力学モデルによるループの自己干渉除去アルゴリズム

1E－1－1

土井　淳*,　山田　敦*,　伊藤　貴之*

*日本アイ・ビー・エム（株）東京基礎研究所
神奈川県大和市下鶴間 1623-14
doichan@jp.ibm.com, ayamada@jp.ibm.com, itot@jp.ibm.com

本報告では，平面上に定義された稜線のループの自己干渉を，力学モデルに基づいて除去していく手法を提案する．平面上に定義された稜線のループに，自己干渉が存在するかどうかをバウンディングボックスを用いた再帰的な判定法によって効率的に判定する．自己干渉が存在するとき，グラフを用いてループの自己干渉の位置関係を表現し，グラフの情報に基づいて自己干渉を除去していく．このときもとの形状に近くなるように力学モデルを用いる．本手法により，位相構造を変更せずに自己干渉を除去しもとの形状に近い形状を出力できる．

A physically-based approach to eliminate self-intersections

Jun Doi*, Atsushi Yamada*, Takayuki Itoh*

*IBM Japan LTD., Tokyo Research Laboratory
1623-14, Shimotsuruma, Yamato, Kanagawa
doichan@jp.ibm.com, ayamada@jp.ibm.com, itot@jp.ibm.com

This paper presents an algorithm to eliminate self-intersections using physically-based approach. This paper also shows an efficient algorithm to detect self-intersections using bounding boxes. We use a graph to represent connectivity between sub-regions of caused by self-intersections and eliminate self-intersections by traversing this graph. We use a physically-based approach to avoid deformation during elimination process.

1　はじめに

CADやCGの分野において，形状の変形や変換によってや，浮動小数点演算の演算誤差によってなどの理由で，形状に望ましくない自己干渉が生じることがたびたび起こる．自己干渉によって，その後の処理に不都合が生じたり，場合によっては幾何的な矛盾が生じるなどして処理が破綻してしまう可能性があり，自己干渉によって処理が不安定になってしまう．

本報告では，平面上に定義された稜線のループについての自己干渉の有無を効率よく判定する手法と，自己干渉を力学モデルに基づいて除去する手法について述べる．

ここで，ループに自己干渉が生じる以下のような例を紹介する．

- 3次元の形状処理を2次元の平面に投影して解く場合におこる自己干渉

・ 演算誤差による自己干渉
・ 曲線のループを多角形で近似する場合に生じる自己干渉
・ モーフィングの途中形状に生じる自己干渉

3 次元の形状処理において，問題を簡単にするために 2 次元の平面上に形状を投影したものを用いて問題を解くことが多い．3 次元空間内に定義された曲面に，メッシュを生成する問題では，曲面を 2 次元の平面に投影して 2 次元の曲線のループとして問題を解くという手法[1]が用いられている．このとき，曲面の状態や投影の方法によっては，投影されたループに自己干渉が発生してしまう場合がある．メッシュ生成の処理に自己干渉がある場合，内外判定の処理がうまくいかないために自己干渉している部分に幾何的な矛盾が生じてメッシュ生成処理が破綻してしまう問題がある．

CAD における変換などの数値演算では，一般的に浮動小数点演算が用いられているが，浮動小数点演算には丸めや桁落ちなどの誤差が付きまとう．この誤差によって，非常に近い点があった場合などに自己干渉が発生してしまう可能性がある．浮動小数点演算による数値誤差を回避するために，可変倍長整数型を用いた整数演算による処理[2][4]が考えられるが，整数演算を用いると処理を進めていくうちにデータ量が増大していく問題[3]がある．このとき，データ量を減らすために精度を落とす処理が考えられるが，精度を落とす処理を行った際にやはり自己干渉が発生する可能性がある[4].

また，平面上に曲線として定義されたループを多角形で近似するという処理もよく使われる処理だが，ここでも自己干渉が発生する可能性がある．

そのほかには，2 次元の形状のモーフィング処理において，最初の形状から目的の形状に変化する過程で自己干渉が生じる場合がある[5][6]. 途中の形状を取り出して色を塗るなどの処理を行う場合にやはり不都合が生じるため自己干渉の除去が必要である．

自己干渉を除去する処理は，解こうとする問題によって，

・ 位相構造を変えない
・ 幾何情報を変えずに位相を変える

の 2 通りの解き方がある．前者はループを構成している頂点や稜線の並び方，接続関係としての位相構造を一切変えることなく幾何的な座標値のみを変えることによって自己干渉を除去する手法である．一方，後者は頂点や稜線の接続関係を切断したり繋ぎ変えたりすることで座標値を一切変えずに自己干渉を除去する手法である．本報告では，3 次元形状を平面に投影する問題，モーフィングの問題，曲線の多角形近似問題などに適用できる，前者の，位相構造を変えずに自己干渉を除去する場合についてのアルゴリズムを述べる．

本報告では，ループを輪ゴムのようなものとして考え，ねじれている輪ゴム，つまり自己干渉のある輪ゴム，をほどいていくような処理として自己干渉の除去を行う．輪ゴムは切ったり繋いだりができないので，位相構造の操作はできないものとし，ゴムの弾性によってもとの形状に近くなろうとするような力を考える．

入力される形状は，平面上に定義された稜線のループであり，自己干渉がないもの，自己干渉があるもの，自己干渉が複数あるものなど，どのような形状のものが入力されても良いものとし，出力される形状は，ループに自己干渉がないもので，入力された形状にできるだけ近い形状をしているものとする．

本報告での自己干渉除去のアルゴリズムは以下のような処理で構成される．

(1) ループの自己干渉の有無を判定する
(2) 自己干渉がなければ処理を終了
(3) 自己干渉の位置関係を自己干渉グラフで表現
(4) 自己干渉グラフに基づき力学モデルを用いた反転処理を行う
(5) (1)に戻り自己干渉がなくなるまで処理を繰り返す

2 章では，ループの自己干渉の有無をバウンディングボックスを用いて効率的に求める手法を述べ，3 章でループの自己干渉の位置関係をグラフを用いて記述し自己干渉を除去していく手法を示し，4 章で輪ゴムの力学モデルを用いた自己干渉の除去に用いる反転処理について述べる．また，5 章で自己干渉の除去例を示す．

2 自己干渉の有無の判定

ループの自己干渉があるかどうかを判定するには，ループ内の稜線同士が交差するかどうかを判定する処理になる．この処理は，ループ内の稜線の組み合わせの数だけ行われるので，$O(n^2)$の処理となり稜線数が多くなると自己干渉の有無を判定するだけで処理に時間がかかってしまう．自己干渉が発生するケースは，全体から見るとそれほど大きな確率ではないので，自己干渉の無いループに判定時間をかけるのは無駄である．

本報告ではループをバウンディングボックスを使って分割し，再帰的なバウンディングボックス同士の交差判定による自己干渉の有無を判定する手法について述べる．

2.1 バウンディングボックスの生成

本報告では，バウンディングボックスにはX-Y方向に直交するような矩形を用いる．バウンディングボックスは，図1のように連続する稜線をたどっていったときに，XとY方向についての増減の符号がそれぞれ等しい区間を1つのバウンディングボックスとして定義する．

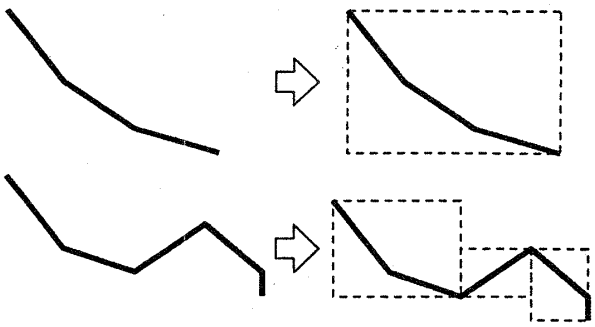


図1　増減の方向が一様な領域

このようにして図2のように，ループ1周分のバンディングボックスを作成する．

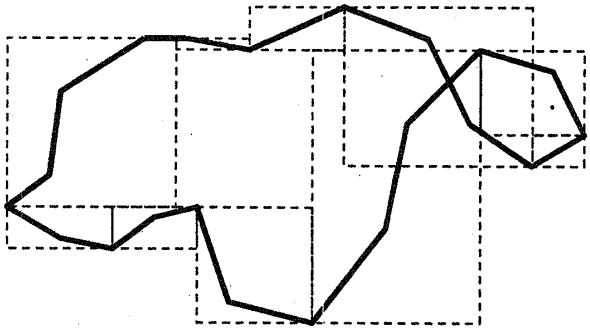


図2　ループのバウンディングボックス

2.2 バウンディングボックス同士の交差判定

続いてバウンディングボックス同士の交差判定を行う．バウンディングボックス同士が交差している部分は，稜線同士が交差している可能性がある部分であり，さらにバウンディングボックスを細かく分割して調べる必要がある．

バウンディングボックス同士の交差判定は単純な大小比較の処理のみでできるので高速である．

もしバウンディングボックス同士が交差していることがわかったら，それぞれのバウンディングボックスを2つに分割してできる4つのバウンディングボックス同士を使って同じように交差判定を行う．交差判定の結果交差する組み合わせが見つかったらさらに2つに分割し，分割できなくなるまでこの処理を再帰的に繰り返していく．交差する組み合わせが見つからなかった場合はこの部分には自己干渉が存在しないと判定できる．

バウンディングボックスを2つに分割する処理は，バウンディングボックスに含まれる稜線の数が半分になるように中央で分割する．なお，分割できない状態とは，バウンディングボックスにただ1つの稜線が含まれる場合である．

最終的に，稜線が1つだけ含まれるバウンディングボックス2つが残ったとき，2つの稜線同士の交差判定を行い，交差すれば自己干渉があると判定できる．自己干渉の有無のみを調べる場合はこの時点

で処理を終了するが，自己干渉の数を調べたり自己干渉点を求める場合にはすべてのバウンディングボックスについて処理を行う．

```
バウンディングボックスの交差判定(B0, B1){
  if(B0, B1 に稜線が 1 つのみ){
    if(2つの稜線が交差)
      自己干渉あり
  }
  else if(バウンディングボックス同士が交差){
    B2,B3←B0 を2つに分割
    B4,B5←B1 を2つに分割
    バウンディングボックスの交差判定(B2,B4)
    バウンディングボックスの交差判定(B2,B5)
    バウンディングボックスの交差判定(B3,B4)
    バウンディングボックスの交差判定(B3,B5)
    if(上の4つの処理で自己干渉あり)
      自己干渉あり
  }
}
```

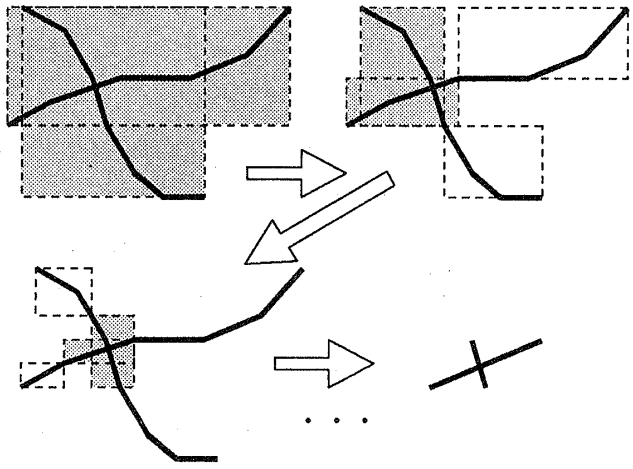


図 3　再帰的な交差判定

バウンディングボックスを使うことで自己干渉の有無の判定処理を，$O(n\log n)$程度で行うことができる．特に，自己干渉がまずおこらないような場合，つまり最初に作成するバウンディングボックス同士がまったく交差しない場合はバウンディングボックスの組み合わせ数で処理が終了する．

しかし，バウンディングボックスの数が稜線の数と一致するようなケースでは結局 $O(n^2)$となりバウンディングボックス作成の時間だけ判定に余計な時間がかかってしまう．(図 4)

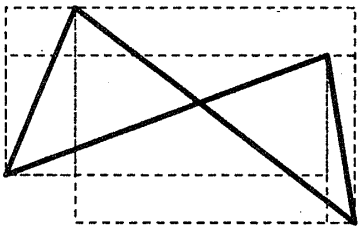


図 4　最悪のケース

3 グラフによる自己干渉の位置関係の記述

自己干渉の存在が確認されたら自己干渉を除去していくが，自己干渉が複数ある場合，どの部分から自己干渉を除去していけば最終的に自己干渉を完全に取り除くことができるかを知る必要がある．

本報告では，ループ内で自己干渉がどのような位置関係にあるかを記述する手法として，自己干渉点をグラフの辺とするようなグラフを考え，グラフに基づいて自己干渉の除去を行っていく．このグラフを自己干渉グラフと呼ぶ．

なお，本報告は位相的なつながりの順序を変えずに自己干渉を除去する手法について述べているが，3 章の自己干渉グラフは位相を変えて自己干渉を除去する場合においても自己干渉点の位置関係の記述に適用することができる．

3.1 自己干渉グラフの作成

自己干渉によってループはいくつかの領域に区切られる．この領域をグラフのノードと見た場合，隣の領域へ移動するには自己干渉点を通過することになる．そこで，自己干渉点を辺として領域であるノードを結んだグラフを用いて，ループ内の自己干渉点の位置関係を表現する．(図 5)

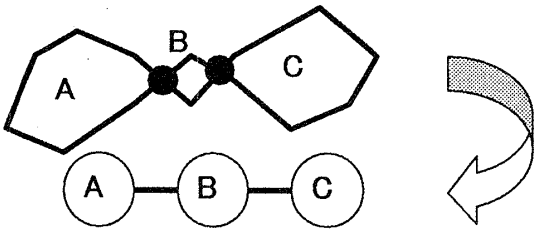


図5　自己干渉グラフ

自己干渉グラフは，稜線のループをたどりながら，出現する自己干渉の情報をもとに作っていく．自己干渉点の情報には，自己干渉の有無をチェックする処理で交差する2つの稜線の対を保持しておき，ループをたどったときに出現する順にソートしておく．

自己干渉グラフの生成処理は次のような処理である．

(1) 最初の自己干渉点を現在の自己干渉点とする
　最初のグラフのノードを作る
(2) 現在の自己干渉点で片方の稜線から対になる稜線までをたどる
　if(次の自己干渉点にたどり着いた){
　　この点を辺にした新しいノードを作る
　　(2)を再帰的に処理
　}

3.2 自己干渉グラフを用いた自己干渉の除去

自己干渉は，自己干渉によってできる領域を反転していくことでほどくことができるが，自己干渉グラフを用いることで，どの部分を反転すれば最終的にすべての自己干渉を除去できるかを知ることができる．

領域の反転で基本的なルールは次のようになる．

(1) ある領域を反転する場合，そこからグラフをたどったとき偶数個先の領域も反転する．
(2) 奇数個先の領域は反転しない．

たとえば，図5の場合にルールを適用する．領域Aを反転するとしたとき，領域Cも反転し，領域Bはそのままである．こうしてすべての自己干渉を除去できる．

このルールをそのまま適用しても多くの場合はうまくいくが，3つ以上の辺があるノードを反転するような場合はきれいな反転処理にならない場合がある．また，反転前と形状を大きく変えないためにも，なるべく狭い領域の部分を反転させるようにすると良い．

そこで，自己干渉グラフを，反転したくないノードを根としたツリー構造のような形になるように表記する．表記上，上に書くものほど反転したくないノードである．このとき下のほうに書かれたノードから順に反転の処理を行い，領域を統合していく．一番下には常に辺を1本しか持たないようなノードがくるため，反転の処理を行うのは常に1つの自己干渉点を持つような領域となる．

こうして最終的には1つの領域になり，自己干渉を完全に除去できる．

この処理の様子を図6に示す．

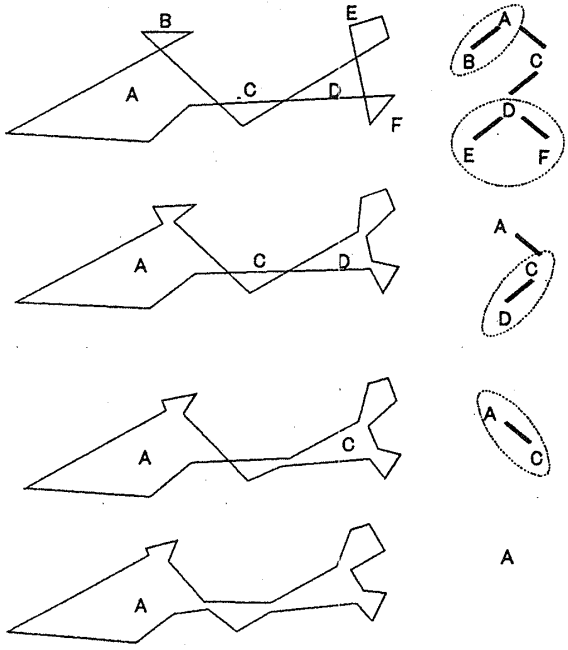


図6　グラフを用いた反転処理

しかし，ここまでの処理は，グラフに閉路がある場合は適用することができない．図7のようにちょうど輪ゴムに結び目ができたような状態のときグラ

フに閉路ができてしまう．

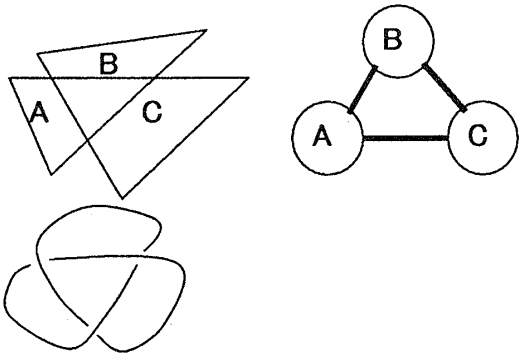


図７　結び目と閉路のあるグラフ

結び目は図７の紐の絵のような状態では紐を切らない限りほどくことはできない．もともとのループは紐のように交差する部分でどちらの稜線が上になるかが決まっているわけではないので，この交差部分の上下関係の解釈を図８のように変更する．

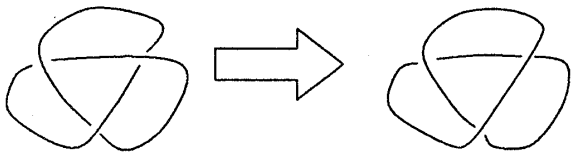


図８　結び目の回避

図８のように解釈すると，１箇所で自己干渉した輪ゴムを折りたたんだような図に見え，結び目はない．この状態をグラフで表現すると，図９の上図のようにA，B2つの領域と１つの自己干渉で表現できる．

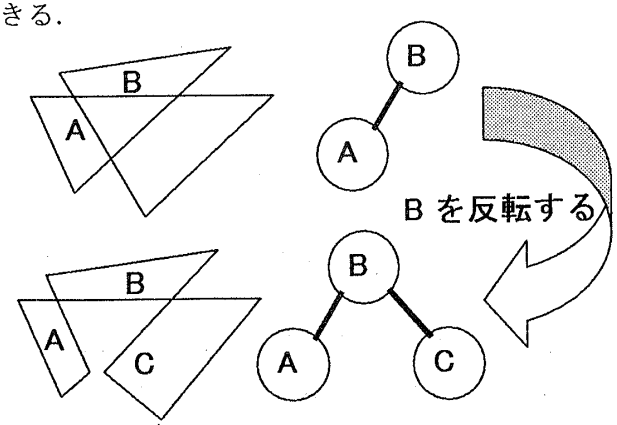


図９　結び目がある場合の前処理

このときの自己干渉部分を反転する処理を行うと，図９下図のように新しい自己干渉の状態ができるので，自己干渉グラフを作り直して図６に示した通常と同じ処理で自己干渉の除去を行うことができる．

4　力学モデルに基づく反転処理

ここでは，輪ゴムの弾力による力と，反転による移動先に向かおうとする力のつりあいを考え，頂点を移動する．

4.1 反転による移動先

反転によって自己干渉を除去するとき，図 10 のように頂点を自己干渉点をはさんでちょうど反対側にある頂点に移動させると幾何的な反転を行うことができる．ここでの移動とは，単に座標値を入れ替えるだけの処理であり，頂点のつながりの位相を変えることではない．

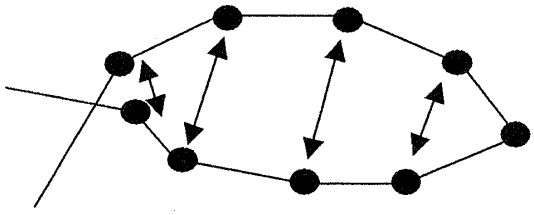


図 10　反転による移動先の決定

自己干渉点から両側から同時に稜線をたどっていき，頂点を順番にそれぞれ対応付けていく．頂点数が奇数の場合最後の点は移動しない点とする．

4.2 ゴムの弾性による力

頂点は両端の稜線によって引っ張られる力を受ける．ゴムの弾性係数をｋとしたときに，稜線の長さをｋ倍した力が頂点の両側にかかることになる．ちょうど，ラプラシアンオペレータと同じ処理になる．ラプラシアンオペレータとは，ある頂点をその両端の頂点の重心に移動させるような操作である．(図 11)

この力によって，形状がもとの形状からかけ離れたものにならないようになり，また稜線同士が滑らかに接続するようになる．

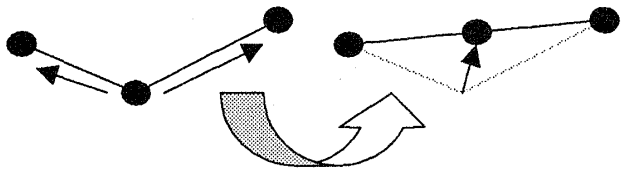


図 11　弾性による力

4.3 つり合いによる座標値の決定

移動先に向かおうとする力と，弾性により両端点から引かれる力の 3 つの力のつり合いを考え，力のつりあう位置に頂点を移動させる．(図 12)

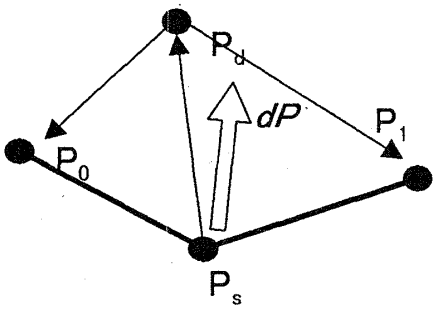


図 12　3 つの力のつりあいと移動先

このとき，移動する頂点を P_s，移動先として対応付けられた点を P_d，両端点をそれぞれ P_0，P_1，ゴムの弾性係数を k とするとき次式によって移動先へ向かうベクトル dP を求めることができる．

$$dP = \frac{P_d P_s + k\left(P_0 P_d + P_1 P_d\right)}{1+2k} \qquad (1)$$

つまり，弾性係数 k を重みにしたような重み付け平均によって移動先を決定する．著者らの実験では，k=0.02 程度にすると良好な結果が得られることがわかった．

5 結果

本報告の処理のうち，処理時間の多くは，自己干渉の存在判定が占めている．本報告で提案したバウンディングボックスによる自己干渉の存在判定の速度試験の結果を図 13 に示す．

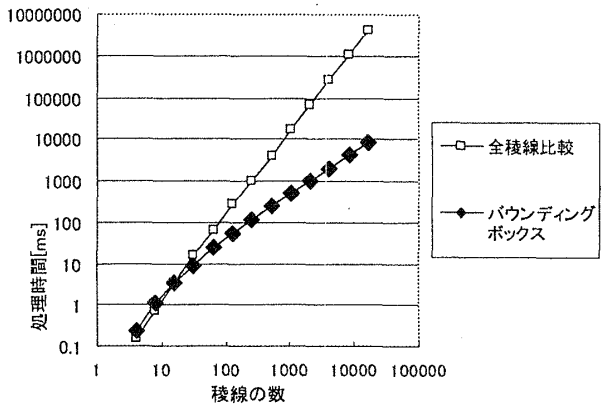


図 13　自己干渉の存在判定の速度試験

この結果，稜線数が少ないときはバウンディングボックスを作成する処理などのオーバーヘッドの分処理速度は劣るが，稜線数が増えたときには大きなアドバンテージがあることが確認された．

また，本報告で述べた自己干渉除去の手法を用いて自己干渉の除去を行った結果を図 14～17 に示す．

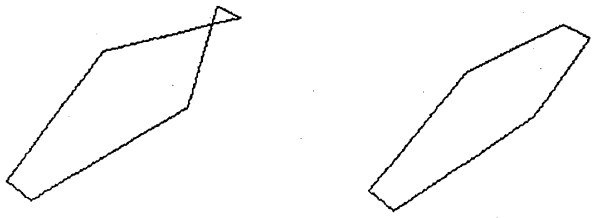
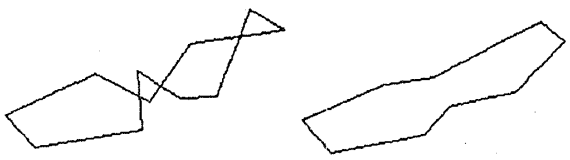


図 14　単純な自己干渉の除去例



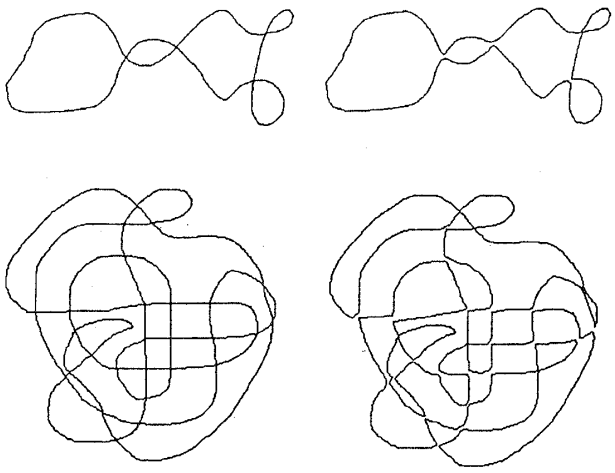


図 15　少し複雑な自己干渉の除去例

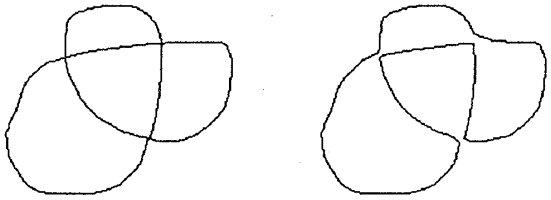


図 16　結び目の除去例

また，本手法では，除去前と除去後のループにおいて，奇偶法を用いた内外判定による結果が図 17 のように等しいことが確認できた．

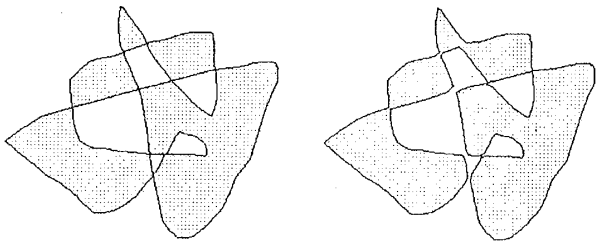


図 17　除去前と除去後の内外判定の結果

これらの例で，入力された形状になるべく近い自己干渉の除去された形状が得られることが確認できる．

6 まとめ

本報告では，ループの自己干渉の有無を効率よく判定する手法，グラフを用いて自己干渉点の位置関係を表現し，自己干渉を除去する手法，輪ゴムの力学モデルを用いて入力形状に近い形状を反転処理で得る手法について述べた．

本手法を使ってループの自己干渉の除去を行い，どのような自己干渉があるループが入力されても自己干渉が除去でき，入力形状に近い出力形状が得られることが確認された．

今後は，メッシュ生成処理における自己干渉除去の問題に，本手法を適用していく．

参考文献

[1] Atsushi Yamada, Kenji Shimada, Takayuki Itoh, "Energy-Minimizing Approach to Meshing Curved Wire-Frame Models," Proceeding of 5th International Mesh Roundtable '96, (1996), pp.179-191.

[2] 山口富士夫, "完全 4 次元同次処理に基づく CAD（第 2 報）", 精密工学会誌, 第 65 巻, 第 1 号, (1999), pp.78-84.

[3] 土井淳,吉田典正,津金尚志,山内俊哉,金俊赫, 山口富士夫, "同次幾何演算の整数値データ長の増加問題に対する考察", 1997 年度精密工学会秋期大会学術講演会講演論文集, (1997), pp.479.

[4] Steven Fortune, "Polyhedral modelling with exact arithmetic," Proceedings of the third symposium on Solid modeling and applications, (1995), pp225-233.

[5] Thomas W Sederberg, Eugene Greenwood, "A Physically Based Approach to 2-D Shape Blending," Proceedings of the 19th annual conference on Computer graphics, (1992), pp.25 – 34.

[6] Tatiana Samoilov, Gershon Elber, " Self-intersection elimination in metamorphosis of two-dimensional curves," The Visual Computer, vol.14, (1998), pp415-428.