本 選 課 題 解 説

スーパーコン０８審査委員会
東京工業大学・大阪大学

はじめに

今回のスーパーコンでは，スパコンの速さを実感し，それを上手に利用してもらうことを狙って，あえて超難問を出すことにしました！ 以下に説明する 単純多角形面積最小化問題 です．

予選問題にも関連した問題です．ちょっと見るとプログラムの技術を問うような問題ですが，実は，なかなかうまい方法がない問題です．生半可なプログラミング技術よりも，腰をすえてスパコンをブン回して効率よく探索しないと解けない問題です．

そのため，今回はあえて前もって課題問題を本選出場の皆さんに公開しました．また，以下に述べるように，二通りの攻略方法と参考アルゴリズムも説明します．それだけ，今回の課題問題が困難である，ということでもありますが，スパコンのパワーをうまく利用する技術に専念して欲しい，という気持ちが込められているのだと思ってください．以下の解説や参考プログラムをヒントに（ただし，それらにとらわれることなく）素晴らしいプログラムを開発してください．

１．課題問題の定義

課題問題は以下の通りです．

単純多角形面積最小化問題

与えられた n 個の格子点 $p_1, ..., p_n$ に対し，それらすべてがちょうど端点となるような単純多角形で，面積ができる限り小さなものを求めよ．

多くの用語は予選問題と同様なので大丈夫だと思いますが，確認をしておきましょう．また，後に示すプログラム中のコメント等のために，対応する英語を示しておきます．

まず，格子点 (grid point) ですが，これは平面上の整数座標を持つ点のことです．予選問題と同じ形式で，この格子点が入力として与えられます．たとえば，図 1（左）の 0 から 9 までの番号のついた点です．この格子点の集合を以下では V と呼ぶことにします．

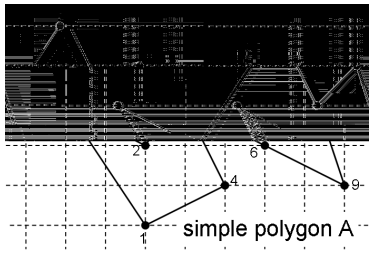
このような格子点の集合 V に対し，それらすべてを（しかもそれだけを）端点 (corner) に使うような単純多角形 (simple polygon) の１つを $P(V)$ とします．そうした単純多角形は１つではありません．そのすべてを集めた集合を $\mathcal{P}(V)$ とします．

たとえば V に対して，図 1（中央）の単純多角形も $P(V)$ ですし，図 1（右）の単純多角形も $P(V)$ です．これらすべてを集めた単純多角形の集合が $\mathcal{P}(V)$ です．

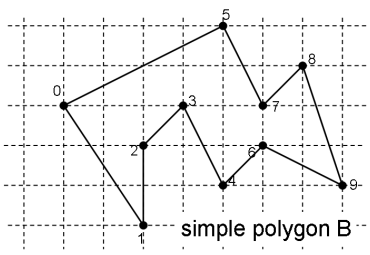
ここで 端点 (corner) とは，外辺の曲がり角になる点のことです．正確には，曲がり角と言っても，図 1 中央の 6 番の頂点のように，180 度の曲がり角（つまり直線上の点）になる場合もあります．こういう場合を含め，与えられた格子点が すべて 端点になるような単純多角形が答えの候

補です．これを以下では「妥当な単純多角形」(simple polygonization) と呼ぶことにします．その中でできるだけ面積の小さいもの（できれば最小面積のもの）を求めよ，というのが今回の課題です．

入力の格子点



単純多角形その１



単純多角形その２

図 1 入力格子点の例と単純多角形の例

２．審査方法

従来は，木曜の昼の〆切時にプログラムを提出してもらい，それに対し，審査委員会が審査用の問題例に対してプログラムを実行し，その性能を評価する，という方法をとってきました．また，プログラムの実行は，3 分間〜 10 分間といったかなり短い制限時間のもとで行なっていました．

しかし，それでは従来のスパコンの使い方とは大きく異なってしまいます．スパコンは，非常に計算時間のかかるものを，スパコンの性能をギリギリまで生かして計算時間を何とか妥当な時間まで持ってきて解くために使用されています．ここで「妥当な時間」とは，通常，数時間〜数日間です．こうしたスパコンの使い方を模擬体験してもらい，スパコンの素晴らしさを実感してもらうために，今回は皆さんに実際に自分たちのプログラムを実行してもらう，という新たな競技方法をとることにしました．具体的には以下のような手順を考えています（注：水曜夕方までに最終版を連絡します）．

競技方法と最終日（木曜）の予定

1. プログラムの作成は 12 時をもって終了とする．
2. 午後 1 時より，各チームごと，作成したプログラムを用いて 1 つの審査用問題例に対して，それに対する解を 1 つ作成する．なお，審査用問題例は 60 〜 70 頂点のものを予定している．
3. 解の妥当性と面積により順位を決める．同じ面積の解を出した場合には同順位とする．
4. 各チームには解作成時間 1 時間と計算ノードを割り当てる．各チームは割り当てられた時間帯の中で，割り当てられた計算ノードを利用し，解を 1 つ作成し，指定した場所に置く．各チームにはレフェリー役として教職員 1 名とチューター 1 名が付く．
5. 割り当てられた時間帯の中で，プログラムをどのように実行し，また改変してもよい．ただし，実行させるプログラムについては，入出力等に関していくつかの制約を守ること．また，実行方法についても，レフェリー役の教職員の指示に従ってもらう場合がある．

２．課題攻略法（その１）：単純多角形の列挙アルゴリズム

単純多角形面積最小化問題この問題は難しいことが知られています．専門家が言うところのＮＰ困難な問題の一つです．また，単にＮＰ困難だけでなく，中途半端な方法がうまくいきそうにない問題のようです．

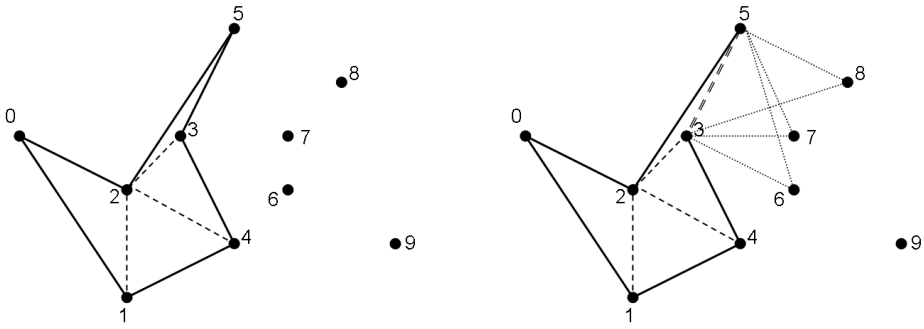
こうしたときの正攻法は，妥当な単純多角形をすべて調べつくす方法 — しらみつぶし法 — です．以下では，この方法を 基本戦略の１つ として説明します．

> **注意**　基本として「しらみつぶし法」を説明しますが，本当に全部を「しらみつぶし的に」調べるのがよいわけではありません．この基本戦略と以下の参考アルゴリズムをもとに，適当な「枝刈り」をして，有力な候補のみを調べるようにしないと，スパコンと言えどもまともな計算時間に収まらない可能性が高いです．

しらみつぶし法は，すべての妥当な単純多角形を調べる方法です．単純多角形を作る方法として，ここでは三角形を貼り付ける方法を説明します．たとえば図 2（左）のように，途中まで単純多角形ができていたとき，

(1) 現在の 外辺 (outer edge) の一つを選ぶ，
(2) その外側に，新たな格子点を端点とする三角形を貼り付ける

という方法です．



途中の単純多角形　　三角形の貼り付け方

図 2　三角形を貼り付けて単純多角形を作る方法

図 2（右）では，辺 (3,5) を選んだときの，すべての妥当な三角形の貼り付け方を示しています．格子点 9 を端点とする三角形は，他の格子点を内部に含んでしまうので貼り付けられません．ちなみに，三角形を貼り付けたときには，辺 (3,5) は，外辺から 内辺 (inner edge) になります．図 2（左）の単純多角形の内部の点線の辺は，このように外辺から内辺に変わった辺を表わしています．

2.1 アルゴリズム上の工夫：同一単純多角形の重複生成を避ける方法

原理的には，上記の方法で妥当な単純多角形をすべて作ることができます．ただ，単純にやると，同じ単純多角形が何度も出てきます．それを防ぐアルゴリズム上の工夫があります．

その基本となるのは頂点や辺の順序です．その順序をまず決めましょう．妥当な順序の一つとして，ここでは座標による 辞書式順序 (lexicographic order, lex. order) を導入します．

頂点（格子点）の場合には，まず x-座標の大小を考え，x-座標が同じならば，y-座標の大小で順序を決める方法です．たとえば，格子点 $(3,7)$, $(3,2)$ $(4,5)$ の大小関係は，

$$\text{格子点の大小関係：}\ (3,2)\ <\ (3,7)\ <\ (4,5)$$

となります．これからは頂点（格子点）は，この順序で 0 から番号が付けられているものとします．

辺は頂点の組で表わされますが，我々は，この頂点の番号の組で，しかも小さい頂点番号を先にした組で，辺を表わすことにします．たとえば，頂点 2 と頂点 5 を結ぶ辺は $(2,5)$ と表わし，$(5,2)$ という表わし方は使わないことにします．そして辺同士の順序関係も，格子点と同様，最初に小さい頂点の方の番号で，次に大きな頂点の方の番号で比較することにします．したがって，たとえば，辺 $(3,7)$, $(2,3)$ $(4,5)$ の大小関係は，

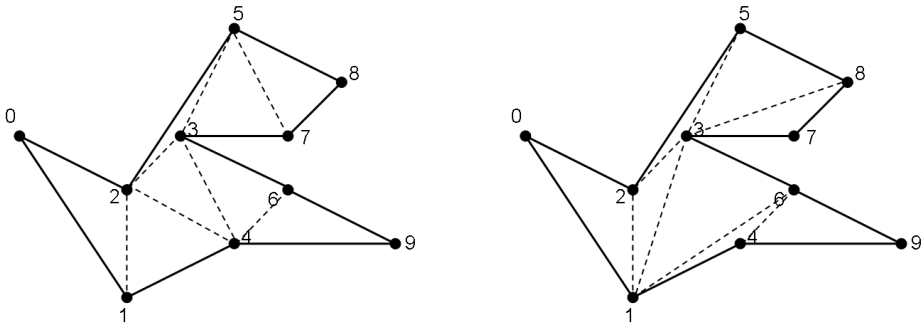
$$\text{辺の大小関係：}\ (2,3)\ <\ (3,7)\ <\ (4,5)$$

となります．辺 $(2,3)$ は $(3,2)$ とは表わさないことに注意して下さい．

さて本題に戻ります．同じ単純多角形が何度も作られる原因は二つあります．一つは，三角形の貼り付け方の順序の違いによるものです．たとえば図 3（左）のような単純多角形ができたとしましょう．これは，　(0,1,2) に　(1,2,4) を貼り付け，それに　(2,3,4)，　(2,3,5)，　(3,5,7)，(3,4,6)，　(5,7,8)，　(4,6,9) と，順に貼り付けて作ることができます．けれども，たとえば，

$$(1,2,4)\rightarrow\quad (0,1,2)\rightarrow\quad (2,3,4)\rightarrow\quad (2,3,5)\rightarrow\quad (3,5,7)\rightarrow\quad (5,7,8)\rightarrow\quad (3,4,6)\rightarrow\quad (4,6,9)$$

と作る方法もあります．この三角形の貼り付け方の順序の違いにより，同じ単純多角形が何度も作られる可能性があります．



ＯＫ単純多角形 (dual max)　　同じ単純多角形

図 3　異なる三角形分割 (triangulation) を持つ単純多角形

このように複数の順序で同じ単純多角形が作られる可能性があります．これを防ぐには貼り付け方の順序を１つにすればよいのです．具体的には，次のような基準を導入します．

(T1) 最初の三角形は頂点 0 を含む三角形とする．

(T2) ある辺 e を選んで，その外側に三角形を貼り付けたとき，その時点の外辺で e より小さな辺は外辺に固定し，以降でそこに三角形を貼り付けない．

最初の基準は簡単ですね．一方，(T2) については，少し説明を補足しておきます．たとえば図 2（左）で辺 $(3,5)$ に貼り付けましたが，この場合，その時点での外辺である辺，たとえば辺 $(3,4)$

は外辺に固定します．つまり，それ以降，そこに三角形が貼り付けられて，辺 $(3,4)$ が内辺に変わるようなことは起きないようにする，というのが (T2) です．たとえば，

$$(0,1,2) \to \quad (1,2,4) \to \quad (2,3,4) \to \quad (2,3,5) \to \quad (3,5,7) \to \quad (3,4,6) \to \cdots$$

という貼り付け方を禁止するわけです．このように規制しても，たとえば，

$$(0,1,2) \to \quad (1,2,4) \to \quad (2,3,4) \to \quad (3,4,6) \to \quad (3,5,7) \to \cdots$$

と貼り付ける方が大丈夫なように，できない単純多角形が生じることはありません．ちなみに，この基準を守って図 2（左）を作ると

$$(0,1,2) \to \quad (1,2,4) \to \quad (2,3,4) \to \quad (3,4,6) \to \quad (3,5,7) \to \quad (4,6,9) \to \quad (5,7,8)$$

となります．

同じ単純多角形が何度も作られる原因の二番目は，同じ単純多角形に対する異なる三角形分割 (triangulation) がある点です．たとえば，図 3（左）と（右）のように，同じ単純多角形が二通りの三角形分割（つまり，三角形の張り合わせ方）が出てきます．

これも，辺の間の順序を用いることにより防ぐことができます．

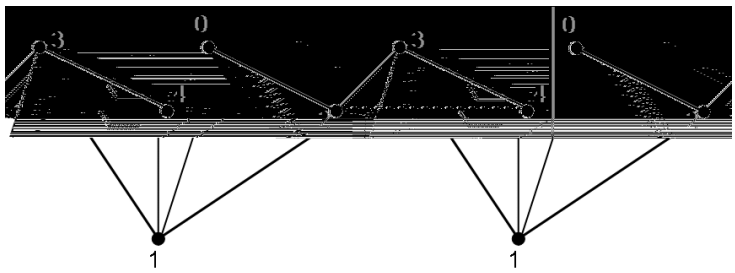


図 4 内辺とその dual 辺（左の細い線が内辺．右の点線が dual 辺）

まず用語を導入しましょう．単純多角形を三角形分割したときの内辺に対しては，各々，それを囲む四角形が存在します．その四角形のもう一つの対角線を，その内辺の dual 辺と呼びます．たとえば，図 4（左）の内辺 $(1,3)$ の dual 辺は，図 4（右）に示したように辺 $(2,4)$ です．ただし，dual 辺が無い場合もあります．たとえば $(1,2)$ を囲む四角形 $(1,3,2,0)$ には他に対角線がないので，$(1,2)$ には dual 辺はありません．

すべての内辺が，(もし存在すれば）その dual 辺に，辺の大小関係で勝っているような三角形分割を，dual-max 型三角形分割と呼ぶことにします．また，dual-max 型三角形分割で作られた妥当な単純多角形を「ＯＫ単純多角形」と呼ぶことにしましょう．図 3（左）は dual-max 型三角形分割になっており，したがって図 3（左）のような単純多角形がＯＫ単純多角形です．

このように定義すると，実は，どんな単純多角形も，ただ一つの dual-max 型三角形分割を持つことが示せます（興味のある人は証明してみて下さい）．しかも，次のような基準で三角形を貼り付けていけば，dual-max 型三角形分割を持つＯＫ単純多角形を作ることができます．

(D1) 辺 e を内辺として三角形を貼り付けられるのは，貼り付けにより，e の dual 辺が生じないか，あるいは生じたとしても，e より小さいかのいずれかの場合である．

たとえば，図 3（右）の単純多角形の三角形分割では，内辺 $(3,8)$ が，その dual 辺 $(5,7)$ に負けています．このような単純多角形ができたのは，$(3,8)$ に　(3,7,8) を貼り付けたからで，基準 (D1) を満たしていない貼り付けを行なったからです．

2.2 アルゴリズムの概略

以上の点を考慮して三角形を貼り付けていけば，同じ単純多角形が二度生成されるような非効率さは防げます．これに基づいて設計したのが次ページに示すアルゴリズムです．

このアルゴリズムでは，ＯＫ単純多角形をすべて生成します．このように，すべてを生成することを 列挙 (enumeration) といいます．ただし，ここでは簡単のため，最初の三角形は　(0,1,2) とします．つまり，　(0,1,2) に貼り付けてできるＯＫ単純多角形の列挙になっています．変数 `Vleft`, `Efixed`, `Eleft`, `I` は，それぞれ次のような情報を蓄えるのに使われています．

`Vleft` = まだ使われていない格子点の集合　　`I` = 内辺の集合
`Efixed` = もう外辺として固定した辺の集合，　　`Eleft` = まだ内辺になりうる外辺の集合

このアルゴリズムで単純多角形が列挙できます．ただし，再度注意しますが，これはあくまで 基本戦略 です．実際，この列挙法では，まともに動きません！次に述べる参考プログラムは，この列挙法に従ったものですが，そのプログラムでは頂点数が 50 くらいになると，手元のパソコンで単純多角形を１つ作り出すのに１日くらいかかる場合もあります．

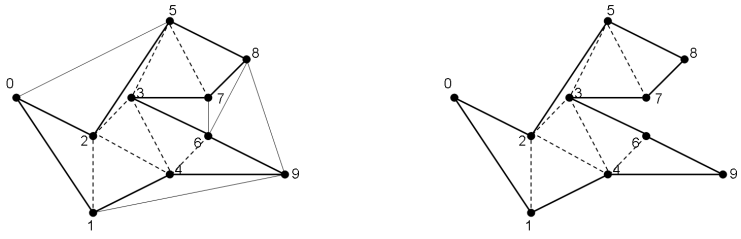
したがって，この戦略に沿ってプログラムを作る場合には大きな改良を考える必要があります．列挙（全部をくまなく作る）のは諦めて，適当によさそうなものだけを作っていく方法を考えるのがよいでしょう．また，一つの部分単純多角形での探索を適度なところで打ち切るなどの方法もあります．

３．課題攻略法（その２）：三角形分割→単純多角形アルゴリズム

上記のアルゴリズムをまともに使うと単純多角形を出すのだけでも大変な時間がかかります．実は，単純多角形だけならば，比較的簡単に出せるアルゴリズムがあります．以下に，その一つとして凸包の三角形分割から単純多角形を作る方法を紹介します．

3.1 三角形分割→単純多角形アルゴリズムの考え方

面積にこだわらないとすると，目標は図 5（右）のように，すべての入力格子点を端点とする単純多角形の作成です．それを無闇に三角形を付け足して作っていくのではなく，まずは，図 5（左）のような，入力点すべてを包含する凸多角形（凸包 (convex hull) といいます）に対する三角形分割をもとに，右のような単純多角形を作成するのです．



凸包の三角形分割　　$\Longrightarrow$　　単純多角形

図 5　凸包の三角形分割と単純多角形の例

アルゴリズム ＯＫ単純多角形の列挙 (enumeration) アルゴリズムの概略

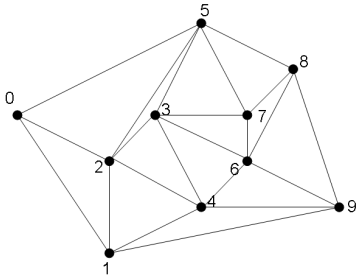
```
/*****************************************/
/* OK simple polygon enumeration procedure */
/*****************************************/
 Vleft = {3, 4, ..., n};
 Efixed = {};  Eleft = {(0,1), (0,2), (1,2)};

 void enumpgon(Vleft, Efixed, Eleft, I) {

    if(Vleft = empty) E = Efixed + Eleft をＯＫ単純多角形として出力;

    for( each e = (v1,v2) in Eleft) {
       Efixed' = { e' in Eleft | e > e };
       Eleft   = Eleft - Efixed' - {e};
       Efixed  = Efixed + Efixed';
       I = I + {e};

       for( each v0 in Vleft ) {
          if( 頂点 v0 が以下の条件を満たす
             (1)   (e,v0) 中に他の格子点を含まない.
             (2)   (e,v0) がすでに作られている他の辺と交わらない.
             (3) 条件 (D1) を満たす ) {

                enumpgon(Vleft - {v0}, Efixed, Eleft + {e1,e2}, I);

          }
       }

       I = I - {e};
       Efixed  = Efixed - Efixed';
       Eleft   = Eleft + Efixed' + {e};
    }
    return;
 }
```

注： + や - は集合への追加や集合からの削除の演算.

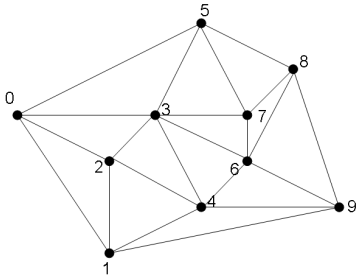
正確には，凸包の三角形分割 (triangulation) とは，すべての入力点を包含する凸包に対し，その内部に位置する入力点を端点として凸包を三角形に分けることです.

入力点が与えられるとその凸包は一つに定まります．しかし，単純多角形の場合と同様，同じ凸包に対しても，多数の三角形分割が考えられます．その中でも，攻略法（その１）の解説で出てきた dual-max 型の三角形分割が重要な役割を果たします.

与えられた入力点に対し凸包が一つに定まります．さらに，その凸包に対して dual-max 型の三角形分割が一意に定まります．それを $T_{\max}$ と呼ぶことにします.



三角形分割その１（$T_{\max}$）



三角形分割その２（左の１辺をフリップしたもの）

図 6 凸包の三角形分割の例

与えらた入力点に対して，その凸包を計算するのは高速にできます．さらに $T_{\max}$ は高速にできます．

しかも，$T_{\max}$ を元に，それを変形してすべて三角形分割を列挙することも可能です．簡単に言うと，三角形分割の中の１つの対角線を dual な辺にフリップすれば別の三角形分割ができ，それを繰り返すことで，すべての三角形分割ができます．ここでは詳細は省略しますが，このフリップをうまく行なっていくことで，すべての三角形分割をちょうど一回ずつ，しかも効率良く生成することが可能です．ただし「効率良く」というのは，１つの三角形分割を生成するのに必要な時間が瞬時，ということです．全体では，三角形分割の種類が非〜〜常に多いため，すべてを列挙するのはとても時間がかかります！

こうして１つの三角形分割が得られたとき，それを元にさらにそこから（複数の）単純多角形を作るのが今回の方法です．具体的には，三角形分割の１つの三角形を元に，それに三角形分割を構成する三角形を貼り付けていって単純多角形を作るのです．やり方は攻略法（その１）と同様です．違いは三角形分割を構成する三角形だけを貼り付けていく点です．もちろん，行き詰まる場合もありますが，無闇にに三角形を付けていくよりはるかに成功率（全部の点を使った単純多角形ができる確率）が高くなります．

付録１：参考プログラムについて

上記の攻略法２つ沿ったアルゴリズムをプログラム化したものを参考プログラムとして配布します．

(1) 攻略法（その１）に基づく参考プログラム：`pgon`

`search.c` ＝ 本体のプログラム　　`search.h` ＝ 型，共通変数，関数のプロトタイプ定義
`search_enum.c` ＝ 列挙手続き `enumpgon` の定義
`search_pre.c` ＝ 前処理手続き `precompEcross`, `precompEokvertex` の定義
`search_tools.c` ＝ 道具として用意した各関数の定義

(2) 攻略法（その２）に基づく参考プログラム：fromtri

tri.c ＝ 本体のプログラム　　tri.h ＝ 型，共通変数，関数のプロトタイプ定義
tri_pgon.c ＝ 三角形分割から単純多角形を列挙するプログラム（補助ルーチンも含む）
tri_pre.c ＝ 前処理計算：Tmax の作成など
tri_tools.c ＝ 道具として用意した各関数の定義

(3) 可視化のプログラム：simplepolygon.jar

単純多角形を作成していく様子を見るためのプログラム simplepolygon.jar も Java で用意しました．これは生成された単純多角形を次々と見ることにも使えます．また，凸包の三角形分割を見たり，さらには三角形分割から単純多角形を作成していく様子を見ることにも使えます．

simplepolygon.jar は単純な命令列を読み込んで実行するプログラムです．最初に格子点の座標を与え（その順番で格子点に番号付けをし），あとは，`+1 i j`「辺 (i,j) 描け」や `-1 i j`「辺 (i,j) を消せ」という命令を与えれば，その通りに描画します（i, j は格子点の番号）．また，「（単純多角形や三角形分割の）完成形である」という指示は `0 ...` のように 0 で始まる命令列で与えます（0 より右の ... は何を書いていても無視されます）．

現在の参考プログラムは，この Java プログラムへのこうした指令を標準出力に出すようになっています．それをファイルにとっておいて，simplepolygon.jar で読み込み，[Go] ボタンを押すと，単純多角形を作っていく様子を見ることができます．速度調整はスライドつまみでできます（左：高速，右：低速）．右端の [NextPoly] を押すと，次の完成形（単純多角形や三角形分割）にジャンプできます．

すべてをトレースする場合には，少し頂点数が多くなると指令が大量になるので，このソフトを使うには，頂点数 15 くらいまでがよいでしょう．

付録２：プログラム pgon 系における工夫

プログラムは上記のアルゴリズムに比較的忠実に従って作られていますが，効率を考え，いくつかの工夫をほどこしてあります．

(1) 前処理計算 search_pre.c について

新たに貼り付ける三角形の候補（アルゴリズム中の　(e,v0)）を選ぶときの高速化のために，列挙に先立って，次のような表を作っておく前処理計算を行なっています（注：集合は線形リストで表わす）．

Eokvertex[$v1$][$v2$] ＝ 辺 $(v1, v2)$ と三角形を作ったときに他の格子点が内部に入らない頂点の集合
Ecross[$v1$][$v2$] ＝ 辺 $(v1, v2)$ と交差する辺の集合

表 Eokvertex を使うと，　(e,v0) として候補を持ってくるときに，まず，条件 (1) を満たす v0 を選ぶことができます．Ecross は条件 (2) のチェックに使いますが，この条件は，列挙計算途中で，どの外辺が加えられるかによって変わってくるので，もう一工夫必要です（次を参照）．

(2) 手続き enumpgon での Vleft, Eleft の表わし方

集合 Vleft は，集合として持っているのではなく，対応する集合に入っているか否かを示す表（配列）で表わします．つまり Vleft は要素数 vnum 個の配列で（vnum は入力格子点数），Vleft[vid] $= +1$ で頂点番号 vid の頂点が Vleft に残っていることを表わします．

プログラム中では，その時点の外辺は Eleft と Efixed の区別なく，すべて E に格納されます．ただ，各辺には fixed という情報が付けられています．これが 0 ならば Eleft の要素，正ならば Efixed の要素であることを表わします．この値を変えることにより，未確定外辺から確定外辺へ辺の種類を変えるわけです．

このアルゴリズム中の条件 (2) を高速に調べるために，表 Echeck を使います．Echeck[$v1$][$v2$] には，辺 $(v1, v2)$ が，これまで付け加えられた辺と交差するか否かの情報を持たせます．この値が 0 で，$(v1, v2)$ に交差する辺がないことを意味します．

付録３：プログラム fromtri 系における工夫

三角形分割を列挙する際のフリップを効率的に行なうため，プログラム tri.c 等では各辺を，次のような少し凝ったデータ構造で表わしています．

```
typedef struct _estruct {
   int outedgeflag;        /* -1 = convex hull edge                 */
                           /*  0 = used as an inner edge            */
                           /* +k = fixed for k times                */
                           /* -2 = bad edge (used only in compTmax) */
   vidtype v1id;
   vidtype v2id;
   struct _estruct *pd;    /* dual edge                     */
   struct _estruct *pe[4]; /* square edges                  */
                           /* pe[i] = (i), i = 0, 1, 2, 3   */
} etype;
```

少し補足しましょう．各々の辺 e に対して，このデータ型が割り当てられます．

outedgeflag は，e が単純多角形の内辺になれるか否かを示すものです．辺 e が凸包の外辺の（単純多角形の内辺には絶対なれない）場合には，この値が -1 になっています．また，現在の探索で外辺になれない e（先の Efixed と同じ意味で）では，この値が正となります．

辺 e に対して関連のある辺はポインタで示しています．pd は e の dual 辺へのポインタ，pe[0] $\sim$ pe[3] は e 隣接辺です．各々下図のような順番で割当てられています（注：v1id $<$ v2id と仮定）．

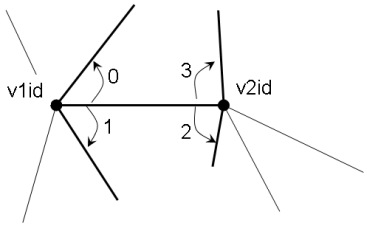


図 7 辺の隣接関係と pe[0] $\sim$ pe[3] への割り当て