二次元　スペース・モデリングの考察
Study on 2D Space Modeling

大沢　晃　　(株)日立製作所武蔵工場
Akira OHSAWA　　(Musashi Works HITACHI Ltd).

A new concept of 2D geometrical data structure is introduced. Since adjacent and/or piled subspace can be retrieved locally by pointer, very high-speed local processing capability is expected, independent of total vertices number N,i.e. $o(N^0)$, and also expected required construction time of data base tend to o(N), if special case can be ignored.
By means of this concept, local geometric Boolean operation(AND, OR), collision detection of moving figure, high-speed windowing, and many other useful applications will be possible effectively.

1，はじめに

計算機は人間に比べて図形処理が苦手だと言われている。また，従来の手法では計算機時間が図形数と共に急増するため，原理的には可能でも実用上不可能な問題が沢山あった。その一つの原因は，CAD 等における図形処理が本来図形相互間の重なりや、隣接図形間の関係を処理するものであるにも拘らず，図形データベースにこれらの情報が記述されていないため，必要図形のみの効率的検索ができず，関係のない遠方の図形データまで読出して来て無駄な処理をしていた所にあると考えられる。

本報告はこの問題を解決するために，図形空間の構造をモデル化して蓄積する図形データ・ベースのファイル構造とその応用に関するものであり，図形処理を注目部分に限定してローカルに実行する手法によって，特にオン・ライン処理のスピードを抜本的に改善することを目的にするものである。本方式によれば，重なり図形や隣接図形が高速に求まるので，ＡＮＤ，ＯＲ等図形論理操作，運動図形の衝突検出，ライトペン等による図形ピックアップ，地図上での道順検索等，大規模図形に対して従来に比べて画期的な高速化が期待される。

2，ポインタによる図形空間モデル化

2，1　部分空間のポインタ表現

図１に示すように，頂点の周りの凸な（辺に挟まれた角度が180°より大な）空間を，Ｙ軸に平行な仮想的境界線Ｂx（図中一点鎖線）で区切り，これと図形辺Ｂyによって図形空間を部分空間ω1，‥‥ωi，‥‥に分割する。部分空間ωiはそのx^+，x^-端に必らず図形頂点を持つので，この頂点に夫々空間に対応するポインタSP(Space Pointer)を設置し，ωiの両端のSPが相互に相手のSPのアドレスを指し合うようにする。この相互に指し合うSPの対をSPP(Space Pointer Pair),図形データ・ファイルのことをSPF (Space Pointer File) と呼ぶ。

SPFの空間分割は図形データと座標系が決まれば一義的に定まるので，任意の部分空間ωiについて隣接部分空間は一義的に定まる。また，各ωiには必らず一本づ々のSPPiが一対一で対応している。従がってωiはSPPiで表現されていると云ってよい。

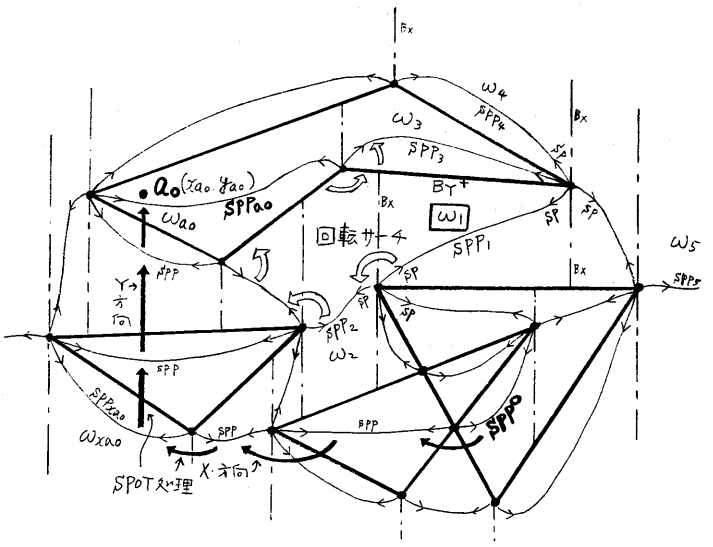


図１．SPFにおける部分空間ωiのSPPiによる表現

ωiがSPPiで表現されているから，図形処理における図形（空間）検索問題はSPFの中から必要なSPPiを検索する問題に置き換えることができる。
なお，ここでは、図形内部も外部も空間としてSPPで平等に表現される。

2．2　ωiの外形と隣接空間の検索、

SPPi の検索によってωiの検索に代えるためには、SPPi が与えられたら，そこからωiに関する全ての情報を少ない手数で調べることができる必要がある。図１から，ω1のX方向外形線Bxと，隣接空間ω2に対応するSPP2 が，ポインタ検索で直ちに求められることは明らかであろう。

Y⁺側外形線By^+を求めるには，白矢印で示すように，頂点で接続しているSPPと辺を一方向へ回転するように辿ってゆけば，Byまたはその延長上の頂点に到達する。これを回転サーチ(Rotary Search)と名付ける。回転サーチで必らず外形線By^+が求まる理由は，SPPの左回転左辿りはBy^+に下側から接する空間の左辿りと等価であり，且つこのBy^+またはその延長線上，少なくとも終端には必らず辿ったSPPの接続する頂点が存在するからである。By^+が求まったら，Y側隣接空間ω3に対応するSPP3は，By^+からもう一度上側への回転サーチで求められる。以上の手法は図形の形状によらず可能である。また，同ω4，ω5等有界でない空間をきちんと取扱うためには図面全体を囲う充分大きい外枠(図面外形線)を入れると考え易い。

本手法の特徴は，上記各種処理がSPF構造が事前に作成してあれば（第４章で説明），部分処理で，従がって極端な特殊ケース（付録）を除けば，全図形頂点数Nに無関係でo(N⁰)の高速処理が期待できることである。

2．3　与えられた点の位置付け，SPOT処理

図１において点aoが座標値(Xao,Yao)で与えられたとき，aoを含む空間ωaoを求める。これは，マウスやタブレット／ペンで図形をピックアップする時等のほか，後述のSPF生成にも重要な役目をする。

さて，SPF上でこれを実現するには，ωaoに対応するSPPaoをファイルの中から検索する問題を解けばよい。そのためには前節で述べた任意方向隣接空間の高速検索機能を利用する。すなわち，図１の既設SPFの中から，最初に任意の一つのSPP⁰をとり，SPP⁰ の両端の頂点のX座標値をXa。と比べもしその範囲がXaoを含んでいなければ，そこからXaoを含む空間に達するまで，且つなるべくYaoにも近い方向へ，X方向の隣接空間のSPPを辿って接近する。そのうちにX座標値がXaoを含む空間ωxaoに達したら，今度は回転サーチによって，Y方向で且つXaoを含む隣接空間だけを順に辿ってゆけば必ずSPPaoに到達する。この手法は探索ルートが最初はX方向に，次はY方向に一方向的であるため，ループや振動はあり得ず，また全空間調査もしないので，特殊ケースを除けば高速と言える。

SPPao探索時間の平均値は，図形分布が均等で，且つ特殊ケースが無視できれば，任意に選んだSPP⁰からSPPaoまでの平均距離に比例するから，オーダーとしてはo($N^{\frac{1}{2}}$)となる（全頂点数Nの図面の中の直線距離）

更に高速化するには，図２の如く図面全体をmヶ(∝N)に区分し，各区間に道しるべ点g_i(i=1,2‥m)をあらかじめ設置し，aoの座標値が与えられたらその区間の

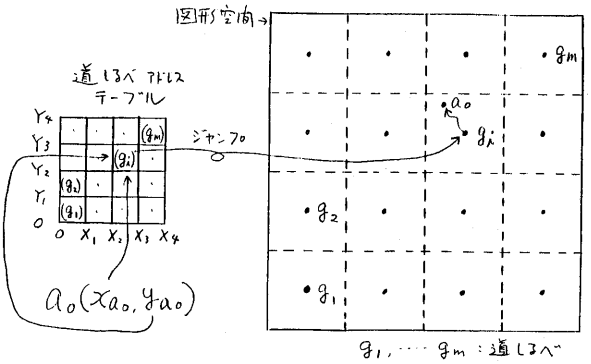


図２．道しるべ導入によるa。(SPPa。)探索の高速化

g_iへ直接ジャンプし，そこから上記手法をとる。区間数mをNに比例してとり，一区間内の頂点数がほぼ一定にでき，且つ付録の特殊ケースの対策をすれば，平均探索時間はo(N⁰)に押えられる。
一種のバケット・ソート[2]である。例えば，一区間の頂点数が100程度になるように区分すれば，g_iからの平均探索距離は大体10以下となり且つg_i設置によるメモリ増加量は１％程度で実用範囲と思われるが，図形の性質によるのでその都度検討を要する。また道しるべとして専用の点でなく，区間内の既設

頂点を登録する方法や，近傍頂点が続けて出現する頻度が高いことを利用して，最近処理した頂点がg_iよりa_oに近ければこれをSPP°に使う等高速化手法も考えられる。

２．４　空間属性保持による重なり検索の超高速化，

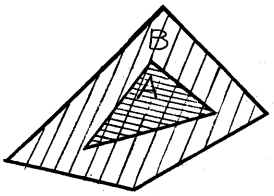
すべての図形の内部が各図形毎に独得の色で塗られており，図形の重なりを色の重なりとして見ることが出来るとすれば，図３(ｉ)において与えられた図形Ａと他の図形の重なりを調べるには，Ａの内部の色の重なりを見るだけで充分であり，図形の形状や数に無関係な高速判定ができる筈である。これを計算機で実現するためにSPFを利用する。

SPFではポインタの組SPPによって部分空間を表現しているので，図形内の部分空間に色を塗る代りに図形内部を表現するSPPに色に相当する空間属性（図形名等）を付加しておけば，図形Ａ内部のSPP を調べるだけでＡがどの図形と重なっているかを直ちに知る事が出来る（図３，(ii)，(iii)）.従来の手法のように辺の交叉を一つづつ調べる必要が全くないので，SPFさえ出来ていれば図形が複雑大規模なときにも重なりが直ちに判明し，処理時間はｏ(N°)である。

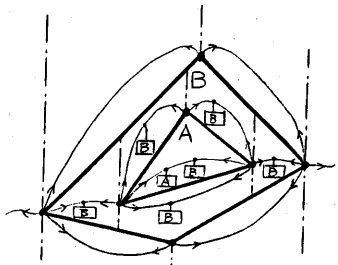
SPFの作成は多少複雑なので別途第４章で説明するが，SPFに図形を追加する場合追加図形内のSPPに，追加図形の空間属性と追加前のバック・グランドのSPPにあった空間属性を加えて付加するだけで重なり表現のデータを作成できる。

２．５　各種図形のSPF表現

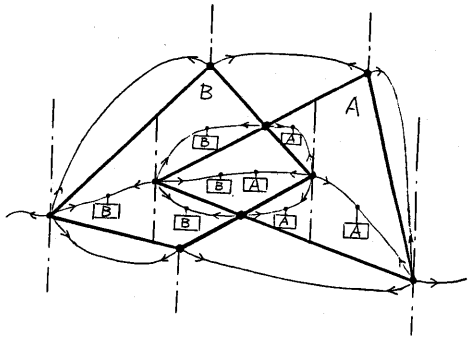
SPFの構造は，図１に示したポリゴン等のほかに図４～図９の各種図形にも適用可能である。
なお図６の曲線については詳論を要するが今回は省略させて頂く，また，SPFにおいて図形が全て点図形ばかりである場合には，各点はＸ座標値の順にポインタで接続され，丁度X値でソートされた形となる。その意味で，SPFはソーティングの二次元有限寸法図形への拡張とも考えられ興味深い。



(ｉ)色による重なり判定



(ii) A , B による重なり判定



(iii)交叉図形の場合

図３．空間属性 A , B による重なりの表現

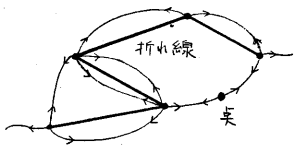


図４．折れ線，点

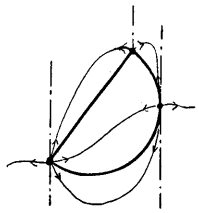


図６．曲線

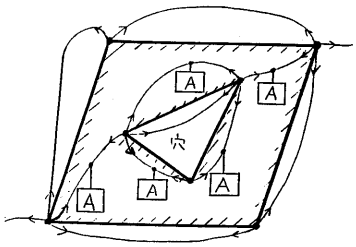


図５．穴のある図形

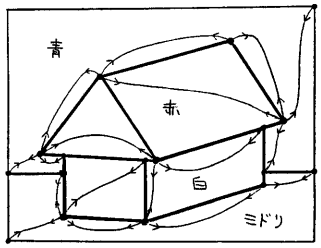


図７．多色図形

２．６　頂点テーブルによるSPFの表現

図10にSPFを具体化するための，頂点テーブルによる図形データ表現方式の原理を示す。ここでは辺もポインタEP（Edge Pointer ）で表わし，頂点から発するEPとSPを左廻りループ状の順に並べた頂点テーブルを作成することにより，頂点周りのEP とSPの隣接性を表現し，回転サーチの便宜を図っている。

図中＊印は頂点テーブルの長さを一定に保つために設置した空欄（内容はＮＵＬＬ）であり，メモリサイズは少し大きくなるがエリア管理は容易になる。

例えば，図形が図１０（iii）のように回転した場合にも頂点テーブルのサイズは不変で，単にSPの指し先を交換する（内容を書き換える。）だけでよい。

またSPに付加して図形の色，名称，重なり等を表現する空間属性は，辺EPの属性(太細，破線etc.)と共にポインタの属性としてATR欄に記述してあるが表現が多少冗長な面もあるので，具体化に当っては工夫の余地もある。

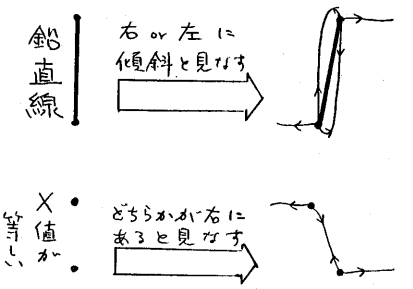


図８．Ｘ座標値の等しい図形

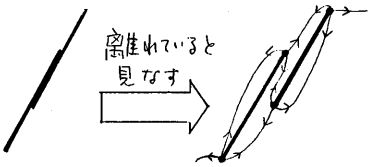


図９．重なり線

３．ＳＰＦの応用

３．１　図形論理操作

（１）図形のＡＮＤ

図形の重なりは与えられた図形内部のSPPの属性を調べれば判定できるので，重なり部の図形だけをとり出せばよい。与えられた図形一個の処理時間は，その図形内部の重なり空間数が全体図形頂点数Ｎに無関係なら o（N°）。二つの図面全体を重ねて処理する時には，後述（４．４）の手法により o（N）が期待される。

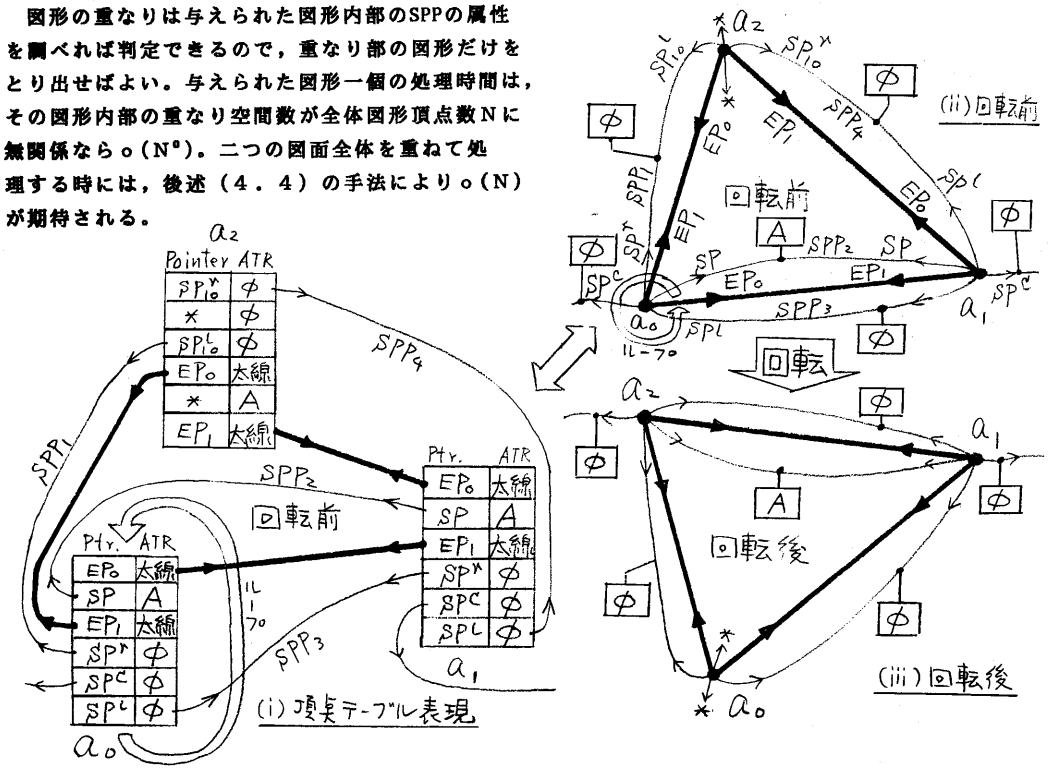


図１０．図形の頂点テーブルと回転時のSPPのメンテナンス

（2）図形のOR

図形データから重なり部分の辺を消去し，SPPに付加された重なり図形名を付け直せばよい，前記と同様の条件で，一個当り o（N°），全体で o（N）

（3）NOT

原理的には実部分を空に，空部分を実に，SPPに付加した図形名を付け直せばよい。しかし，この付け換えは図形内外全空間におよぶので，全体図形規模Nが増大すると時間がかかり過ぎ，実用上問題がある。幸い，実際には次のEORまたはCUT（NOTAND）操作の方が実用的であり，NOTは殆ど使わなくて済む。NOT対象図形に拘らず o（N）である。

（4）EOR

図形重なり部SPPに付加する図形名を空にする。o（N°），o（N）

（5）CUT（NOTAND；切削）

図形ファイルから，与えられた図形との重なり部を削除する。o（N°），o（N）。

3．2　ウインドーイング，ベクタ／ラスタ変換

大規模図形データの一部を表示用ウインドー等の形状に切り出す。最初に切出すべき形状の外形線をSPFに入力し，次にその内部にある全ての図形をSPPを辿りながら抜き出し，最後に当初入力した外形線を削除する。この場合もウインドーの外側のデータを調べることは原理的に不要なので，全頂点数Nと直接には無関係で従来のいわゆるクリッピングより高速，且つ切り抜き形状が任意という特徴もある。

ウインドーの変った応用として，静電プロッタ等に使うベクタ→ラスタ変換が考えられる。

ウインドーの外形をピクセルの一ケ分（ラスタの1ドット）として，その中に図形があれば“1”，無ければ“0”として順にスキャンすれば，ベクタ→ラスタ変換が出来る。スキャンの方向，形状等をコントロールすれば，大きさ，部分，回転，精度等は任意で，静電プロッタの出力順にオンライン処理することも可能と思われる。

3．3　注目，近傍の探索，図形認識への対応。

図形空間の中の一点が与えられたとき，SPFを使ってその点を含む部分空間や，これに隣接する部分空間を部分処理で調べる，人間の眼の注目動作と類以の処理が考えられる。この方法はライト・ペン，タブレット等で指定された図形の高速ピック・アップに有効である他，地図の検索等にも利用可能と思われる。部分処理が可能なので，ＬＳＩのレイアウト等では図形の対話形入力，修正に伴なうオンライン・デザインルール・チェックが高速で実行できるだろう。図形認識への対応について，多少の異論もあろうが，以下本方式の持つ好都合な性質を列挙してみる。

（1）部分への注目能力

（2）図形相対位置関係の把握能力

上下，左右，隣接，重なり，包含の認識，近傍図形をグループ視する能力等が簡単に賦与できる。

（3）位置，寸法等に関するデータ構造の不変性

但し回転には弱いので何らかの対策を要する。

以上は従来計算機では困難とされていた事であり，人間の持つ高度な図形処理能力に一歩近ずくものと考えられないだろうか？。例えば図11の中から△と△△を探し出すような問題は，図形数が多くなると従来手法ではかなり難問であるが，□の中に△があるか，△の横に△があるか等をSPF上で調べれば，全体を調べても o（N）程度で高速に実行できると思われる。

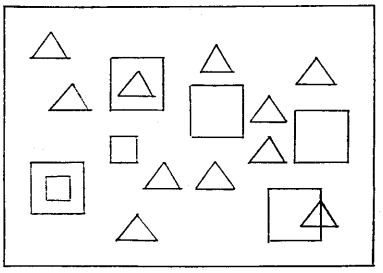


図１１．△と△△を捜す問題

3．4　運動図形の衝突検出

静止図形間の干渉チェックは従来から多く行なわれていたが，運動図形の衝突検出は簡単ではなかった。干渉チェックがスタティックであるのに，衝突検出は時間を含むダイナミックな処理であり，次元が異なる処理である事に注意したい。

大規模図形空間の衝突検出における大きな問題は衝突する相手の図形がどれかを探し出すことであるが，SPFを使うとこれが大変うまく出来る。図12において衝突点は衝突直前には必らず衝突する相手と同一部分空間の向うとこちらで対峙した形になっているのに違いないから，衝突のチェックは常に運動図形に接する部分空間だけについて行なえばよく極めて高速にできる。衝突の可能性のある相手の図形はSPFのポインタSPPを使えば直ちに求めることが出来る。図形Ａを動かしてゆくと，Ａと他の図形との相対位置が変化するのでＡの周囲の部分空間が変形し

てゆくが，ある所まで来ると，部分空間のトポロジカルな位置関係が変化するので，その時点でSPF上のSPPのつなぎ換え（メンテナンス）を行なう。更に運動を進めてゆくと，そのうちAの前面の部分空間のどれかで衝突が起こる。運動図形相互間の衝突も同様に取扱える。但し運動図形が多いとメンテナンスが大変にはなる。衝突発生または部分空間のトポロジー変化の時点は，運動や図形々状が解析的なものならば，方程式の解として求めることが出来る。複雑な形状の場合には，衝突可能性のある（チェックすべき）図形周辺の部分空間の数nが増加するが，計算機時間の増加はo(n)の程度であり，全体頂点数Nとは特殊ケース（付録）を除き関係ない。
衝突の検出は，LSI設計におけるデザイン・ルールを守ったレイアウト図形のコンパクション，板取り問題におけるつめ合せ，ロボットの衝突回避問題等広範な用途がある。

4．SPFの生成とメンテンス

4．1　SPF生成の準備：点aoの入力

SPFの初期生成は，メンテナンスにおける図形1個の追加を図形数だけ繰り返すことで行なう。
図形1個の追加は，最初に追加図形上の一つの頂点aoを点として入力し，次にその点から逐次辺を伸ばしてSPFのポインタをメンテナンスしながら図形辺全体を一巡する手法をとる。

最初の点aoを入力するときに，ao(xao，yao)が既設SPF（図1）のどの部分空間の中に入るかを調べるには，2．3で述べた点の位置付け処理SPOTルーチンを使えればよい。SPPaoが検索できたらこれを切断してaoをつなぎ込めば図形入力の前段階の処理は完了する。なお，点aoの入力は前述の道しるべを使えば一応o(N°)になるが，大量の図形を処理するSPF初期生成時には処理時間を少しでも減らすため，区間分割数mの値の最適化や，最近入力した点が道しるべgiよりもaoに近ければそれをSPP°に使う等，充分な配慮をすべきである。

4．2　SPFへの図形展開 DEVELOPルーチン

図形Aの入力は，前述のSPOTルーチンにより一つの頂点aoを入力した後図13(i)~(vi)に示すようにaoから出発してAの辺に沿った折れ線$\overrightarrow{a_0t}$を順に伸ばしながら，それに伴って起る空間分割の変化（イベントEiと呼ぶ）に対処してSPPを順次メンテナンスし，Aの辺を一巡し終ったら，Aの内部の空間のSPPにAの図形名や色など空間属性を付加して完了する。以上の手順をAの展開（DEVELOP）処理と呼ぶ。$\overrightarrow{aot}$の伸張に伴うイベントは基本的には$\overrightarrow{aot}$が伸びていって，その先端tが周囲の空間の境界BxまたはByを

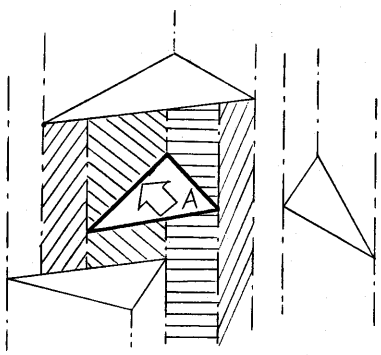
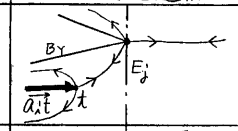
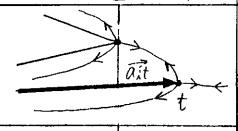
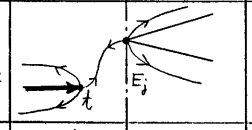
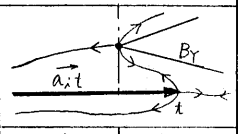
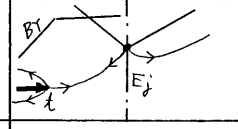
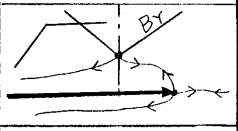
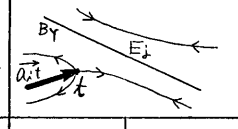
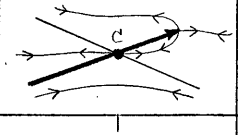
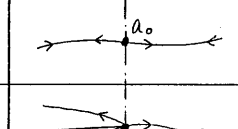
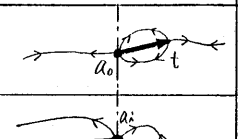
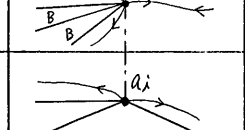
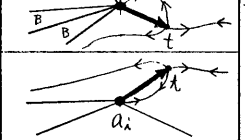
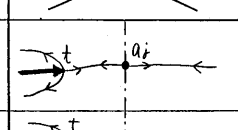
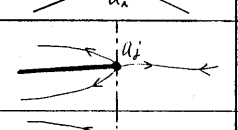
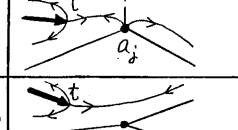
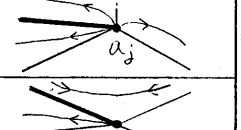
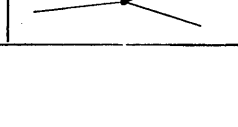
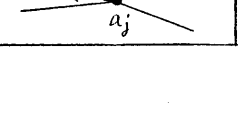


図１２．運動図形の衝突検出
斜線の空間のみチェックすればよい

突き抜ける時点のみに発生するので，SPPのメンテナンスは間欠的でよい。
起り得るイベントの種類は，上述の$\overrightarrow{aot}$のBx，Byの貫通に，$\overrightarrow{aot}$の始点と終端の処理を加えて表1のようになる。
イベント処理プログラムは次に発生するイベントの種類を判定してポインタの接続変更をすればよい。イベントの種類を減らして取扱いを簡単化するために，図13に示すように最初に追加図形の辺を全てX^-からX^+方向への折れ線に分解して，$\overrightarrow{ait}$の伸張を全て一方向に限定する方法をとってもよい。

展開処理は前述の回転サーチ等と同じく，入力図形辺の付近の空間（$\overrightarrow{ait}$の伸張処理）および，入力図形内部（空間属性付加）に限定されたローカル処理であるから，先のSPOTの高速化と合わせて，SPFのメンテナンス時間はo(N°)程度，SPF全体の初期生成時間もo(N)程度の高性能が見込まれる。
但し，特殊ケースが無視でき，且つ入力図形内部の空間属性を付加すべき図形空間数が全頂点数Nと無関係なことが必要である。
以上の結果からSPFは，上記条件付ではあるが少なくともオーダーの上では，オンライン用にも，バッチ処理用にも適した高性能図形データ・ベースとなることが期待される。なお，DEVELOPの逆にDELETEルーチンがあるが全くの逆操作なので説明を省略する。

表１．$\overrightarrow{a_i t}$の伸張に伴うイベントの種類

区分	No.	形状例 Event E_j通過前	形状例 E_j通過後	対象形等の存在
B_X貫通	1			±X対象 ±Y〃
	2			〃
	3			〃
B_Y貫通	4			〃
辺$\overrightarrow{a_i t}$の始点	5			±X対象
	6			左記と合み形状計7種あまり ±X対象 ±Y〃
	7			
辺$\overrightarrow{a_i t}$の終端	8			±X対象
	9			左記と合み形状計7種あまり ±X対象 ±Y
	10			

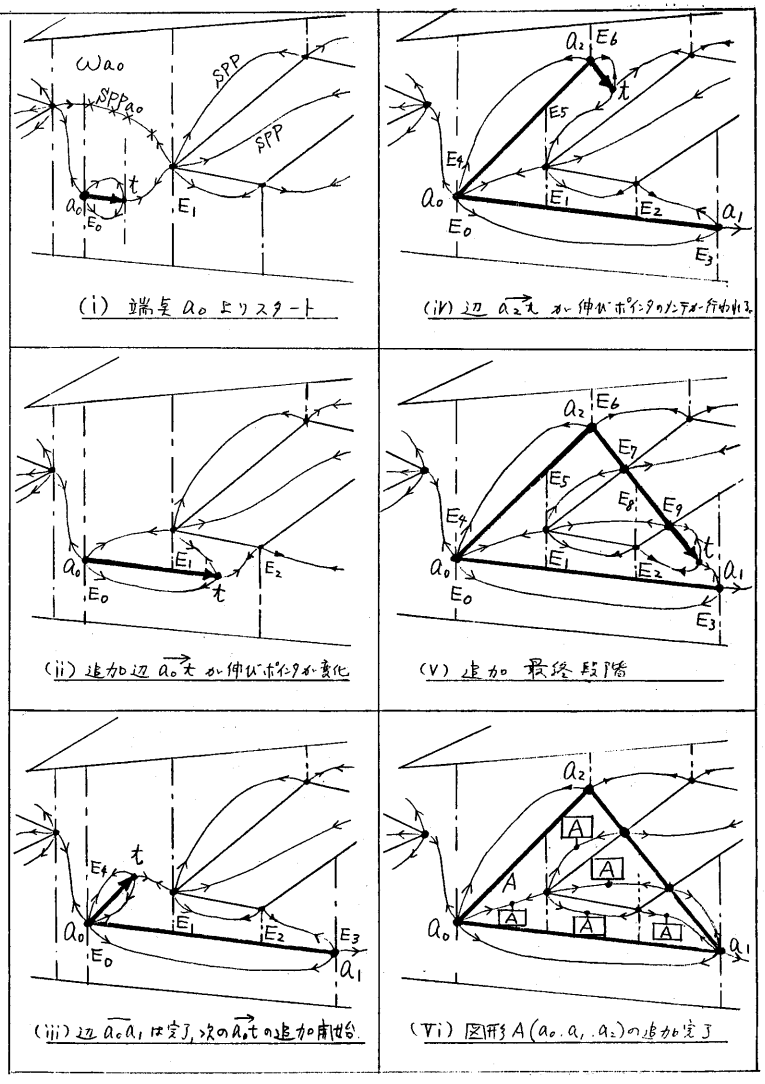


図１３．(i)~(Vi)図形Aの入力（「DEVELOP」）の手順

４．３　複数層のSPF重ね合わせ合成の合理化

ＬＳＩレイアウト図のチェック等，図14に示す複数層間パッチ図形処理で必要な、対象層の合成SPFの作成は，一つの層のSPFに他の層の図形を一ヶづつ入力する方式で行なうが，合成前の各層が層毎にSPF化されていれば，合成処理の改善が可能である。

すなわちSPF化された入力データでは図15の如くＢ層を片端から入力することによれば，各図形毎の最初の一点のSPOT処理を，Ｂ層SPFでSPPで接続されていた近傍の既入力点から開始できるので，道しるべ方式を使わなくても（特殊ケースを除けば）一点当り o（N°），全体で o（N）の高性能が期待できる。

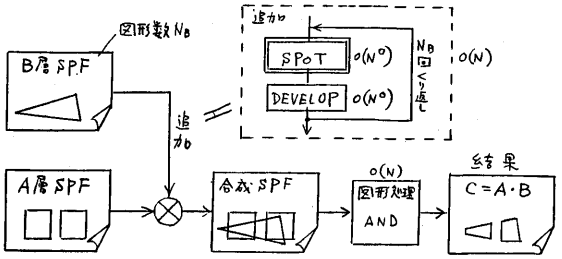


図１４．Ａ，Ｂ層間パッチ図形処理

４．４　大規模図形データのディスク・アクセス

ＣＰＵの主メモリに入り切らない大規模図形データでは，図16の如く全体を複数のページに分割してディスクに格納し，必要ページのみを主メモリに呼出す方式をとる。このときページ間を渡るSPPによってディスク・アクセスが多発するのを避ける為には，同図に示すように各ページの外形線を，エンド・ユーザーには見えない線図形として設置すれば（そこで空間が切断されるので）SPPは全てページ内で終端してしまうから，ページ内処理のときにSPPを検索しても処理が主メモリ外の他ページへ飛ぶことが無くなり処理効率が向上する。

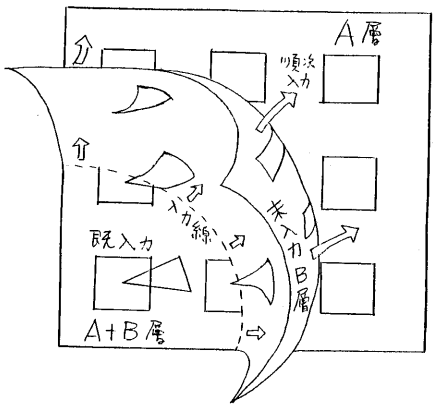


図１５．Ｂ層を片端から入力

５．むすび

新らしい図形空間モデル化方式を提案し，ファイル作成の基本問題と応用の可能性について考察した最大の特徴は本質的に図形規模によらない部分図形処理である。一旦ファイルが出来てしまえば，従来困難とされていた図形部分論理操作から衝突チェック等，広範囲の図形処理やファイルのメンイナンスが o（N°）で可能，ファイルの初期生成は一寸大変だが，オーダーとしては同じく o（N）で可能の見込である。処理時間の長くなる特殊ケースについては，ＬＳＩのレイアウト図等人為的な図形では発生し易い可能性があり注意が必要であるが，実際にはあまり極端なケースは少なく（文献６），付録で示したような対策をとらなくても実用化できる可能性がある。

また本方式ではポインタを多用するため，メモリ容量が従来の頂点座標値のみの図形データの数倍になる欠点があるが，これも o（N）であるし，最近のハードウェアの進歩を考えれば許容範囲としてよかろう。　本方式のシステムは未だ作成してないが，大きな可能性が有りそうなので今後実現に努力する所存である。

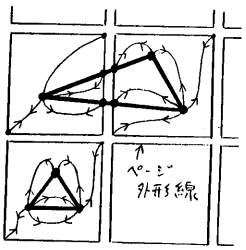


図１６．大規模データのページ分割

終りに，名古屋大学工学部杉原助教授，日立製作所半導体事業部金原事業部長，武蔵工場牧本副工場長，福田部長ほか，本研究に御支援，御助言下さった方々に感謝の意を表します。

参 考 文 献

1)鈴木則久；VLSI CADへの計算幾何学の応用，情報処理,VOL.24,NO.4,PP563～566(Apr.1983)

2)浅野,今井,伊理；計算幾何学へのいざない。bit,VOL.16,NO.12, PP52～61(1984)

3)D.T.Lee；Computational Geometry-A Survey,IEEE TRANS. on Computers, Vol.C-33, NO.12

4)John K.Ousterhout et al.; Magic:A VLSI Layout System 21st Design Autmation Conference,Proc.'84,IEEE ACM,PP152~159(1984)

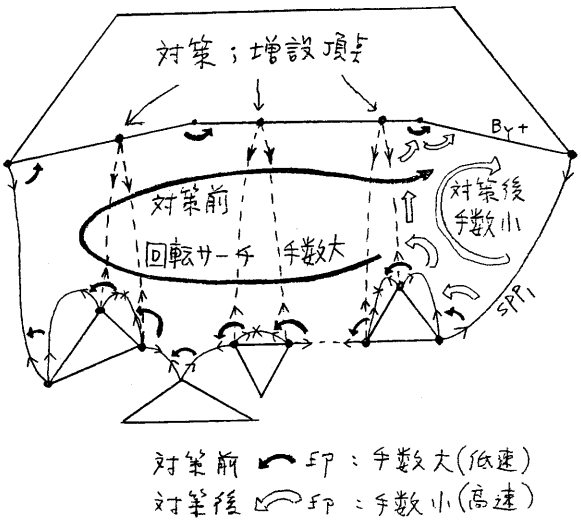
5)A.Tsukizoe,J.sakemi,T.kozawa,H.Fukuda；MACH: A High-Hitting pattern Checker for VLSI Mask Data ACM IEEE 20th Design Automation Conf.Proc.Jun. 1983 PP726~731

6)John K.Ousterhout ; Corner Stitching: A Data-Structuring Technique for VLSI Layout Tools,IEEE Trans.on CAD,Vol.CAD-3,No.1,Jan.1984 PP87～100

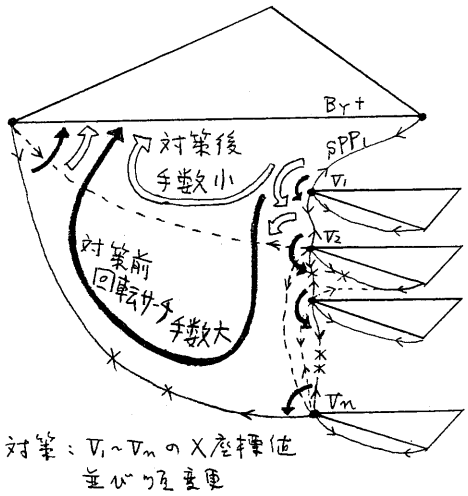
付録．特殊ケースの考察

本方式の性能は対象図形の形状，分布等に影響され，図Ａ１，図Ａ２等の特殊ケースでは回転サーチの手数が増加して性能が低下する。対策としてはデータ・ベース生成時に回転サーチの手数を一定値以下に押えるような間隔で，図Ａ２に示すような図形々状に影響を与えない頂点を増設する方法がある(その代り副作用として頂点が増加する)．なお図Ａ２では頂点のＸ座標値が等しいので，上記対策の極限としてＸ値の等しい点に微小間隔だけ離れた複数個の増設頂点を設置してもよいが,本文図８で示したＸ値の等しい頂点はどちらかが右にあると見なす方法を適用して，各頂点のＸ方向並び順を変える事により図示のように改善する方法もある。
(但しプログラムが複雑化する等の副作用が出る)

以上特殊ケースと対策の例を述べたが,対策はいずれも副作用があり,トレード・オフが必要となる。しかし,幸いLSIマスク・パタン等実用的な対象では図形サイズ,分布等は比較的均一な場合が多く(文献６),また大規模図形では本文4.4図16の示すページ分割を行なう事も一種の対策になるので,時々一寸応答が遅くなる事を我慢すれば,あまり無理な対策等はしなくても充分実用的になると思われる。



図Ａ１．By検索時間が長くなる特殊ケース（１）
（多数の図形と長いBy）



図Ａ２．By検索時間が長くなる特殊ケース（２）
（Ｘ座標値の等しい頂点V_1～Vn）