) (X1,上)，(X3,上)，(X2,下)のpush downで“上”と“下”が連続．
(2) X3≦X≦X2→偽．(X3,上)，(X2,下)をpop up．
(3) (X4,下)のpush downで“上”と“下”が連続．
(4) X1≦X≦X4→真．領域内と判定．

図9．直線補間によるアルゴリズムでの点の包含判定

このアルゴリズムには，囲み線を何本書いても判定ができるという特徴がある．内側と外側の囲み線の向きを変えることにより，ドーナツ

状の領域も指定できる．

3．3 両アルゴリズムでの処理速度の測定

両アルゴリズムに従い，C言語でプログラミングして代表点の包含判定を行うツールを作成した．次に，それぞれの処理時間の比較測定を行う．

現在，我々が使用している表示一体型タブレットは，入力解像度：10 line／mm，ペン座標のサンプリングレート：120 point／sec，表示解像度：約 5 line／mm，表示サイズ：縦 166 mm×横 236 mm の仕様である．

実行環境は，20 MHz の 68020 を CPU とするワークステーションで，本研究室で開発した OS／omicron を使用している．

最初に，処理の本質的な速度を比較するために，囲む点数を 150，279，…と増やし，点の増加に伴う処理時間を測定した．その測定結果を図 10 に示す．

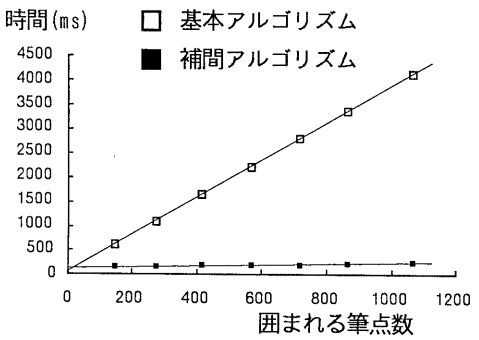


図 10．囲まれた点数の増加に伴う処理時間の測定

次に，このツールを実際にアプリケーションにのせるときの処理時間の測定も行う．

ストロークが指定されるという最も簡単な方法は，ストロークの中点などの代表点一点の包含判定を行えばよい．しかし，図 11 のフローチャートで，四角の中の文字列を四角の（人間にはどこかがわからない）代表点を含めずに囲むことは，人間にとって非常に困難である．したがって，ストロークに対し，全筆点，あるいは，ある割合以上の代表点が囲み線で囲まれたかを判定することにより，人間の負担を軽減させることができる．

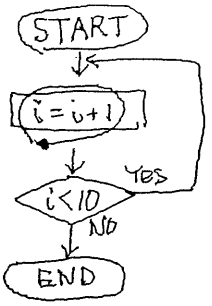


図 11．フローチャートでの文字列の囲み

全筆点の包含判定では，一点でも領域外と判定されたなら，即座に“含まれない”という結果を返せるので，全筆点を検索するわけではない．判定の論理を図 12 に示す．

```
for ( i = 1; i <= m; i++ )
  if ( 1点が領域内か否か( Xi, Yi ) != 領域内 )
    return( ストロークは領域外 );
return( ストロークは領域内 );
```

(Xi, Yi)：判定対象の代表点（1≦i≦m）．

図 12．包含判定を全代表点で行う判定論理

それに対し，ある割合以上の代表点の包含判定では，全筆点を検索しなければならない．図 10 の測定結果から，“ある割合”を設定して包含判定する場合，直線補間によるアルゴリズムの有効性の高さが確認できる．

現時点では，実用的な“ある割合”が設定できないので，オンライン入力された，図や表を含む，総ストローク数 1434 本の対象集合に対し，囲むストローク数の全筆点の包含判定における処理時間を測定した．その測定結果を図 13 に示す．

図 10，図 13 から，直線補間によるアルゴリ

ズムでの処理は，基本アルゴリズムの処理よりも数倍速いことが確認できた．これは，最初に囲み線の処理を行うことによって，代表点の判定時間を格段に低減させているためである．

4．囲み線の処理を行うアルゴリズム

直線補間によるアルゴリズムの囲み線の処理

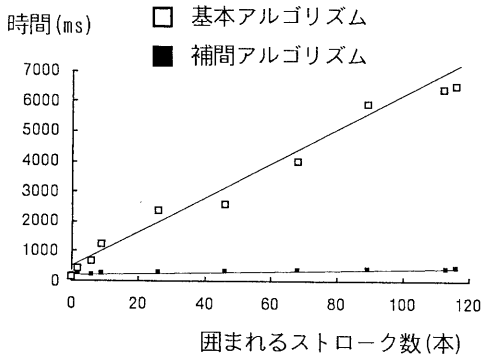


図 13．全筆点の包含判定での処理時間の測定

は，直線補間によって表示ドット単位で計算する．したがって，対象を大きすぎる囲み線で囲んだ場合に，不必要なYのスキャンラインまで登録してしまい，無駄が生じる可能性がある．

この章では，囲み線の処理と代表点の包含判定の処理の関係を考察するために，囲み線の処理を軽減させるアルゴリズムについて述べるとともに，直線補間によるアルゴリズムとの処理時間の比較を行う．

4．1　セグメント分割によるアルゴリズム

4．1．1　囲み線の処理

囲み線の筆点列から，極大・極小を判定して分割したものをセグメントという．つまり，時系列順の筆点間の方向（上向きか下向き）を調べていき，同じ方向あるいは水平方向が続く筆点列を一つのセグメントとする．また，セグメントの検出順に番号付けを行い，各セグメントには，次の三つの情報を登録する（図 14）．

(1) 最小点と最大点のｙ座標値
(2) 方向（上向きか下向き）
(3) 各筆点の座標値

4．1．2　代表点の包含判定

次に，対象の代表点（X，Y）の包含判定について述べる．まず，セグメント番号順に「セグメントのｙの最小値≦Y≦セグメントのｙの最大値」を調べる．真であるセグメントに対しては，二項探索によりＹのスキャンラインと筆点間の

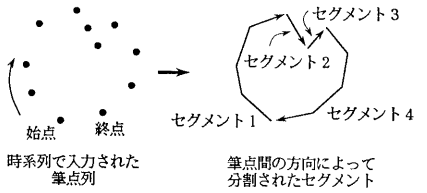
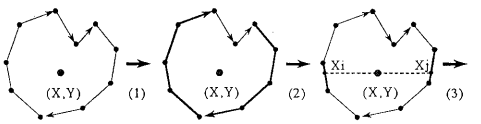


図 14．囲み線のセグメント分割

交差区間を検出する．その交差区間において，Ｙのスキャンラインとの交点のｘ座標値を計算し，ｘの値でソートしながら（ｘ，方向）をリストにつなげていく．すべてのセグメントの検出が終了したとき，直線補間によるアルゴリズムの判定と同様に，リスト情報とスタックを用いて判定する．以上の手順を図 15 に示す．



(1)「セグメントのｙの最小値≦Y≦セグメントのｙの最大値」を満たすセグメントを調べて，セグメント１とセグメント４が検出される．
(2) 検出されたセグメントに対して，筆点間とＹのスキャンラインとの交差区間を二項探索によって調べる．
(3) 交差点の情報（Xi,上）と（Xj,下）をリストにつなげる．Xi≦X≦Xj → 真．領域内と判定．

図 15．セグメント分割によるアルゴリズムでの代表点の包含判定

4．2　直線補間とセグメント分割によるアルゴリズムでの処理速度の測定

この測定には，前述した測定と同じデータを用い，囲まれる点数の増加に伴う処理と全筆点の包含判定による処理について時間の測定を行った．その結果をそれぞれ図16，17に示す．

セグメント分割による囲み線の処理は筆点単位で行うので，表示ドット単位で行う直線補間よりも処理の負担がかなり軽減されるように思われる．実際に，囲み線には画面広く書く場合と，小さく書く場合の二種類を用意し，点の包含判定を行わないで，囲み線だけの処理時間の測定を行った．その結果を表1に示す．

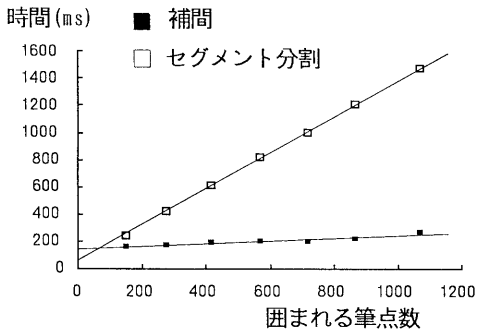


図16．囲まれた点数の増加に伴う処理時間の測定

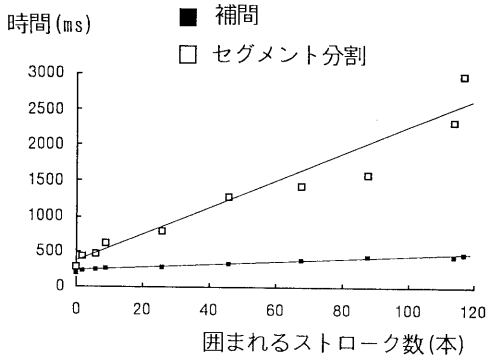


図17．全筆点の包含判定での処理時間の測定

表1．囲み線の大きさによる処理時間

	囲み線小	囲み線大
直線補間	0.031	0.187
セグメント分割	0.015	0.062

単位：秒

この表の結果から，囲み線の処理時間だけでは，セグメント分割の処理は速い．しかし，図16，17の結果から，囲み線の処理の負担の軽減によって，包含判定全体の処理の負担を軽減させるわけではないことがわかる．包含判定全体の処理時間を，囲み線の処理時間a，一点を判定する処理時間b，判定する筆点数nを用いて表すと次の式になる．

全体の処理時間 $=a+b\times n$

つまり，包含判定全体の処理時間として，nの数が非常に少ないときにはaの影響が強いが，nの数が増えるほどbの影響がいっそう強くなる．

以上の結果から，ストロークの代表点一点だけを指定の対象とした場合には，セグメント分割によるアルゴリズムでも有効であるが，判定する点数の増加に伴い直線補間によるアルゴリズムの有効性は著しく向上する．

5．おわりに

表示一体型タブレット上で，対象を指定する方法として，手書きで対象を囲むことに注目した．そして，手書きによる多様な囲み方に対応するアルゴリズムを三種類提案し，動作するツールを作成した．それぞれの処理速度を比較測定したところ，直線補間によるアルゴリズムは最も高速で，精密な判定に優位であることがわかった．

我々は，直線補間によるアルゴリズムを用いたツールが，多くの手書きアプリケーション上で応用されることを期待する．

参考文献

(1) Cyrus , M . and Beck , J . : Generalized Two−and Three−Dimensional Clipping , Computers & Graphics Vol . 3 , pp . 23～28 (1978) .

(2) Foley , J . D . and Van Dam , A . : Fundamentals of Interactive Computer Graphics , Addison−Wesley , Reading , Mass . (1982) .

(3) 中前栄八郎：コンピュータグラフィックス，オーム社，pp . 81～83 (1987) .

(4) 日本工業規格：コンピュータグラフィクス中核系 (GKS) 機能記述 JIS X 4201 , 日本規格協会 (1990) .

(5) Rogers , D . F . : Procedural Elements for Computer Graphics , McGraw−Hill , NewYork (1985) .

(6) Sutherland , E . E . and Hodgeman G . W . : Reentrant Polygon Clipping , Comm . ACM , 17 , 1 , pp . 32～42 (1974) .

(7) Vatti , B . R . : A Generic Solution to Polygon Clipping , Comm . ACM , 35 , 7 , pp . 56～63 (1992) .