エーデルで工夫したこと

はじめに

2014 年 12 月 17 日、私は首相官邸で「内閣府バリアフリー・ユニバーサルデザイン推進功労者表彰」の特命担当大臣表彰優良賞を授賞しました。その理由は、私が、盲学校に勤めていた 1991 年に図形を点訳するソフト「エーデル」を開発し、以来今日まで改良を続けながら、これを無償で提供してきたことです。現在ではエーデルは全国に普及し、点訳ボランティアの方々による点字教科書の製作などに活用されています。最近では、筑波技術大学の率いるグループが「チャート式 基礎からの数学」ＩＡ・ⅡＢ・ⅢＣ のすべてを点訳しました。この 3 冊が点字版では 200 冊余り、計 1 万 5 千ページにもなります。たいへんな労作ですが、グラフや図をたくさん含んでおり、エーデルがなければできなかった事業でした。

エーデルは、MS-DOS の時代から 4 半世紀近くも改良を続けてきた結果、現在では図形だけでなく文章も点訳することができ、点訳のあらゆるニーズに応えるものとなっています。

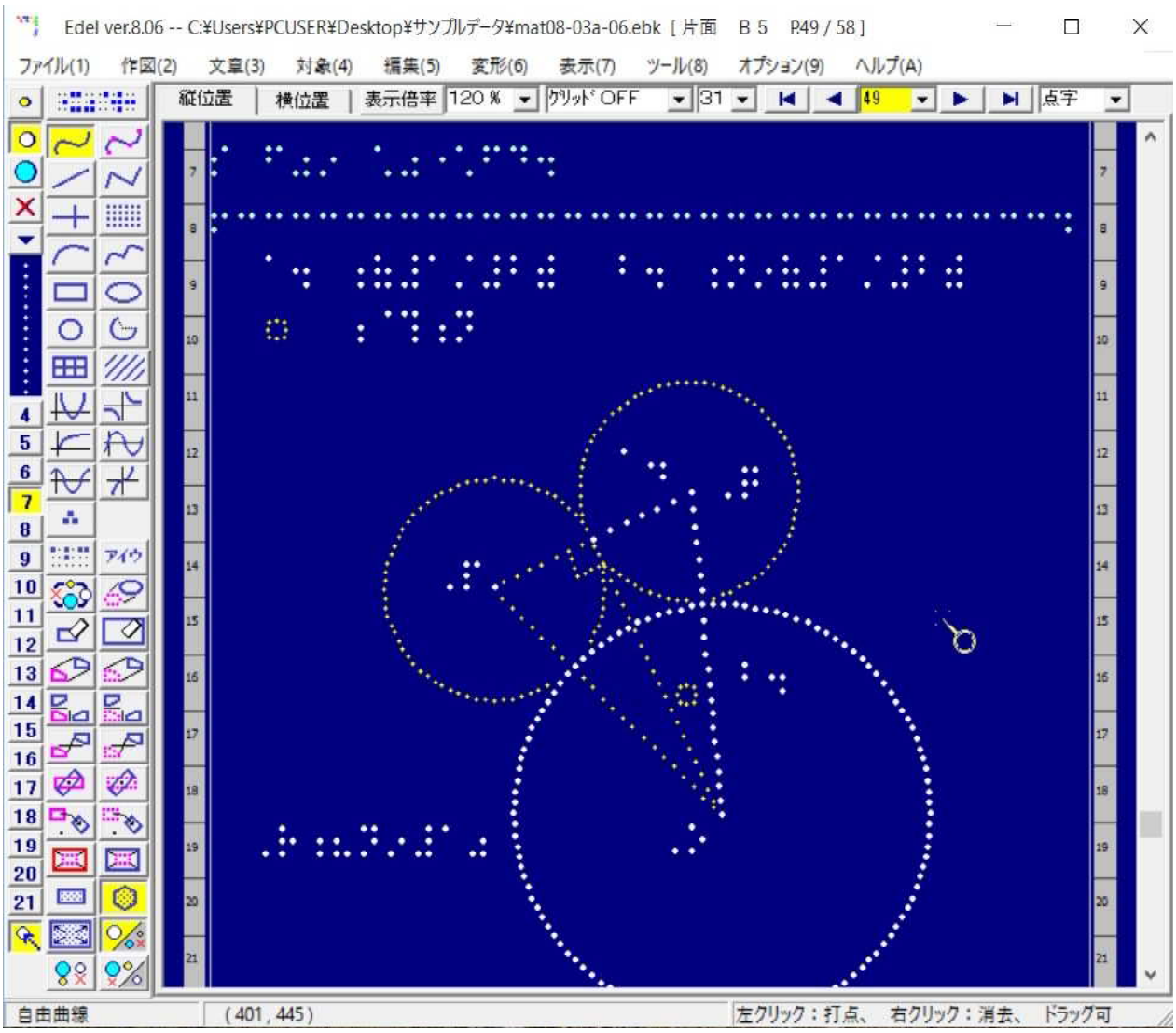

図 1　エーデルの画面

作図のモードには、フリーハンドの他、楕円や長方形、及び、各種関数のグラフなど、20 種類以上があります。それをいろいろに変形・編集する機能もたくさんあります。また、文章については、3 種類の入力方法をもっており、切り取り・コピー・貼り付けや検索・置換の機能、及び、校正作業を支援する独特の機能なども備えています。点字を知らなくても、ローマ字入力をするとエーデルが点字に変換してくれます。こうしたエーデルの機能のすべてをここで説明する余裕はありませんが、エーデルは、点訳者にとって「痒いところに手が届く」、図入り点訳本の製作になくてはならない、唯一無比のツールとなっています。（図１）

私がエーデルを作ろうと思ったきっかけは、鳴門教育大学大学院を卒業して盲学校で勤務することになり、全盲の男子生徒ひとりのクラスを担任することになったことでした。数学の授業は、文部省（当時）が提供する点字教科書を使って問題なくできましたが、その点字教科書に載っているような点図（点字と同じ凸点の列で描いた図やグラフ）を現場の教員が製作することはできませんでしたので、例えば、テストに図形問題を出題することは難しかったのです。当時、盲学校ではすでにコンピュータの利用が進んでおり、文章のパソコン点訳も始まっていて、パソコンで動かす点字プリンタが活躍していました。あるとき、ふと点字プリンタの説明書を見ると、点図を描く「プロッタモード」というものがあると書いてありました。読んでみると、確かに点図を打てるのですが、ひとつの点図を製作するためにひとつのプログラムを組まなければならないことが分かりました。このため、誰もこれを利用していなかったのでした。そこで、私は、パソコンの画面上にマウスで図を描くと、それをその通りに点字プリンタで印刷するパソコンソフトを作ろうと考えました。こうして、「絵が出る」エーデルが誕生しました。

私にどうしてそのプログラミングができたのかというと、鳴門教育大学大学院時代にプログラミングを自学自習していたからです。当時、自然棟 7 階にはコンピュータ室があり、C 言語のソフトもありました。私は Basic は少しかじっていましたが、「この際、本格的な C 言語を勉強しよう」と思い立ったのでした。C 言語の開発者カーニハンとリッチーによる『プログラミング言語 C』を読み、小さなテストプログラムを作ったりしました。盲学校では、これを実際の課題に応用したいと考えました。しかし、もちろんまったくの専門外ですので、現実の作業は試行錯誤の連続でした。例えば、点字プリンタに信号を送るときのプロトコルは RS-232C というのですが、そんなものは聞いたこともないし、普通の C 言語の教科書には載っていません。また、プログラミングにはミスを修正する「デバッグ」が付き物ですが、これがなかなか難しいのです。致命的なエラーが起きても、その原因が分からないことが度々でした。それに集中すると、周りが見えなくなり、上の空になって家族とのコミュニケーションに問題が発生するのでした。しかし、その度にミスの原因を突き止めて解決し、また、たくさんの創意と工夫を重ねて、エーデルを成長させてきました。

以下では、エーデルのプログラミングにおける工夫のいくつかについて述べます。

１．図形を等間隔の点列で描く

点図は一定の間隔をとった凸点の列で描きます。例えば、フリーハンドで曲線を描く場合、マウスをドラッグしますが、直前に打った点からの距離を絶えず計算しており、それが設定した間隔の値になった瞬間に次の点を打ちます。これはとくに問題ありませんが、例えば、１本の線分を描く場合は問題が起きます。線分を描く場合、始点と終点を指定しますが、その間の距離は設定した間隔の値の整数倍とは限りません。従って、始点から等間隔に点を打っていくと、終点のところで点の間隔が違ってしまいます。それで、できるだけ等間隔にし、かつ、始点と終点にも打点するために次のようにしています。

① 線分の距離（実数）を、設定している間隔の値（整数）で割った商（整数）を求める。
② 線分の距離（実数）を、①の商（整数）で割り、その答え（実数）を実際の間隔とする。

この実数値の間隔をとって線分を描くと、終点のところだけ間隔が違うということはなくなります。もちろん、パソコンの画面は正方形のピクセルで構成されており、打点する位置の座標はすべて整数なので、実際に打点する位置の座標は上記で求めた値を丸めたものになります。

円を描く場合は、極座標を使って計算し、等間隔にするための角度間隔を、線分の場合と同様にして求めます。つまり、

① 円の全周（実数）を、設定している間隔の値（整数）で割った商（整数）を求める。
② 全周に相当する角度 2 π（実数）を、①の商（整数）で割り、その答え（実数）を等間隔に相当する角度間隔（一定値）とする。

しかし、楕円ではこのような一定値の角度間隔を求めることはできません。楕円の周上で等間隔に打点するための角度間隔は絶えず変化するからです。それで、楕円については、数年前まで、フリーハンドの曲線と同様の方法で、始点（右端の点）から等間隔をとって描いていました。このため、終点（右端の点）のところでは間隔が違ってしまっていました。このことを修正してほしいという要望が点訳ボランティアの方から寄せられましたが、なかなか解決法は思いつきませんでした。これについて、やっと思いついた工夫は、楕円については、右端から反時計回りにフリーハンドと同じ方法で描いていき、終点（右端）の手前約 0.5 ラジアンについては円で描くというものです。この約 0.5 ラジアンについて、楕円を近似する円の中心と半径を計算で求めています。円の一部ならば、上記のようにして、等間隔に対応する一定値の角度間隔を求めることができます。（図２）

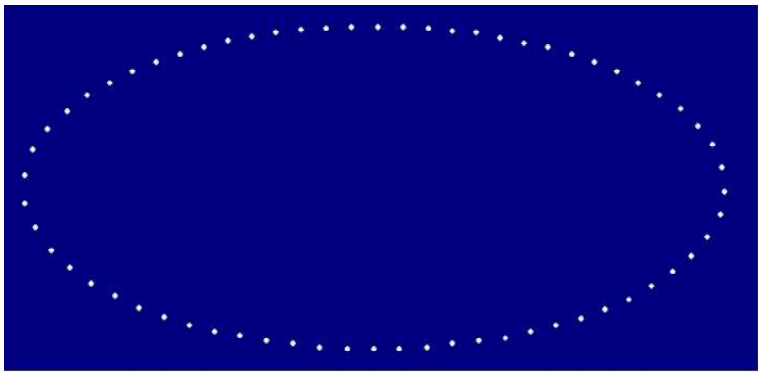

図２　右下約 0.5 ラジアンは円、右端で間隔の乱れなし。

２．ペイント機能の３モード

エーデルでは 15 種類のパターンで「ペイント」することができます。「ペイント」には 1）刷毛、2）オートフィル、3）領域指定の３つの方法があります。

「刷毛」は、マウスのドラッグに連れて、サイズを設定できる小領域を連続的に塗っていく、あるいは、マウスでクリックした位置で小領域を塗るものです。一般のお絵描きソフトの「ペン」に当たるものです。「ペン」ならば、何も考えずにクリックした位置を基準にして塗ればいいですが、エーデルでは「塗る」と言っても間隔をとった点のパターンで塗るので、クリッ

クした位置を基準にしたのでは、その位置は微妙に異なることが多く、クリックする毎にパターンの位置がずれることになります。このため、例えば２度塗りをすると、少しずれたふたつのパターンがいずれも打点されることになり、結果的に異なったパターンになってしまいます。ふたつの凸点が近づき過ぎると、重なったり、極端な場合は用紙が破れてしまいます。これを防ぐため、エーデルでは、どのパターンでも打点の位置は絶対的に決まっており、クリックした位置に依らないようにしています。（図３）

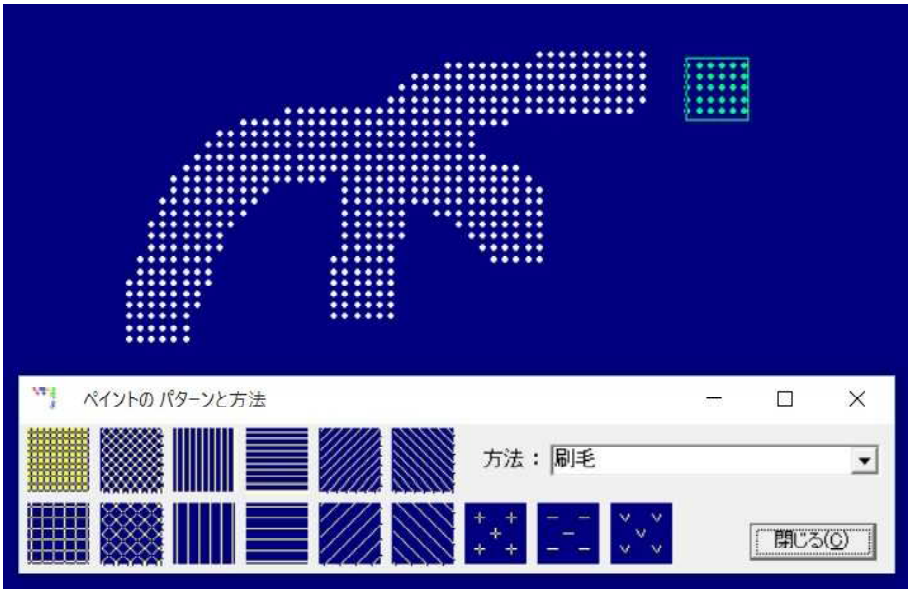

図３　「刷毛」でのペイント

「オートフィル」では、閉じた領域の内側をクリックすることでその領域を塗りつぶします。これは一般的な「塗りつぶし」と同じ機能です。しかし、一般的な「塗りつぶし」では、領域が完全に閉じていなければならず、ちょっとでも開いていると、そこから溢れ出して全体が１色になってしまったりします。ところが、エーデルでは閉じた領域というものはひとつもありません。図形はすべて間隔をとった点列で描かれるからです。そこで、エーデルでは、「オートフィル」＝「塗りつぶし」を実現するために次のような処理をおこなっています。

① 作業用の描画画面を別に用意し、そこへ本来の作図画面をコピーする。
② 作業画面において、描かれているすべての点について、各点の位置を中心とする、一定の半径（＝設定されている点間隔程度）の円内を調べ、そこに点があれば、中心の点とその点とを線で結ぶ。これで、作業画面においては、間隔をとった点列で描かれていた図形が、すべて線で描かれることになり、開いていた領域を閉じることができる。
③ クリックされた位置を含む閉領域を塗りつぶす。

この③の段階は、一般的な「塗りつぶし」と同じアルゴリズムでおこなえるはずですが、しかし、それはどういうアルゴリズムなのか？、私にとってはそれも問題でした。閉じた領域といってもいろいろな形があるわけですので、それがどのようなものであっても対応できなければなりません。すぐ思いつくのは、その領域内のあるピクセルについて周囲（上下左右の４点）を調べ、境界に達していなければ塗る、塗った場合はまたそのピクセルについて周囲を調べて・・・ということを、クリックで指定した位置のピクセルから始めて、塗るべきピクセルがなくなるまで続ける、という方法です。こういう動作は、ひとつの関数（ひとまとまりの処理をおこなうルーチン）の内部に当の関数を含んでいる「再帰関数」によって実現できます。しか

し、実際にこれを作って動かしてみると、再帰の階層が深くなりすぎて、スタックオーバーフロー（メモリの不足による致命的なエラー）を起こしてしまいました。そこで、再帰の階層があまり深くならないようにする工夫が必要でした。それは、あるピクセルについて、まず左右に塗れるところまで塗り、塗った各点から上下の 2 点を調べて、塗れる場合はまたそこから左右の塗れるところまで塗る、また上下を調べて・・・ということを繰り返すアルゴリズムで実現できました。

「領域指定」では、長方形などの領域を描くと、その内部を塗ります。以前は、領域の形として長方形に限定していました。長方形なら、対角の 2 点を指定することですぐ描けますし、その内部を走査することも容易です。しかし、長方形だけでなく、任意の多角形領域を指定して塗りたい、という要望が以前からありました。多角形を描くことは難しくありませんので、多角形を描いた後で、塗る場所を指定するために、その内部を 1 クリックすることにするなら、上記の「オートフィル」を使うことによって実現できます。しかし、私は、多角形の内部を指定する 1 クリックを省きたいと考えました。このためには、多角形が描かれた後、どこがその内部であるのかを自動的に判断する必要があります。そこで、ある位置がその多角形の内部であるかどうかを、その位置から画面の端まで引いた直線が何回多角形の辺（＝領域の境界）と交わるかを調べて判断することにしました。内部であれば、これが奇数になるはずだからです。ただ、例外もあるので、その位置から上下左右の 4 方向を調べています。

この、多角形領域を指定して、どこがその内部であるかを自動的に判断するアルゴリズムは、複写や移動などの対象領域を指定するときにも使っています。これによって、図が込み合っている中から、移動したい部分だけを指定して移動することなどができます。（図４）

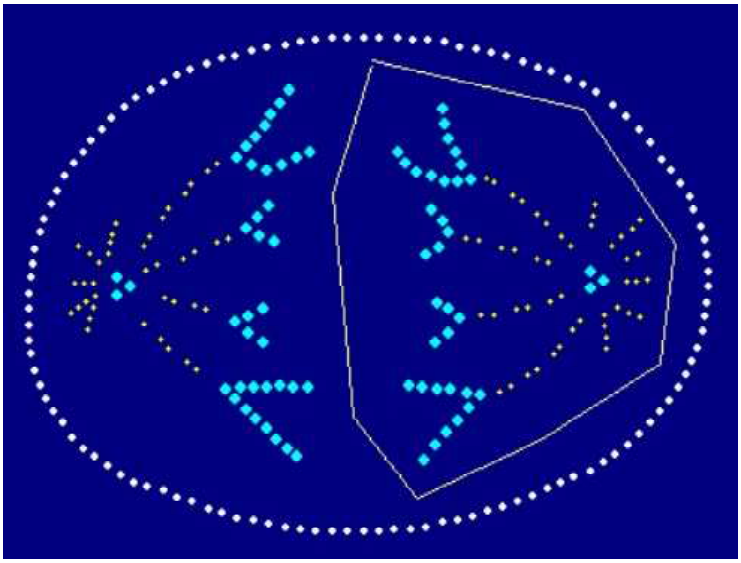

図４　右側だけを多角形領域として指定している。

３．スプライン補間曲線

スプライン補間曲線というのは、いくつかの点を次々に指定していくと、それらを、指定した順に滑らかな曲線で結ぶものです。これ自体は私のオリジナルではありません。以前から、これを実現すべく、『C による スプライン関数』という本を何度も読みかけましたが、その都度挫折していました。だいたい、たとえスプライン関数の数学的な意味が分かっても、実際にプログラミングするにはいくつもの問題があります。しかも、エーデルの場合は間隔をとった点列で描かなければなりません。とくに、閉曲線も描く必要があり、媒介変数表示にしなけれ

ばなりません。その媒介変数として何をとれば良いのか、具体的なイメージが持てませんでした。実際には、単に媒介変数を t として、i 番目の区間の曲線上の点（x,y）について、$x=f_i(t)$, $y=g_i(t)$ （ $i \leqq t \leqq i+1$ ）とすれば良いのでした。そして、後から考えると、t は、曲線上を端から点が動いていくときの経過時間に当たるものでした。

スプライン補間曲線を求めるためには、指定した点で区切られる各区間毎に 3 次関数（上記の f_i , g_i ）を決定しなければなりません。それは次の 3 つの条件を満たすようにします。

① 指定した点の位置で曲線が繋がる、つまり、連続である。
② 指定した点の位置で、繋がる 3 次関数の第 1 次導関数の値が等しい。
③ 同じく第 2 次導関数の値が等しい。始点と終点の位置では第 2 次導関数の値を 0 とする。

結局、すべての区間の 3 次関数を決定するためには、かなり未知数の多い連立一次方程式を解かなければなりません。この計算（ガウス・ジョルダン法）と結果の描画を、経過点を増やす毎におこないます。（図 5）係数行列の次数が高くなると誤差が発生して計算が破たんすることがあるので、このプログラミングには注意が必要でした。

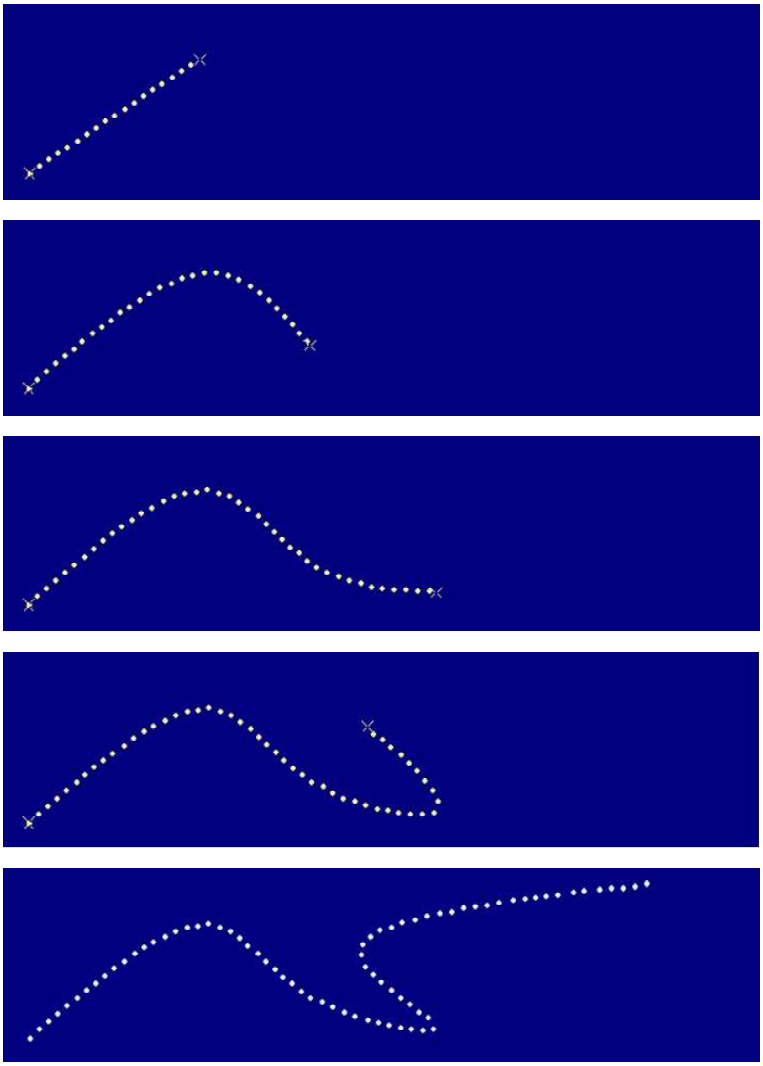

図 5　スプライン補間曲線を描く

４．画像の自動点図化

一般的な BMP ファイル、JPEG ファイルの画像を自動的に点図に変換することができます。これも点訳ボランティアの方から要望されていたものですが、あるとき集中的に考えて実現しました。まず輪郭を抽出しなければなりませんが、このために、ある明度を境にして明るい部分は真っ白に、暗い部分は真っ黒にします。これを「２値化」というのですが、これが「画像処理」において一般的に行われるものであることを後になって知りました。さて、この２値化画像の白と黒の接しているところが輪郭ですので、その部分のみ、または、黒い部分のすべてを点図にすれば良いわけです。これには、次のようなアルゴリズムを考案しました。まったくのオリジナルです。

２値化画像において、端から走査しながら、黒いピクセルが見つかればそこに打点し、かつ、その周囲の一定半径（＝設定している点間隔）の内部を黒以外の色に替える。

カラー写真などは自動点図化してもそのままでは実用になりませんが、白地に黒の線で描かれた白地図などはきれいに点図にできます。（図６、７、８）

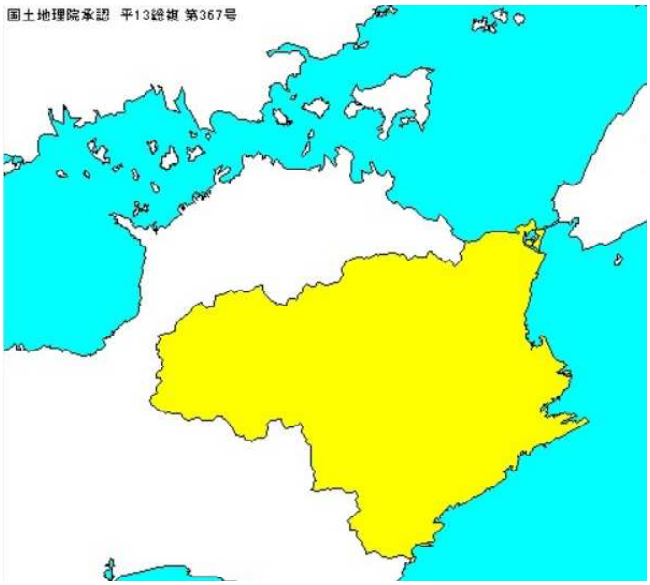

図６　元の画像

図７　２値化画像

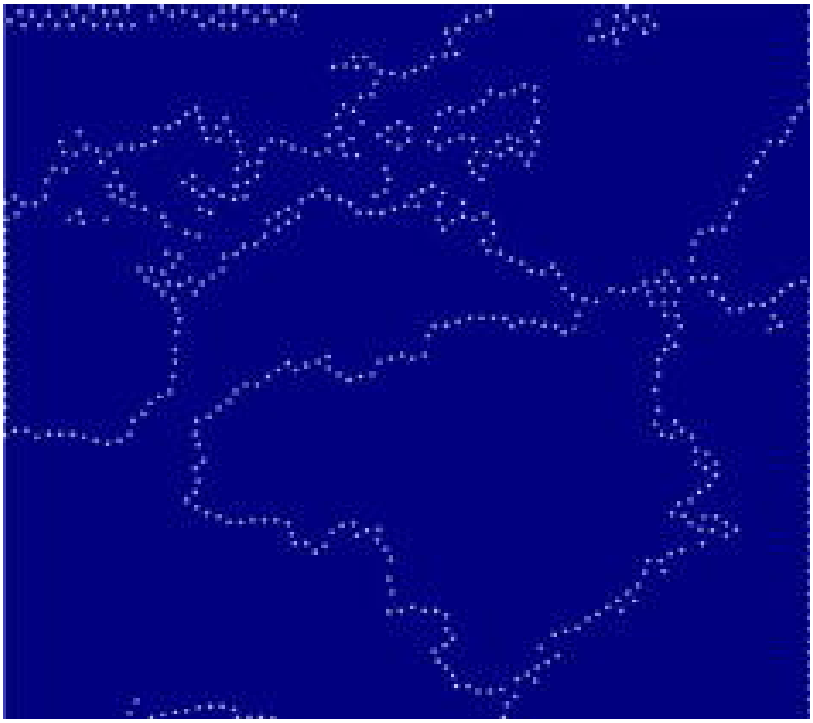

図８　点図化

これを使って漢字などの文字も点図化できますが、文字の場合は、骨格だけ＝線画の点図にする方が望ましいと考えられます。このためには、ゴシック体などの太さのある文字を、まず「細線化」する必要があります。「細線化」についてはすでに研究されていて、いろいろな方法があるようですが、代表的なものを使って実際にやってみると、あまりうまくいきませんでした。確かに太さ１の線画にすることができるのですが、不要な「ひげ」が生じます。（図９）

それで、私は、独自にいろいろと試みました。その結果、結局、黒い部分について、その「表皮」の黒いピクセルの層を１枚ずつ剥いでいくのが良いと思いました。これで「ひげ」は出ませんが、しかし、いつまでも続けると真っ白になってしまいますし、ひとつの文字でも太さが違うところがありますので、何層剥いだ時点で止めるかが問題になります。それで、この夏までは、１回クリックする毎に１層剥がすようにし、何時止めるかはユーザーが判断するようにしていました。これだと、完全な線画にはならず、少し太いところが残ってしまいますが、これ以上はできないと考えて諦めていました。

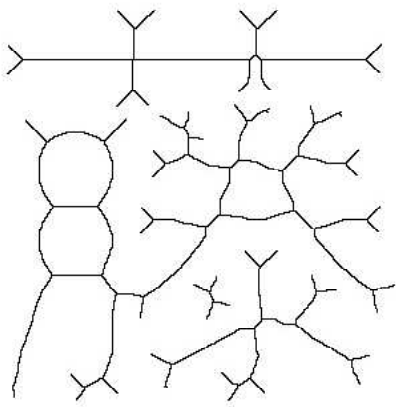

図９

ところが、今年の夏休み前に、やはり点訳ボランティアの方から「どうにかならないか」という要望が寄せられました。それで、夏休みの宿題のつもりで、この夏、改めて取り組みました。次のようないろいろなアイディアが浮かびました。

A）文字のいろいろな場所において４方向に「太さ」を調べ、その最頻値から、「表皮」を剥ぐ走査を何回やるのが適当か求められるのではないか。

B）上のように「太さ」が調べられるなら、文字のいろいろな場所において、その中央の位置が分かるのではないか。

C）白地を海、文字の黒い部分を陸に例える。そして、陸地のすべての位置は海岸からの距

離に比例して高くなっているとする。陸地のすべての位置について海岸までの距離、すなわち標高を求め、稜線を抽出すれば良いのではないか。

それぞれを試してみたり、それらを組み合わせてみたりした結果、この夏の時点での結論としては、次のようにすることにしました。

① 上の A）によって「表皮」を剥ぐ回数を決定し、実際に剥ぐ。ここであまりやり過ぎないように注意する。この処理によって「ひげ」の発生を防ぐ。
② 今度は「表皮」の１点１点について、それを白にすることによって「その辺りが真っ白になってしまうのでなければ」白に替える。

これで、一応満足のいく結果が得られます。（図１０）ただ、②の条件をどうやって判断するかが問題ですが、これはいろいろと試した末の「結果オーライ」で解決しました。また、ユーザーが手動で若干の点を追加したり消したりして、結果を調整できるようにもしました。
「文字の細線化」については、今後も追及していきたいと考えています。

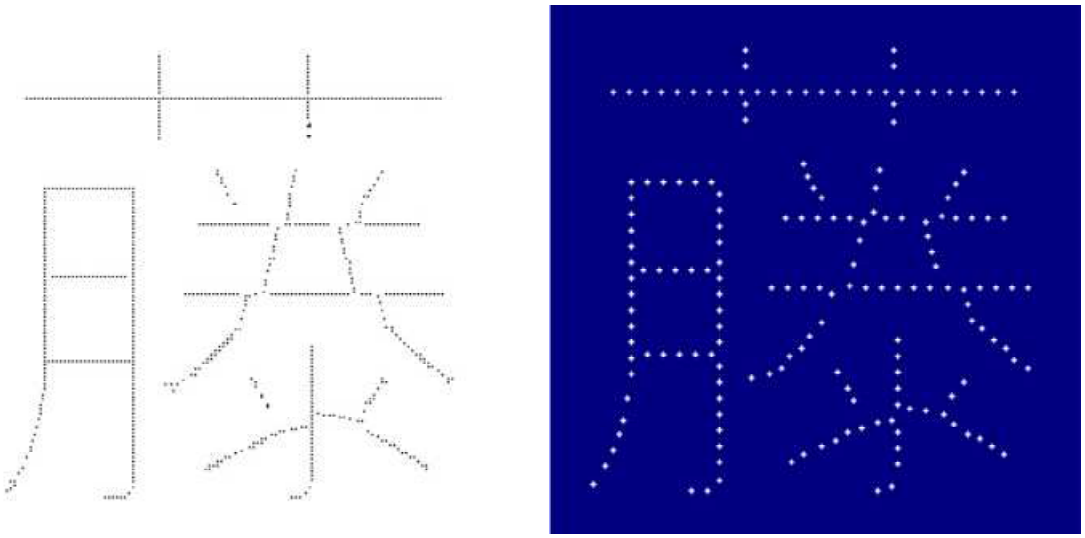

図１０　「藤」を細線化したものとその点図化

おわりに

私がプログラミングを勉強できたのは、その環境を与えてくれた鳴門教育大学大学院のおかげです。この場を借りて感謝申し上げます。
もちろん素人ですので、習得したプログラミング技術の内実は大したことはありません。それでも、エーデルは広く普及し、フリーソフトながら、現在では図形点訳ソフトのデファクトスタンダードになっています。そのため、この事業を将来に渡って継続していくことが社会的に求められていると言っても過言ではありません。それを担ってくれる、プログラミングの知識とボランティア精神の両方を持った若い人が現れることを期待しています。

８８Ｍ０５　　ふじのとしひろ
徳島県立城東高校　088-653-9111