システム・ソフトウェア開発

画像処理のためのソフトウェア：
(3) 粒子像の識別と輪郭線の抽出

中野 司*・藤井 直之**

Softwares for Digital Image Processing：
(3) Cluster Labeling and Perimeter Extraction

Tsukasa NAKANO* and Naoyuki FUJII**

ABSTRACT : Softwares of our image processing system are explained. 'exgrain' is a program available for labeling all grains in the binary digital image in order of their size. All perimeter of the grains can be extracted by 'exedge'. Since data structure of the perimeter is rather complex, we present the data converters which are useful when the data is analyzed.

KEY WORDS : Image processing, Grain-shape analysis, Network analysis, Perimeter

1. はじめに

画像処理（もしくは画像認識）のプロセスは、対象とする画像データの種類や処理目的によって非常に様々である。しかしながら、そのプロセスを細かく分けると、例えば画像データをスクリーンやプリンタに出力することはもちろんのこと、それ以外にも画像中の輝度の出現頻度の分布を調べたり、また、輝度の異なる領域の輪郭線（等高線）を取り出したりすることはもとなるデータや処理目的がまったく異なる場合であってさえもしばしば共通して現われる。このように、画像処理のプロセス全体を細かく分割してそれぞれに応じた小さなプログラムを作成しておき、処理目的に応じてそれらを組み合わせて（シェルファイルもしくはバッチファイルとして）使用するというのが、本文も含めた一連の論文で我々が紹介していこうと思っている画像処理ソフトウェアの基本的な考え方である（この考え方は、実はUNIXのソフトウェアの基本となる考え方と同じであるが）。第1表に我々の開発した画像処理シス

テムのソフトウェア群の一部を示す。

我々がおもに行おうとしている画像処理の流れの大筋を簡単に示すと第1図のようになる。第1図でオリジナルの画像としてあるものは、おもに顕微鏡（光学顕微鏡、走査型電子顕微鏡）で得た鉱物粒子の含まれる画像データである。ただし、それ以外でも、例えばリモートセンシングで得られたデータや数値シミュレーションの結果などを

第1図 画像処理の流れ

* 神戸大学自然科学研究科。現在、地質調査所。Graduate School of Science and Technology, Kobe University, Nada, Kobe 657, Japan. Now at Geological Survey of Japan, 1-1-3, Higashi, Tsukuba 305, Japan.

**神戸大学理学部。現在、名古屋大学理学部。Faculty of Science, Kobe University, Nada, Kobe 657, Japan. Now at Faculty of Science, Nagoya University, Chikusa, Nagoya 464, Japan.

第 1 表

画像データの取り込み、前処理

<i>tip</i>	他のアプリケーションプログラムと共存可能な、バックグラウンドで走る基本的な画像処理を行うためのフロントエンドプロセッサ。NEC pc9801 + ALTEC 画像処理ボード用。
------------	--

画像の輝度出現頻度分布解析と二値化

<i>bmhist</i> *1	ビットマップファイル内の画像データの輝度出現頻度分布の調査を行う。
<i>smooth</i>	多項式近似によって輝度分布をスムージングし、ピークサーチを行う。
<i>sffit</i>	輝度分布のピークサーチをスプライン関数近似によって行う。
<i>gfit (lfit)</i>	輝度分布の Gaussian (Lorentz) fit を行う。
<i>bmbin</i> *1	画像データを指定された輝度値で二値化し、新しいビットマップファイルを作成する。

二値画像中の (粒子) 像の識別とその輪郭線の抽出

<i>exgrain</i> *1	画像中の 0 でない輝度をもつ画素のクラスタ (粒子像) を識別し、各クラスタをそのサイズ順に輝度をつけてぬり分けたビットマップを作成する。また同時に、クラスタの形を楕円近似した情報を得る。
<i>exedge</i> *1	<i>exgrain</i> で色分けしたすべてのクラスタの輪郭線を抽出して、チェーンコード化する。
<i>dispedge</i> *1	<i>exedge</i> で得た粒子像の輪郭線、 <i>exgrain</i> で粒子形状を近似した楕円を表示する。

粒子の形状解析

<i>grainsz</i>	画像中の 0 でない輝度をもつ画素のクラスタ (粒子像) のサイズ分布を調べる。
<i>eachedge</i> *1	<i>exedge</i> で得た粒子像の輪郭線のうち指定した粒子のものだけを取り出して、 <i>fdfl</i> , <i>cflfd</i> の入力に合うようにする。
<i>fdfl</i> *2	<i>eachedge</i> で得た粒子輪郭線のデータから折れ線法を用いて粒形のフラクタル次元を求める。
<i>cflfd</i> *2	<i>eachedge</i> で得た粒子輪郭線のデータから粒形のフーリエディスクリプタを求める。

ネットワーク解析

<i>exprmt</i> *1	<i>exedge</i> で得た粒子像の輪郭線のうち指定した粒子のものだけを取り出して、 <i>tricut</i> の入力に合うようにする。
<i>tricut (tc2)</i> *3	<i>exedge</i> で得た輪郭線内部を三角形で分割する。
<i>disptri</i> *3	<i>tricut</i> で得たネットワーク領域の輪郭線内部を分割した三角形を表示する。
<i>dispNet</i> *3	<i>tricut</i> で得たネットワーク領域の輪郭線内部を分割した三角形の情報からネットワークの骨格 (スケルトン) を求める。
<i>prune</i> *4	<i>tricut</i> で得たネットワーク領域の輪郭線内部を分割した三角形の情報からネットワークの骨格 (スケルトン) を求め、さらにネットワークのポアとチャンネル領域の幾何学的な情報を抽出する。
<i>pcnet</i> *4	<i>prune</i> で得たネットワークの幾何学的な情報からネットワークを電流や粘性流体が流れたときの各チャンネルの '抵抗値' を計算する。
<i>netsolve</i> *4	<i>tricut</i> , <i>prune</i> で得たネットワークの端にポテンシャル (電圧, 圧力) 差が与えられたときの各ポアでのポテンシャル, チャンネルを通る流量を <i>pcnet</i> で計算したチャンネルの抵抗値からキルヒホッフの法則を用いて計算する。

ビットマップの表示など

<i>bm2scr</i> *5	カラービットマップファイルの画像データを NEC pc9801 のグラフィックスクリーンもしくは SONY NEWS のフレームバッファに表示する。
<i>cbm2xwd</i>	カラービットマップファイルの画像データを X-window (version 11) 上に表示する。
<i>inkjetbm</i> など *6	カラービットマップファイルの画像データを (カラー) プリンタにハードコピーする。

参考文献: *1 本文参照, *2 藤井・中野 (1988), *3 中野・藤井 (1991a), *4 中野・藤井 (1991b), *5 中野・藤井 (1989), *6 藤井・中野 (1989)。

用いることもできる（我々はそのような応用も実際に行っている）。このような画像データは、取り込みに用いられる装置などによって画素数や輝度の範囲が様々で取り扱いが不便である。そこで、それを我々の研究室の標準的な画像データのフォーマットに変換する。このような標準フォーマットとして、我々は bitmap と呼んでいるものを採用している（中野・藤井，1989）。次に画像データの二値化を行う。二値化に用いる輝度のしきい値選択の仕方は以降の画像処理の結果に大きく影響するため、できるだけ客観的かつ系統的に行う必要がある。現在のところ、これは bitmap 化されたもとの画像の輝度値の頻度分布にもとづいた、その Gaussian fit など（第 1 表）によって行っているが、これらはどれも決定的なアルゴリズムでなく、実際にはデータの種類によって様々な手法を試してみる必要がある。

画像を二値化した後は、画像認識の範疇にはいるプロセス群である。本文では、この画像認識のプロセスの最初の、画像データ内に散在する（粒子）像の識別（ラベリング）とその輪郭線の抽出を行うためのソフトウェア、‘exgrain’ と ‘exedge’ について説明する。これらと同等のソフトウェア ‘grain’ と ‘edge’ を我々はすでに以前に紹介している（中野他，1988）が、ここで紹介するものは以前のものと比べると様々な制約（ハードウェアへの依存性、画像データのサイズに対する制限など）をできるだけなくした、しかも高速なソフトウェアになっているので、今回取り上げることにした。

なお、exedge の出力ファイルはそれによって得た情報がすべて書き込まれた複雑な構造をしているので、その一部だけを必要とする、このプロセスに続く粒子形状の解析や画像データ内のネットワーク構造の解析（中野・藤井，1991a, 1991b）のプログラムにとっては取り扱いにくい。そこで、本文では、exgrain の出力をそれ以降の解析のソフトウェアの入力に合わせるためのファイル上のデータ構造の変換を行う（必要なデータだけを抽出する）ためのソフトウェアなどについても説明する。また、第 1 表に示したもののうち、それだけでは独立して取り上げるほどのものではない、いくつかの小さなソフトウェアも本文でまとめて紹介する。

2. アルゴリズム

2. 1. exgrain

exgrain で行われる処理は、主目的である画像中の粒子像の識別とラベリング、およびそれに付随的な粒子像の楕円近似の二つの部分に分けることができる。

画像中の粒子像の識別、ラベリングの処理は以下の三つのステップからなる。まず画像中に散在する粒子像（輝度値 0 でない画素のクラスタ）の識別とそのサイズの調査を領域塗りつぶしのサブルーチン（中野・藤井，1989）を

用いて行う。クラスタ内での画素のつながりなどの取り決め、および塗りつぶしのアルゴリズムについてはすでに詳しく述べた（中野他，1988；中野・藤井，1989）ので、ここではその説明は省略する。クラスタの識別は、すべての画素を順に走査して、0 でない輝度を持つ画素ですでに塗りつぶされたクラスタに属していないものがあればそれを始点とした、各クラスタごとに異なった色を用いた塗りつぶしによって行う。次に、クラスタのサイズ情報にもとづいて、各クラスタにサイズの大きいものから順番に 1 から始まる番号（ラベル）をつける。そして、この番号を輝度値とするように各クラスタの画素を色分けした新しいビットマップを作成する。

我々の作成した領域塗りつぶしのためのサブルーチンは、その引き数として与えられた（ユーザーの作成した）サブルーチンを画像データのアクセス（輝度の読み取り、書き込み）の際に使用する（中野・藤井，1989）。exgrain でこのサブルーチンを使う際にも、クラスタの塗りつぶしと同時にそれぞれのクラスタの重心（一次のモーメント）や二次のモーメントの調査など、クラスタの楕円近似に必要な量の計算を行うサブルーチンを引き数として与えている。各クラスタの楕円近似は以下のようにして行う。まず、楕円の長軸および短軸方向と平行な二直線を求める。これらは、クラスタに含まれるすべての画素から直線におろした垂線の足の長さの自乗和が極値（長軸の場合は極小値、短軸は極大値）をとるように決める。そして、これら二軸方向の楕円半径を、近似楕円の面積がもとのクラスタの面積に等しくなるように、上記の二直線の垂線の足の長さの自乗和の平方根の比から計算する。楕円近似の計算の詳細はすでに述べた（中野他，1988）ので、ここでは省略する。

2. 2. exedge

exedge は、exgrain で作成した粒子像（輝度 0 でない画素のクラスタ）ごとに輝度値を変えた（背景は輝度値 0 の）カラービットマップから、それぞれのクラスタの輪郭線を抽出する。輪郭線の定義やその抽出の方法についてはすでに述べた（中野他，1988）ので、ここでは詳しくは説明しない。ここで用いているアルゴリズムを一言で言うと、物体（クラスタ）を常に同一の側の片手で触りながらそれを一周してその形状（輪郭）を求める、というものである。

画像中の粒子像には、ドーナツ状に穴があいていて二本以上の輪郭線がある場合もある。これは、‘ネットワーク構造’（中野・藤井，1991a, 1991b）を持つ画像の場合には本質的である。このようなクラスタの‘外側の輪郭線’（どんなクラスタにも存在する）と‘内側の輪郭線’（内部に穴

のあるクラスタにだけ存在して、その場合には複数あることもある)は、輪郭線の大局的な周回方向によって区別できるようにする(外側の輪郭線は時計周り、そうでない場合は反時計周り)。ただし、コンピュータで取り扱う画像上での座標系は通常のもの(右手系; 右向きに x 軸正の方向を取ると y 軸は上向きが正)とは逆の左手系であるため、外側の輪郭線なら時計周りの周回方向が x 軸を y 軸に重ねあわす方向であることに注意する必要がある。

3. プログラムの説明

付録にC言語で書いたプログラムのソースリストを示した。これらは Turbo-C および UNIX-C でコンパイルして走ることを確認済みである。ただし、UNIX-C ではソースリスト中で宣言している include file の一部が存在しないというエラーが起こることもあるので、その場合には対応する include 宣言を取り除く必要がある。i8086系のコンピュータで Turbo-C などのコンパイラにかける場合には、メモリモデルとしてコンパクト、もしくはラージモデルを指定する必要がある。

それぞれのプログラムのコンパイルに必要なファイルは、Turbo-C のプロジェクトファイル(拡張子 '.prj')として書き並べたものを付録のソースファイルとともに示してある。これらのプログラムで使用しているビットマップを取り扱うサブルーチン群についてはすでに詳しく説明した(中野・藤井, 1989)。bmhist, dispedge, eachedge, exprmt のグラフィックス出力は、プロジェクトファイルの 'glio' ('glio.c'), 'g2' ('g2.c') で示したファイルに含まれるサブルーチンを用いて行っている。このうち glio は NEC pc9801 (もしくは EPSON pc286)用のハードウェアに依存したプリミティブなグラフィックス用サブルーチンパッケージである。g2 は UNIX マシンでも利用できる論理的な二次元座標系(window)でのグラフィックスサブルーチンを含んでいる。bmhist などからは g2 に含まれる七つのサブルーチンだけを使用しているので、プログラムを移植する際には以下の g2 のサブルーチンの仕様に合うようにそれぞれのハードウェアに対応したサブルーチンを自作しさえすればよい。

```
void initscr ()
```

グラフィックススクリーンを初期化する。

```
void clrscr ()
```

グラフィックススクリーン上を消去する

```
void window (double x1, double y1,
```

```
double x2, double y2, int flag)
```

グラフィックスを行う範囲を指定する。 y 軸正の向きが flag が 0 なら 下向き(座標系が左手系)、それ以外なら 上向き(右手系)になる。

```
void movep (double x, double y)
```

線分の始点を移動する(描画動作は伴わない)。

```
void plotp (double x, double y, int color)
```

color で指定した色の点をうつ。

```
void drawl (double x, double y, int color, int style)
```

一つ前に行った描画動作の終点から (x, y) まで color, style で指定した色、種類(点線、実線)の線分を引く。線分の色、種類は N88-BASIC の指定の仕方と同じである。

```
void plots (double x, double y, char *msg, int color)
```

色 color で msg で示した文字列を (x, y) がその中央にくるように表示する。

4. プログラムの使い方

付録に示したソースリストのプログラムはどれも、オペレーションシステム(MSDOS, UNIX)のコマンドラインからそれらのプログラムが必要とするパラメータを引き数として与えて起動するようになっていて、起動後の会話的なパラメータの入力はない(したがって、これらのプログラムを組み合わせるパッチファイルやシェルフファイルとして走らせることができる)。以下では、これらのプログラムの起動時のパラメータ指定の仕方について説明する。なお、起動時のパラメータのうちで中括弧でくくってあるものは、それがオプション指定であることを示していて省略可能である。

4. 1. bmhist

画像中の輝度出現頻度分布を調べる bmhist は以下のようにして起動する。

```
bmhist file {x1 y1 x2 y2}
```

'file' とは、調査すべき画像の入ったビットマップファイルの名前である。オプション指定の $x1, y1, x2, y2$ は、ビットマップ上で調査する領域の指定に用いる。この時、調査領域は $(x1, y1)$ を座標値の最小値、 $(x2, y2)$ を最大値とする長方形領域になる。この指定がなければ、調査はビットマップのすべての画素について行われる。

プログラム起動後は、グラフィックススクリーンに輝度の出現頻度分布を示すヒストグラムが描かれる(縦軸が輝度値、横軸が画素数)。その後、標準出力に、

出現した輝度の総数 (= 輝度の最大値 + 1)

輝度値 (0 ~) とその出現頻度 (画素数)

...

が書き込まれる。この出力は別の解析(第1表)に用いられることもあるので、リダイレクトしてファイルに落としておくとよい。

4. 2. bmbin

指定したカラービットマップファイルの画像を指定した輝度で二値化して、新しいビットマップファイルとして格納する bmbin は以下のようにして起動する。

```
bmbin srcfile {-} threshold dstfile
```

ここで、'srcfile' はオリジナルのカラービットマップの名前、'threshold' は二値化のしきい値（整数値）、'dstfile' は二値化して作成されるビットマップの名前である。しきい値に負の符号をつけるか否かでこのプログラムが行う二値化の処理は異なる。ある画素の輝度を b 、与えられたしきい値（符号も含めた）を c 、その絶対値を a とすると、

$c \geq 0$ の場合は、 $b > c$ なら 1、それ以外は 0

$c < 0$ の場合は、 $b \leq a$ なら 1、それ以外は 0

のように二値化が行われる。二値化終了後、標準出力に総画素数、二値化して 1 となった画素の数とその割合が出力される。

4. 3. exgrain

exgrain は以下のようにして走らせる。

```
exgrain srcfile dstfile {x1 y1 x2 y2}
```

ここで'srcfile'は二値ビットマップファイルの名前、'dstfile'は粒子像(クラスター)をサイズの順に色分けしたビットマップファイルの名前である。オプション指定の意味はbmhistの場合と同じで、解析領域の広がり指示する。

このプログラムは、求めた粒子像の形状（楕円近似）の情報を標準出力に書き出す。これをファイルにリダイレクトしておけば、dispedge（後述）で利用できる。なお、このファイルの詳細については後で説明する。

4. 4. exedge

exedge は以下のようにして起動する。

```
exedge file
```

上記の'file'はexgrainで得た粒子像をサイズ順に色分けしたビットマップファイルの名前である。

このプログラムで得られた粒子輪郭線情報は標準出力に書き込まれるので、リダイレクトしてファイルに落とす必要がある。このファイルのデータ構造については後で詳しく説明する。

4. 5. dispedge

dispedge は exedge で得た粒子像の輪郭線、exgrain で得た粒子形状を近似した楕円情報をグラフィック表示するためのプログラムで、以下のようにして走らせる。

```
dispedge edgefile {ellipsefile} {start last}
```

'edgefile' は exedge が標準出力に書き出したものをリダイレクトした粒子輪郭線情報の入ったファイルの名前、'ellipsefile'

は exgrain が標準出力に書き込んだデータをリダイレクトしたファイルの名前である。'start' と 'last' で表示したい粒子のラベル（最大サイズの粒子が 1 で、以下サイズ順に振られた番号）の最初と最後のものを指定する。これらを指定しなければすべての粒子について処理が行われる。

4. 6. eachedge

主として粒形解析のために、exedge で得た粒子輪郭線情報から指定した粒子の指定した輪郭線だけを抽出する場合は、eachedge を以下のようにして走らせる。

```
eachedge file {grain edge}
```

上記の'file'はexedgeで得た粒子輪郭線情報の入ったファイルの名前である。'grain'はサイズ順に1から振られた粒子の番号で、指定しなければ1を指定したことになる。'edge'は'grain'で指定した粒子の輪郭線の番号（外側の輪郭線が1、内側の輪郭線は2以上の数値）を指定する。指定しなければ1、すなわち外側の輪郭線を指定したことになる。

もとの輪郭線から抽出した輪郭線情報は標準出力に書き込まれるので、リダイレクトしてファイルに落とす必要がある。そのファイルの中味については後で詳しく説明する。なお、抽出した輪郭線の情報はグラフィック表示される。

4. 7. exprmt

主としてネットワーク解析のために、exedge で得た粒子輪郭線情報から指定した粒子の輪郭線を抽出する場合には、exprmt を以下のようにして走らせる。

```
exprmt file grain
```

上記の'file'はexedgeで得た粒子輪郭線情報の入ったファイルの名前で、'grain'はサイズ順に1から振られた粒子の番号である（省略不可）。eachedge と異なり、exprmt は指定した粒子のすべての輪郭線（外側の輪郭線と内側の輪郭線）を抽出する。

もとの輪郭線から抽出した輪郭線情報は標準出力に書き込まれるので、リダイレクトしてファイルに落とす必要がある。そのファイルの中味については後で詳しく説明する。なお、抽出した輪郭線の情報はグラフィック表示される。

5. データファイル

ここでは上で説明したプログラムが取り扱うデータファイルで、ビットマップファイル以外のものについて説明する。これらのうち、粒子像の輪郭線情報を書き込んだデータファイルは、本文で紹介した処理の後に続く粒形解析やネットワーク解析（第1表）で使用するため重要である。

5. 1. 粒形情報のファイル

exgrain は求めた粒形情報などのデータを以下のような順

番で標準出力に書き出す。

もとの画像の縦横の画素数 (x_w, y_w)

解析領域 (x_1, y_1, x_2, y_2)

解析領域の面積 (S)

粒子像の面積 (S_i)

粒子の総数 (N)

各粒子の情報 ($i = 1 \dots N$)

粒子の番号 (i)

面積 (S_i)

重心の座標 (x_i, y_i)

粒子と同じ面積の円の半径 (R_i)

粒子を近似した楕円の

偏平率 (O_i),

長軸半径 (L_i),

長軸方向 (θ_i)

標準出力に書き出されるデータは、リダイレクトしてファイルにセーブすることができる。これらのデータは一部重複していて、例えば、 $S = (x_2 - x_1 + 1) \times (y_2 - y_1 + 1)$ や $R_i = \sqrt{S_i / \pi}$ が成り立つ。また、もし exgrain 起動時に解析領域の指定がされていなければ、 $x_1 = 0, x_2 = x_w - 1, y_1 = 0, y_2 = y_w - 1$ となっているはずである。

各粒子についてのデータは粒子の個数 N 個分の記述がある。粒子の番号 i はその面積 S_i の順につけられている ($S_1 \geq S_2 \geq \dots$)。ここに書かれていない楕円の短軸半径 M_i は、 $M_i = S_i / (\pi \times L_i)$ として求めることができる。なお、楕円の偏平率は、 $O_i = 1 - M_i / L_i$ である。粒子が一画素だけ、もしくは一直線に並ぶ画素から成るために楕円で近似できなかつたり、また、楕円の長軸方向を決めることができなかつた場合には、出力される偏平率 O_i は 0 もしくは 1 にしてある。この O_i の値は定義どおり 0 なら粒子の形状が円、1 なら線分になるようにしてあるが、それに続く楕円の長軸半径 L_i とその方向 θ_i の値としては、それぞれ、 $O_i = 0$ のときは L_i は粒子と同じ面積の円の半径、 θ_i は意味のない数値が、 $O_i = 1$ のときは L_i は線分の長さの半分、 θ_i は線分に沿った方向が入るようにしてある。 θ_i の単位はラジアンで、 x 軸の方向が 0、 y 軸方向が $\pi / 2$ (90°) である。

5. 2. 輪郭線ファイル

画像中の粒子像の輪郭線情報のファイルは、exedge が標準出力に書き出したものをリダイレクトして作成したオリジナルのものと、そこから指定した粒子の輪郭線だけを抜きだしたものがある。後者には、ここでの処理に引き続く粒形解析とネットワーク解析のソフトウェアがそれぞれ扱い易いような二種類のものがある。これらの三種類の輪郭線情報のデータ構造を以下で説明する。

5. 2. 1. exedge の出力する輪郭線ファイル

exedge が標準出力に書き出す、すべての粒子の輪郭線データの構造は以下のようなものである。

もとの画像の縦横の画素数 (x_w, y_w)

粒子の総数 (N)

各粒子の輪郭線情報 ($i = 1 \dots N$)

粒子の番号 (i)

その粒子の輪郭線の総数 (P_i)

各輪郭線に属する辺の情報 ($j = 1 \dots P_i$)

輪郭線番号 (j)

辺ベクトルのチェインコードの

始点の座標 (x_j, y_j)

その輪郭線に含まれる

辺ベクトルの個数 (M_j)

辺ベクトルのチェインコード

($k = 1 \dots M_j$)

辺ベクトルの名前 (V_k)

上で段づけて表わしたようにこのデータ構造は入れ子になっていて、各粒子について複数の輪郭線（ただ一つの外側の輪郭線と複数あることもある内側の輪郭線）の情報がああり、また、各輪郭線はそれを構成する多数の向きを持つ辺要素（辺ベクトル）の情報からなる。

各粒子につけられた番号 i は exgrain の場合と同じで、解析している画像中に含まれる粒子のサイズの大きいものから順につけられたものである。その粒子の輪郭線の総数 N は、粒子の内部に穴がない場合には 1、ドーナツ状の粒子の場合には 2 以上の値になる（この値から 1 を減じたものはホール数と呼ばれている）。これらの複数の輪郭線は、外側の輪郭線なら 1、内側の輪郭線には 2 から始まる番号 j がつけられている（内側の輪郭線につけられた番号は exedge が輪郭線抽出を行う処理中に現われた順番に対応しているだけで、輪郭線の長さ等に対応しているわけではない）。すでに説明したように、これらの輪郭線の大局的な周回方向は外側の輪郭線が時計周り（座標系が通常のものとは逆であるため x 軸を y 軸に重ねるような回転方向）に、内側の輪郭線はその逆方向になっている。

各輪郭線は、画像を構成する画素のサイズを単位とする多数の辺からなる。本文で対象としている長方形画素の場合はこれらの辺に向きの異なる四種類があるので、それらを‘辺ベクトル’と呼んでいる。これらの辺ベクトルはそれぞれ 0, 1, 2, 3 の名前（番号）がつけられていて、それらを E_0, E_1, E_2, E_3 と表わす。これらの x, y 成分は以下のとおりである。

$$E_0 = (1, 0), E_1 = (0, 1),$$

$$E_2 = (-1, 0), E_3 = (0, -1)$$

一筆書きの折れ線を始点の座標とそれに続く多数の辺ベクトルの名前を表わす手法はチェーンコードと呼ばれていて、折れ線上の点の座標をすべて書き並べるよりもデータの量を減らすことができる（中野他，1988）。exedge の出力する輪郭線情報はチェーンコード化されていて、さきに示したexedge の出力するデータ構造で V_k と記されているものは上の辺ベクトル E_0, E_1, E_2, E_3 の名前 0 ～ 3 のいずれかである。

5. 2. 2. eachedge で抽出した輪郭線ファイル

exedge で作成した輪郭線データを粒形解析のソフトウェア（第 1 表）の入力に合わせるときには eachedge でそのデータの一部を抽出する。eachedge で抽出するデータは、さきに述べたexedge のデータのうち指定した粒子の指定した輪郭線（外側の輪郭線、内側の輪郭線）だけを取り出して、その上の点の個数と座標値そのものを書き並べた以下のようなものである。

輪郭線を構成する点の個数（ N ）

点の座標値（ $i = 1 \dots N$ ）

点の x 座標と y 座標（ x_i, y_i ）

上記の輪郭線を構成する点で、その始点（ x_1, y_1 ）と終点（ x_N, y_N ）は同じものである。そして、点の個数としてこれらをどちらも数えたものが書き込まれている。なお、さきにも述べたとおり、画像上での座標系は x 軸を右向き正とすると y 軸は下向きが正である。

5. 2. 3. exprmt で抽出した輪郭線ファイル

exedge で作成した輪郭線データをネットワーク解析のソフトウェア（第 1 表）の入力に合わせるときには exprmt でそのデータの一部を抽出する。exprmt で抽出するデータは、exedge のデータのうちの指定した粒子のすべての輪郭線（外側の輪郭線と内側の輪郭線）の情報で、以下ののような構造をしている。

もとの画像の縦横の画素数（ x_w, y_w ）

指定した粒子の輪郭線の本数（ P ）

各輪郭線に属する辺の情報（ $j = 1 \dots P$ ）

輪郭線を構成する点の個数（ N_j ）

点の座標値（ $i = 1 \dots N_j$ ）

点の x 座標と y 座標（ x_i, y_i ）

eachedge の場合と違って、上記の各輪郭線を構成する点の終点（ x_{N_j}, y_{N_j} ）は始点（ x_1, y_1 ）とは異なるものであり、これらを結んだ辺を考慮して初めて輪郭線は閉じたものになる。

6. 使用例

本文で紹介したソフトウェア群を用いた一連の画像処理

の例を第 2 図に示す。第 2（a）図に示したオリジナルの画像データは、新日本製鉄株式会社から提供していただいた鉄鉱石焼結体の X 線 CT（Computer Tomography）画像で、各画素に 0 ～ 150 の CT 値が輝度として入っている 150×150 画素からなるものである。このデータの詳細、またデータのもととなった実験条件については資料（新日本製鉄株式会社，1989）を参照して頂きたい。第 2（a）図で輝度値の小さい黒っぽい像は気孔、輝度値の大きい白い部分は焼結した石である。

第 2（b）図に示した（ビットマップ化した）画像データの輝度の頻度分布からも明らかなように、第 2（a）図の画像は輝度値 50 ～ 60 で二値化すれば画像中の黒い部分（気孔）を白い部分から分離できる。第 2（b）図のヒストグラムを表 1 の輝度頻度分布解析のソフトウェアで調べると、気孔の像を抽出するためには輝度値 58 を境に二値化（輝度 0 ～ 58 を 1 に、それ以外を輝度 0 に）すればよいことがわかった。bmbin でこの値（58）を指定して二値化した画像が第 2（c）図である。この二値化画像のビットマップファイルを exgrain にかけて、画像中に散在する気孔それぞれを色分けしたビットマップファイルを作成し、そして exedge でそれらの輪郭線を抽出した。こうして得たそれぞれの気孔の輪郭線を dispedge を使って描かせたものを第 2（d）図に示した。また、exgrain で得た各気孔の楕円近似の結果を輪郭線とともに dispedge で描かせたものが第 2（e）図である。ただし、第 2（e）図では気孔のうちサイズが大きいものから 10 個の分だけを表示させた。

7. おわりに

本文の説明でもわかるように、我々の画像処理システムに含まれるソフトウェア（第 1 表）は、それぞれが単機能であるかわりに入出力データやパラメータは少数で単純にしてある。また、プログラムの独立性も高いので、他の用途に用いたり、またそれぞれの処理を新たに作成したプログラムで置き換えて行なわせることが容易になっている。また、eachedge や exprmt などのデータ変換用のプログラムも各種作成してあって、もとなるデータがたとえばデジタイザで得られた画像データではないものであっても、それを処理することが可能になっている。

謝辞

鉄鉱石焼結体の X 線 CT（Computer Tomography）画像を提供していただいた新日本製鉄株式会社に感謝します。

(a)

(b)

(c)

(d)

(e)

第2図

(a) 鉄鉱石焼結体の X 線 CT 画像データ (新日本製鉄株式会社, 1989)。黒い部分は気孔で、白っぽいものが焼結した部分である。(b) (a) の画像の輝度の出現頻度分布。縦軸が輝度で、横軸がその輝度の画素数。輝度の値の小さいものは、(a) では黒い部分に対応する。(c) (a) の画像を輝度値 58 を境に二値化したもの。(d) (c) の二値画像から得た気孔 (黒い部分) の輪郭線。(e) (d) の輪郭線をもとに、気孔の形状を楕円近似したもの。楕円とともにその長軸、短軸も描いてある。

文 献

藤井 直之・中野 司（1988）フーリエディスクリプターとフラクタル次元－粒形解析のためのソフトウェア－，情報地質，No. 13， 119－139.

藤井 直之・中野 司（1989）画像処理のためのソフトウェア：（２）モジュール化ハードコピー・ルーチン，情報地質，No. 14A， 109－135.

中野 司・藤井直之・堀井洋一（1988）”GRAIN”と”EDGE”－粒子画像解析のためのソフトウェア－，情報地質，No. 13， 93－117.

中野 司・藤井 直之（1989）画像処理のためのソフトウェ

ア：（１）領域塗りつぶしと仮想スクリーン処理，情報地質，No. 14A， 93－107.

中野 司・藤井 直之（1991a）画像処理のためのソフトウェア：（４）画像データからのネットワーク構造の抽出，情報地質， 本号.

中野 司・藤井 直之（1991b）画像処理のためのソフトウェア：（５）画像中のネットワーク構造上の流れ解析，情報地質， 印刷中.

新日本製鉄株式会社（1989）粒度偏析制御式鉄鉱石焼結法の開発（詳細資料）， pp 65.

要 旨

画像処理のためのソフトウェア：（３）粒子像の識別と輪郭線の抽出： 中野 司・藤井 直之

我々の開発した画像処理システムを構成するソフトウェアについて説明する．‘exgrain’は二値化した画像データ中の粒子像（クラスタ）にそのサイズ順にラベリングを行い，画像をラベルに対応する輝度で色分けするためのプログラムである．また，プログラム‘exedge’はexgrainで得た各クラスタの輪郭線を抽出するために使用できる．得られた輪郭線データは複雑な構造をしているので，それを解析する際に必要になるデータ構造変換のプログラムも紹介する．

キーワード：画像処理， 粒形解析， ネットワーク解析， 輪郭線

（1990年8月3日受付，1991年3月15日受理）

付録 : プログラムリスト

bmhist.prj :

```

1: bmhist
2: bitmap
3: bmfile
4: glio
5: g2

```

bmhist.c :

```

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <alloc.h>
5: #include "bitmap.h"
6:
7: static void      Error(msg)
8: char             *msg;
9: {
10:      fputs(msg,stderr); fputc('\n',stderr); exit(1);
11: }
12:
13: #define H        (bitmap->hsize)
14: #define V        (bitmap->vsize)
15: #define D        (bitmap->dsize)
16:
17: #ifndef BSTEP
18: #define BSTEP    10
19: #endif
20:
21: void      main(argc, argv)
22: int       argc;
23: char     **argv;
24: {
25:      Bitmap  bITMAP,*bitmap=&bITMAP;
26:      long    *f,fmax;
27:      int     width,h1,h2,v1,v2,h,v,b,bmax;
28:      char    num[256];
29:
30:      if (argc!=2 && argc!=6)
31:          Error("usage : bmhist bmfile {x1 y1 x2 y2}");
32:
33:      if (!loadbitmap(bitmap,argv[1])) Error("file not found.");
34:
35:      if (argc==2) {
36:          h1=0; v1=0; h2=H; v2=V;
37:      }
38:      else {
39:          h1=atoi(argv[2]); h2=atoi(argv[4])+1;
40:          v1=atoi(argv[3]); v2=atoi(argv[5])+1;
41:          if (h1<0 || h1>h2 || h2>H ||
42:              v1<0 || v1>v2 || v2>V) Error("bad area is assigned.");
43:      }
44:
45:      width=1<<D;
46:      if ((f=(long *)calloc(width,sizeof(*f)))==NULL)
47:          Error("can not allocate memory for frequency.");
48:
49:      for (h=h1; h<h2; h++)
50:          for (v=v1; v<v2; v++)
51:              ++f[getbitmap(bitmap,h,v)];
52:
53:      bmax=0; fmax=0L;
54:      for (b=0; b<width; b++)
55:          if (f[b]!=0L) {
56:              bmax=b; if (f[b]>fmax) fmax=f[b];
57:          }
58:
59:      printf("%d\n",bmax+1);
60:
61:      initscr(); clrscr();
62:      window(-0.2*(double)fmax,-1.0,
63:             1.1*(double)fmax, 1.1*(double)(bmax+1),0);
64:      movep(0.0,-0.5);
65:      drawl((double)fmax,-0.5,1,-1);
66:      drawl((double)fmax,(double)bmax+0.5,1,-1);
67:      drawl(0.0,(double)bmax+0.5,1,-1);
68:      drawl(0.0,-0.5,1,-1);
69:

```

```

70:         for (b=BSTEP; b<=bmax; b+=BSTEP) {
71:             sprintf(num,"%4d",b);
72:             plots(-0.1 * (double)fmax, (double)b, num, 4);
73:             movep(-0.0125*(double)fmax, (double)b);
74:             drawl(0.0, (double)b, 1, -1);
75:             movep( 1.0125*(double)fmax, (double)b);
76:             drawl((double)fmax, (double)b, 1, -1);
77:         }
78:
79:         sprintf(num,"%ld",fmax);
80:         plots((double)fmax, 1.05*(double)(bmax+1), num, 4);
81:
82:         for (b=0; b<=bmax; b++) {
83:             movep(0.0, (double)b);
84:             drawl((double)f[b], (double)b, 6, -1);
85:             printf("%d\t%ld\n", b, f[b]);
86:         }
87: }

bmbin.prj :
1: bmbin
2: bitmap
3: bmfile

bmbin.c :
1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include "bitmap.h"
5:
6: static void    Error(msg)
7: char          *msg;
8: {
9:     fputs(msg, stderr); fputc('\n', stderr); exit(1);
10: }
11:
12: #define H      (oldbm->hsize)
13: #define V      (oldbm->vsize)
14:
15: void    main(argc, argv)
16: int     argc;
17: char    **argv;
18: {
19:     Bitmap  oLDBM, *oldbm=(#oLDBM), nEWBM, *newbm=(#nEWBM);
20:     long    t, p;
21:     int     c, s, h, v;
22:
23:     if (argc!=4) Error("usage : bmbin cbmfile {-}threshold bwbmfile");
24:
25:     if (!loadbitmap(oldbm, argv[1])) Error("file not found.");
26:     if (!mkbitmap(newbm, H, V, 1)) Error("can not make bitmap.");
27:     c=atoi(argv[2]); s=(c>0)?1:-1; t=(long)H*(long)V; p=0L;
28:     for (h=0; h<H; h++)
29:         for (v=0; v<V; v++)
30:             if (getbitmap(oldbm, h, v)*s>c) {
31:                 ++p; putbitmap(newbm, h, v, 1);
32:             }
33:     printf("%ld\t%ld\t%ld\n", t, p, (double)p/(double)t);
34:     if (!savebitmap(newbm, argv[3])) Error("can not create file.");
35: }

exgrain.prj :
1: exgrain
2: paint
3: bitmap
4: bmfile

exgrain.c :
1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <alloc.h>
5: #include <math.h>
6: #include "bitmap.h"
7:
8: #define PAINT_WORK_AREA_SIZE    0x1000
9:
10: #define Moment    struct MOMENT

```

```

11:
12: /* definition */ Moment {
13:     long    area,x,y;
14:     double  xx,yy,xy;
15: };
16:
17: #define Thread struct THREAD
18:
19: /* definition */ Thread {
20:     Moment *moment;
21:     union {
22:         Thread *link;
23:         int     name;
24:     } attribute;
25: };
26:
27: static Moment *moment;
28: static Thread **array;
29: static Bitmap srcbitmap,dstbitmap;
30: static int grain,xl,yd,xr,yu;
31:
32: #define AREA moment->area
33: #define X moment->x
34: #define Y moment->y
35: #define XX moment->xx
36: #define YY moment->yy
37: #define XY moment->xy
38:
39: static void ppf(x,y)
40: register int x,y;
41: {
42:     long xx,yy;
43:
44:     putbitmap(&srcbitmap,x,y,0);
45:     putbitmap(&dstbitmap,x-xl,y-yd,grain);
46:     ++(AREA);
47:     xx=(long)x; yy=(long)y;
48:     (X)+=xx; (Y)+=yy;
49:     (XX)+=(double)(xx*xx); (YY)+=(double)(yy*yy); (XY)+=(double)(xx*yy);
50: }
51:
52: static int gpf(x,y)
53: register int x,y;
54: {
55:     if (x<xl || y<yd || x>xr || y>yu)
56:         return(1);
57:     else
58:         return(!getbitmap(&srcbitmap,x,y));
59: }
60:
61: #define H (bitmap->hsize)
62: #define V (bitmap->vsize)
63: #define D (bitmap->dsize)
64: #define P (bitmap->plane)
65:
66: static int AppendBitmap(bitmap)
67: Bitmap *bitmap;
68: {
69:     int d,s;
70:     Byte **p;
71:
72:     if ((p=(Byte **)calloc(D+1,sizeof(*p)))==NULL) return(0);
73:     s=(int)((long)H*(long)V+7L)/8L;
74:     if ((p[D]=(Byte *)calloc(s,sizeof(**p)))==NULL) {
75:         free(p); return(0);
76:     }
77:     while (s!=0) p[D][--s]=(Byte)0;
78:     for (d=0; d<D; d++) p[d]=p[d];
79:     free(P); P=p; ++D; return(D);
80: }
81:
82: static long key;
83: static Thread *work;
84:
85: static void QuickSort(lower,upper)
86: int lower,upper;
87: {
88:     int l,u;
89:
90:     l=lower; u=upper; key=(array[(lower+upper)/2])->AREA;
91:     do {
92:         while ((array[l])->AREA>key) ++l;

```

```

93:         while (key>(array[u])>AREA) --u;
94:         if (l<=u) {
95:             work=array[l]; array[l]=array[u]; array[u]=work;
96:             ++l; --u;
97:         }
98:     } while (l<=u);
99:     if (lower<u) QuickSort(lower,u);
100:    if (l<upper) QuickSort(l,upper);
101: }
102:
103: static void    Error(msg)
104: char          *msg;
105: {
106:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
107: }
108:
109: #define LINK    attribute.link
110: #define NAME    attribute.name
111:
112: #define PI      3.141592653589793
113:
114: void main(argc,argv)
115: int  argc;
116: char **argv;
117: {
118:     int    limit,x,y,dx,dy,name,*index;
119:     long   total,painted;
120:     double r,xgc,ygc,a,b,bx,by,c,c1,c2,c3,c4,d,l,o;
121:     Thread *thread,*link;
122:     char   *usage="usage : exgrain srcfile dstfile {x1 y1 x2 y2}";
123:
124:     if (argc<3) Error(usage);
125:
126:     if (!loadbitmap(&srcbitmap,argv[1])) Error("file read error.");
127:
128:     printf("%d\t%d\n",srcbitmap.hsize,srcbitmap.vsize);
129:
130:     switch(argc) {
131:     case 3 :
132:         xl=0; yd=0; xr=srcbitmap.hsize-1; yu=srcbitmap.vsize-1;
133:         break;
134:     case 7 :
135:         xl=atoi(argv[3]); xr=atoi(argv[5]);
136:         yd=atoi(argv[4]); yu=atoi(argv[6]);
137:         if (xl<0 || yd<0 ||
138:             xr>srcbitmap.hsize || yu>srcbitmap.vsize ||
139:             xl>xr || yd>yu) Error("illegal area is assigned.");
140:         break;
141:     default :
142:         Error(usage);
143:     }
144:
145:     dx=xr-xl+1; dy=yu-yd+1; total=(long)dx*(long)dy;
146:     printf("%d\t%d\t%d\t%d\n",xl,yd,xr,yu);
147:     printf("%ld\n",total);
148:
149:     if (!mkbitmap(&dstbitmap,dx,dy,1))
150:         Error("can not make destination bitmap.");
151:
152:     limit=2;
153:
154:     grain=0; thread=NULL; painted=0L;
155:     for (y=yd; y<=yu; ++y)
156:         for (x=xl; x<=xr; x++)
157:             if (getbitmap(&srcbitmap,x,y)) {
158:                 ++grain;
159:                 if ((moment=(Moment *)calloc(1,sizeof(Moment)))==NULL ||
160:                     (link=(Thread *)calloc(1,sizeof(Thread)))==NULL)
161:                     Error("can not allocate parameter area.");
162:                 else {
163:                     link->moment=moment; link->LINK=thread;
164:                     thread=link;
165:                 }
166:                 if (grain==limit)
167:                     if (!AppendBitmap(&dstbitmap))
168:                         Error("can not append destination bitmap.");
169:                 else
170:                     limit<=1;
171:                 paintarea(x,y,PAINT_WORK_AREA_SIZE,gpf,ppf);
172:                 painted+=AREA;
173:             }
174:

```

```

175:         rmbitmap(&srcbitmap);
176:
177:         printf("%ld\n", painted);
178:         printf("%d\n", grain);
179:
180:         if (!grain) Error("no grain exists.");
181:
182:         if ((array=(Thread **)calloc(grain, sizeof(*array)))==NULL ||
183:             (index=(int *)calloc(grain, sizeof(*index)))==NULL)
184:             Error("can not allocate sorting area.");
185:
186:         for (name=grain-1; name>=0; name--) {
187:             link=thread->LINK;
188:             thread->NAME=name; array[name]=thread;
189:             thread=link;
190:         }
191:
192:         QuickSort(0, grain-1);
193:
194:         for (name=0; name<grain; name++) index[(array[name])->NAME]=name;
195:
196:         for (y=0; y<dy; y++)
197:             for (x=0; x<dx; x++)
198:                 if ((name=getbitmap(&dstbitmap, x, y))!=0)
199:                     putbitmap(&dstbitmap, x, y, index[name-1]+1);
200:
201:         if (!savebitmap(&dstbitmap, argv[2])) Error("file write error.");
202:
203:         for (name=0; name<grain; name++) {
204:             r=sqrt((double)((array[name])->AREA)/PI);
205:             xgc=(double)((array[name])->X)/(double)((array[name])->AREA);
206:             ygc=(double)((array[name])->Y)/(double)((array[name])->AREA);
207:             a=((array[name])->XY)-(double)((array[name])->AREA)*xgc*ygc;
208:             bx=((array[name])->XX)-(double)((array[name])->AREA)*xgc*xgc;
209:             by=((array[name])->YY)-(double)((array[name])->AREA)*ygc*ygc;
210:             if (a==0.0)
211:                 if (bx==by) {
212:                     o=0.0; l=r; d=0.0;
213:                 }
214:                 else
215:                     if (bx>by) {
216:                         o=1.0-sqrt(by/bx); d=0.0;
217:                         if (by!=0.0)
218:                             l=r*sqrt(sqrt(bx/by));
219:                         else
220:                             l=(double)((array[name])->AREA)/2.0;
221:                     }
222:                     else {
223:                         o=1.0-sqrt(bx/by); d=PI/2.0;
224:                         if (bx!=0.0)
225:                             l=r*sqrt(sqrt(by/bx));
226:                         else
227:                             l=(double)((array[name])->AREA)/2.0;
228:                     }
229:             else {
230:                 b=bx-by; c=sqrt(b*b+4.0*a*a);
231:                 c1=(-b+c)/(2.0*a); c2=(-b-c)/(2.0*a);
232:                 c3=(bx*c1*c1-2.0*a*c1+by)/(1.0+c1*c1);
233:                 c4=(bx*c2*c2-2.0*a*c2+by)/(1.0+c2*c2);
234:                 o=1.0-sqrt(c3/c4); l=r*sqrt(sqrt(c4/c3)); d=atan(c1);
235:             }
236:             printf("%d\t%ld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\n",
237:                 name+1, (array[name])->AREA, xgc, ygc, r, o, l, d);
238:         }
239: }

```

exedge.prj :

```

1: exedge
2: bitmap
3: bmfile

```

exedge.c :

```

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <alloc.h>
5: #include "bitmap.h"
6:
7: static struct {
8:     int     dx, dy;

```



```

9:         } edgevector[4]={1,0},{0,1},{-1,0},{0,-1}};
10:
11: static struct {
12:     int      name,dx,dy;
13:     } base[4]={0,0,0},{1,0,0},{0,-1,0},{1,0,-1}};
14:
15: static int      next[4][3]={1,0,3},{2,1,0},{3,2,1},{0,3,2}};
16:
17: static Bitmap    bitmap,edge[2];
18: static char      *vectorname;
19:
20: static void      Error(msg)
21: char             *msg;
22: {
23:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
24: }
25:
26: static int      Trace(xstart,ystart,dir)
27: int             xstart,ystart,dir;
28: {
29:     int          x,y,n,step,dirnew,namenew,xnew,ynew;
30:
31:     x=xstart; y=ystart; n=0;
32:     do {
33:         for (step=0; step<3; step++) {
34:             dirnew=next[dir][step];
35:             namenew=base[dirnew].name;
36:             xnew=x+base[dirnew].dx;
37:             ynew=y+base[dirnew].dy;
38:             if (getbitmap(&edge[namenew],xnew,ynew)) break;
39:         }
40:         if (step==3) Error("dead end.");
41:         putbitmap(&edge[namenew],xnew,ynew,0);
42:         dir=dirnew;
43:         vectorname[n++]=(char)dir;
44:         x+=edgevector[dir].dx; y+=edgevector[dir].dy;
45:     } while (x!=xstart || y!=ystart);
46:     return(n);
47: }
48:
49: #define Thread struct THREAD
50:
51: /* type */      Thread {
52:     int          vectorno,x,y;
53:     char         *vectorname;
54:     Thread       *thread;
55: };
56:
57: #define Grain struct GRAIN
58:
59: /* type */      Grain {
60:     char         *vectorname;
61:     int          perimeterno;
62:     Thread       *thread;
63: };
64:
65: void            main(argc,argv)
66: int             argc;
67: char           **argv;
68: {
69:     int          hsize,vsize,hsize1,vsize1,
70:     grainnomax,*edgeno,grainno,name,x,y,dir,perimeter,vector;
71:     Grain        *grain;
72:     Thread       *thread,*link;
73:
74:     if (argc<2) Error("usage : exedge bitmapfile");
75:
76:     if (!loadbitmap(&bitmap,argv[1])) Error("error in file read.");
77:
78:     hsize=bitmap.hsize; vsize=bitmap.vsize;
79:     printf("%d\t%d\n",hsize,vsize);
80:
81:     hsize1=hsize+1; vsize1=vsize+1; grainnomax=1<<bitmap.dsize;
82:     if (!mkbitmap(&edge[0],hsize,vsize1,1) ||
83:         !mkbitmap(&edge[1],hsize1,vsize,1) ||
84:         (edgeno=(int *)calloc(grainnomax,sizeof(*edgeno)))==NULL)
85:         Error("can not allocate edge buffer area.");
86:
87:     grainno=0;
88:     for (y=0; y<vsize; y++)
89:         for (x=0; x<hsize; x++)
90:             if ((name=getbitmap(&bitmap,x,y))!=0) {

```

```

91:             if (!getbitmap(&bitmap,x,y-1)) {
92:                 if (!((edgeno[name-1])++)) ++grainno;
93:                 putbitmap(&(edge[0]),x,y,1);
94:             }
95:             if (!getbitmap(&bitmap,x-1,y)) {
96:                 if (!((edgeno[name-1])++)) ++grainno;
97:                 putbitmap(&(edge[1]),x,y,1);
98:             }
99:         }
100:     else {
101:         if ((name=getbitmap(&bitmap,x,y-1))!=0) {
102:             if (!((edgeno[name-1])++)) ++grainno;
103:             putbitmap(&(edge[0]),x,y,1);
104:         }
105:         if ((name=getbitmap(&bitmap,x-1,y))!=0) {
106:             if (!((edgeno[name-1])++)) ++grainno;
107:             putbitmap(&(edge[1]),x,y,1);
108:         }
109:     }
110:
111:     printf("%d\n",grainno);
112:
113:     if ((grain=(Grain *)calloc(grainno,sizeof(*grain)))==NULL)
114:         Error("can not allocate grain buffer area.");
115:
116:     for (name=0; name<grainno; name++)
117:         if ((grain[name].vectorname=
118:             (char *)calloc(edgeno[name],sizeof(char)))==NULL)
119:             Error("can not allocate vector area.");
120:
121:     free(edgeno);
122:
123:     for (x=0; x<hsize; x++)
124:         for (y=0; y<vsize; y++)
125:             if (getbitmap(&(edge[0]),x,y)!=0) {
126:                 if ((thread=(Thread *)calloc(1,sizeof(Thread)))==NULL)
127:                     Error("can not allocate link area.");
128:                 if ((name=getbitmap(&bitmap,x,y))!=0) {
129:                     --name;
130:                     grain[name].thread=thread;
131:                     vectorname=grain[name].vectorname;
132:                     dir=3;
133:                 }
134:                 else {
135:                     name=getbitmap(&bitmap,x,y-1)-1;
136:                     for (link=grain[name].thread;
137:                         link->thread!=NULL; link=link->thread) ;
138:                     vectorname=(link->vectorname[link->vectorno]);
139:                     link->thread=thread;
140:                     dir=2;
141:                 }
142:                 thread->x=x;
143:                 thread->y=y;
144:                 thread->thread=NULL;
145:                 thread->vectorname=vectorname;
146:                 thread->vectorno=Trace(thread->x,thread->y,dir);
147:                 ++(grain[name].perimetro);
148:             }
149:
150:     for (name=0; name<grainno; name++) {
151:         printf("%d\t%d\n",name+1,grain[name].perimetro);
152:         for (perimeter=0, thread=grain[name].thread;
153:             perimeter<grain[name].perimetro;
154:             perimeter++, thread=thread->thread) {
155:             printf("%d\t%d\t%d\t%d\n",
156:                 perimeter+1,thread->x,thread->y,thread->vectorno);
157:             for (vector=0; vector<thread->vectorno; vector++)
158:                 printf("%d\n",thread->vectorname[vector]);
159:         }
160:     }
161: }

```

outline.h :

```

1:
2: typedef struct {
3:     struct {
4:         int    h,v;
5:     } area;
6:     int    gn;
7:     struct {
8:         int    en;

```

```

9:             struct {
10:                 struct {
11:                     int      x,y;
12:                 } start;
13:                 int      vn;
14:                 char      *vector;
15:             } *edge;
16:         } *grain;
17:     } Outline;
18:
19: static struct {
20:     int      dx,dy;
21:     } edgevector[4]={1,0},{0,1},{-1,0},{0,-1}};

```

outline.c :

```

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <alloc.h>
5: #include "outline.h"
6:
7: #define AH          (outline->area.h)
8: #define AV          (outline->area.v)
9: #define GN          (outline->gn)
10: #define G           outline->grain
11: #define EN(g)       (G[g].en)
12: #define E(g)        G[g].edge
13: #define X(g,e)      (E(g)[e].start.x)
14: #define Y(g,e)      (E(g)[e].start.y)
15: #define VN(g,e)     (E(g)[e].vn)
16: #define V(g,e)      E(g)[e].vector
17:
18: int      loadoutline(outline,name)
19: Outline *outline;
20: char     *name;
21: {
22:     int      g,e,v,i,j,vector;
23:     FILE     *file;
24:
25:     if ((file=fopen(name,"r"))==NULL) return(0);
26:
27:     fscanf(file,"%d %d",&AH,&AV);
28:     fscanf(file,"%d",&GN);
29:
30:     if ((G=(void *)calloc(GN,sizeof(*(G))))==NULL) return(0);
31:
32:     for (g=0; g<GN; g++) {
33:         fscanf(file,"%d %d",&i,&EN(g));
34:
35:         if (i!=g+1 ||
36:             (E(g)=(void *)calloc(EN(g),sizeof(*(E(g)))))==NULL)
37:             return(0);
38:         for (e=0; e<EN(g); e++) {
39:             fscanf(file,"%d %d %d %d",&j,&X(g,e),&Y(g,e),&VN(g,e));
40:             if (j!=e+1 ||
41:                 (V(g,e)=(char *)calloc(VN(g,e),sizeof(char)))==NULL)
42:                 return(0);
43:             for (v=0; v<VN(g,e); v++) {
44:                 fscanf(file,"%d",&vector); V(g,e)[v]=(char)vector;
45:             }
46:         }
47:     }
48:
49:     fclose(file); return(1);
50: }
51:
52: static void      drawedge(outline,grain,edge,color)
53: Outline          *outline;
54: int              grain,edge,color;
55: {
56:     int      x,y,v;
57:     char      *vector;
58:
59:     x=X(grain,edge); y=Y(grain,edge);
60:     movep((double)x,(double)y);
61:     for (vector=V(grain,edge), v=0; v<VN(grain,edge); v++, vector++) {
62:         x+=edgevector[(int)*vector].dx; y+=edgevector[(int)*vector].dy;
63:         drawl((double)x,(double)y,color,-1);
64:     }
65: }
66:

```

```

67: static void      drawgrain(outline,grain,edge,color)
68: Outline          *outline;
69: int              grain,edge,color;
70: {
71:     if (edge>=EN(grain)) return;
72:
73:     if (edge>=0)
74:         drawedge(outline,grain,edge,color);
75:     else {
76:         drawedge(outline,grain,0,color);
77:         for (edge=1; edge<EN(grain); edge++)
78:             drawedge(outline,grain,edge,color+8);
79:     }
80: }
81:
82: void      drawoutline(outline,grain,edge,color)
83: Outline *outline;
84: int     grain,edge,color;
85: {
86:     if (grain>=GN) return;
87:
88:     if (grain>=0)
89:         drawgrain(outline,grain,edge,color);
90:     else
91:         for (grain=0; grain<GN; grain++)
92:             drawgrain(outline,grain,edge,(grain%7)+1);
93: }

```

dispedge.prj :

```

1: dispedge
2: outline
3: glio
4: g2
5:

```

dispedge.c :

```

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include <alloc.h>
5: #include <math.h>
6: #include "outline.h"
7:
8: typedef struct {
9:     double x,y,r,o,l,d;
10: } Grain;
11:
12: typedef struct {
13:     struct {
14:         int      hsize,vsize;
15:     } original;
16:     struct {
17:         int      x1,y1,x2,y2;
18:     } analyzed;
19:     int      gn;
20:     Grain    *grain;
21: } Ellipse;
22:
23: #define GX      grain->x
24: #define GY      grain->y
25: #define GR      grain->r
26: #define GO      grain->o
27: #define GL      grain->l
28: #define GD      grain->d
29:
30: #define EO      ellipse.original
31: #define EOH     EO.hsize
32: #define EOV     EO.vsize
33: #define EA      ellipse.analyzed
34: #define EAX1    EA.x1
35: #define EAY1    EA.y1
36: #define EAX2    EA.x2
37: #define EAY2    EA.y2
38: #define EGN     ellipse.gn
39: #define EG(g)   ellipse.grain[g]
40:
41: #define OA      outline.area
42: #define OAH     OA.h
43: #define OAV     OA.v
44: #define OGN     outline.gn

```

```

45:
46: static int      LoadEllipse(ellipse,name)
47: Ellipse        *ellipse;
48: char           *name;
49: {
50:     int      g,i;
51:     long     s,s0,si;
52:     FILE     *file;
53:
54:     if ((file=fopen(name,"r"))==NULL) return(0);
55:
56:     fscanf(file,"%d %d",
57:             &(ellipse->original.hsize),&(ellipse->original.vsize));
58:     fscanf(file,"%d %d %d %d",
59:             &(ellipse->analyzed.x1),&(ellipse->analyzed.y1),
60:             &(ellipse->analyzed.x2),&(ellipse->analyzed.y2));
61:     fscanf(file,"%ld",&s);
62:     fscanf(file,"%ld",&s0);
63:     fscanf(file,"%d",&(ellipse->gn));
64:     if ((ellipse->grain=
65:          (Grain *)calloc(ellipse->gn,sizeof(Grain)))==NULL) return(0);
66:     for (g=0; g<ellipse->gn; g++) {
67:         fscanf(file,"%d %ld %lf %lf %lf %lf %lf %lf",
68:                 &i,&si,
69:                 &(ellipse->grain[g].x),&(ellipse->grain[g].y),
70:                 &(ellipse->grain[g].r),&(ellipse->grain[g].o),
71:                 &(ellipse->grain[g].l),&(ellipse->grain[g].d));
72:         if (i!=g+1) return(0);
73:     }
74:
75:     fclose(file); return(1);
76: }
77:
78: #define PI      3.141592653589793
79: #define DIVISION 100
80:
81: static void      DrawEllipse(grain,dx,dy,color)
82: Grain           *grain;
83: double          dx,dy;
84: int             color;
85: {
86:     double  da,a,x,y,sd,cd,sa,ca,p;
87:
88:     movep(GX-dx+0.5,GY+0.5);
89:     drawl(GX+dx+0.5,GY+0.5,color,-1);
90:     movep(GX+0.5,GY-dy+0.5);
91:     drawl(GX+0.5,GY+dy+0.5,color,-1);
92:
93:     cd=cos(GD); sd=sin(GD);
94:     if (GO==1.0) {
95:         x=GL*cd; y=GL*sd;
96:         movep(GX-x+0.5,GY-y+0.5); drawl(GX+x+0.5,GY+y+0.5,8,-1);
97:     }
98:     else {
99:         p=1.0-GO;
100:        da=2.0*PI/(double)DIVISION;
101:        movep(GX+GL*cd+0.5,GY+GL*sd+0.5);
102:        for (a=0.0; a<2.0*PI; a+=da) {
103:            ca=cos(a); sa=sin(a);
104:            x=GX+GL*(ca*cd-p*sa*sd); y=GY+GL*(ca*sd+p*sa*cd);
105:            drawl(x+0.5,y+0.5,8,-1);
106:        }
107:        drawl(GX+GL*cd+0.5,GY+GL*sd+0.5,8,-1);
108:        if (GO!=0.0) {
109:            movep(GX+GL*cd+0.5,GY+GL*sd+0.5);
110:            drawl(GX-GL*cd+0.5,GY-GL*sd+0.5,8,-1);
111:            movep(GX-GL*p*sd+0.5,GY+GL*p*cd+0.5);
112:            drawl(GX+GL*p*sd+0.5,GY-GL*p*cd+0.5,8,-1);
113:        }
114:    }
115: }
116:
117: static void      Error(msg)
118: char            *msg;
119: {
120:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
121: }
122:
123: void            main(argc,argv)
124: int             argc;
125: char            **argv;
126: {

```

```

127:      int      hsize,vsize,x1,y1,x2,y2,start,last,grain;
128:      double  dx,dy;
129:      Outline outline;
130:      Ellipse ellipse;
131:
132:      if (argc<2 || argc>5)
133:          Error(
134:              "usage : dispedge edgefile {ellipsefile} {startgrain lastgrain}"
135:          );
136:
137:      if (!loadoutline(&outline,argv[1]))
138:          Error("error in read edge file.");
139:
140:      if (argc==2 || argc==4) {
141:          hsize=0AH; vsize=0AV;
142:      }
143:      else {
144:          if (!LoadEllipse(&ellipse,argv[2]))
145:              Error("error in read ellipse file.");
146:          if (0AH!=EAX2-EAX1+1 || 0AV!=EAY2-EAY1+1 || 0GN!=EGN)
147:              Error("files are not matched.");
148:          hsize=EOH; vsize=EOV;
149:      }
150:
151:      if (argc==2 || argc==3) {
152:          start=0; last=0GN;
153:      }
154:      else {
155:          start=atoi(argv[argc-2])-1; start=(start<0)?0:start;
156:          last=atoi(argv[argc-1]); last=(last>0GN)?0GN:last;
157:      }
158:
159:      initscr(); clrscr();
160:      window(-2.0,-2.0,(double)hsize+2),(double)vsize+2,0);
161:      movep(-1.0,-1.0);
162:      drawl((double)hsize+1,-1.0,8,-1);
163:      drawl((double)hsize+1,(double)vsize+1,8,-1);
164:      drawl(-1.0,(double)vsize+1,8,-1);
165:      drawl(-1.0,-1.0,8,-1);
166:
167:      for (grain=start; grain<last; grain++)
168:          drawoutline(&outline,grain,-1,(grain%7)+1);
169:
170:      if (argc==3 || argc==5) {
171:          x1=EAX1; y1=EAY1; x2=EAX2; y2=EAY2;
172:          movep((double)(x1-1),(double)(y1-1));
173:          drawl((double)(x2+2),(double)(y1-1),8,-1);
174:          drawl((double)(x2+2),(double)(y2+2),8,-1);
175:          drawl((double)(x1-1),(double)(y2+2),8,-1);
176:          drawl((double)(x1-1),(double)(y1-1),8,-1);
177:
178:          dx=0.0075*(double)hsize; dy=0.0075*(double)vsize;
179:          for (grain=start; grain<last; grain++)
180:              DrawEllipse(&(EG(grain)),dx,dy,(grain%7)+1);
181:      }
182: }

```

eachedge.prj :

```

1: eachedge
2: outline
3: glio
4: g2

```

eachedge.c :

```

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include "outline.h"
5:
6: static void      Error(msg)
7:     char          *msg;
8: {
9:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
10: }
11:
12: #define H          outline.area.h
13: #define V          outline.area.v
14: #define OGE        outline.grain[g].edge[e]
15:
16: void      main(argc,argv)

```



```

17: int      argc;
18: char     **argv;
19: {
20:     int      x,y,v,n,g=0,e=0;
21:     Outline  outline;
22:     char     *usage="usage : eachedge edgefile {grainname edgename}";
23:
24:     if (argc==1 || argc==3 || argc>4) Error(usage);
25:     if (argc==4) {
26:         g=atoi(argv[2])-1; e=atoi(argv[3])-1;
27:     }
28:
29:     if (!loadoutline(&outline,argv[1])) Error("file not found.");
30:
31:     if (g<0 || g>=outline.gn || e<0 || e>=outline.grain[g].en)
32:         Error("bad grain or edge number.");
33:
34:     initscr(); clrscr();
35:     window(-2.0,-2.0,(double)(H+2),(double)(V+2),0);
36:     movep(-1.0,-1.0);
37:     drawl((double)(H+1),-1.0,1,-1);
38:     drawl((double)(H+1),(double)(V+1),1,-1);
39:     drawl(-1.0,(double)(V+1),1,-1);
40:     drawl(-1.0,-1.0,1,-1);
41:
42:     drawoutline(&outline,g,e,6);
43:
44:     printf("%d\n",OGE.vn+1);
45:     printf("%d\t%d\n",x=OGE.start.x,y=OGE.start.y);
46:     for (v=0; v<OGE.vn; v++) {
47:         n=(int)OGE.vector[v];
48:         printf("%d\t%d\n",x+=edgevector[n].dx,y+=edgevector[n].dy);
49:     }
50: }

exprmt.prj :

1: exprmt
2: outline
3: glio
4: g2

exprmt.c :

1:
2: #include <stdio.h>
3: #include <stdlib.h>
4: #include "outline.h"
5:
6: static void Error(msg)
7: char *msg;
8: {
9:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
10: }
11:
12: void main(argc,argv)
13: int argc;
14: char **argv;
15: {
16:     int      g,e,x,y,v;
17:     Outline  o;
18:
19:     if (argc!=3) Error("usage : exprmt edgefile grainname");
20:     if (!loadoutline(&o,argv[1])) Error("edge file not found.");
21:     if ((g=atoi(argv[2])-1)<0 || g>=o.gn) Error("bad grain name.");
22:
23:     initscr(); clrscr();
24:     window(-2.0,-2.0,(double)(o.area.h+2),(double)(o.area.v+2),0);
25:     movep(-1.0,-1.0);
26:     drawl((double)(o.area.h+1),-1.0,1,-1);
27:     drawl((double)(o.area.h+1),(double)(o.area.v+1),1,-1);
28:     drawl(-1.0,(double)(o.area.v+1),1,-1);
29:     drawl(-1.0,-1.0,1,-1);
30:     drawoutline(&o,g,-1,6);
31:
32:     printf("%d\t%d\n",o.area.h,o.area.v);
33:     printf("%d\n",o.grain[g].en);
34:     for (e=0; e<o.grain[g].en; e++) {
35:         printf("%d\n",o.grain[g].edge[e].vn);
36:         x=o.grain[g].edge[e].start.x;
37:         y=o.grain[g].edge[e].start.y;
38:         for (v=0; v<o.grain[g].edge[e].vn; v++) {

```

```
39:         printf("%d\t%d\n",x,y);
40:         x+=edgevector[(int)o.grain[g].edge[e].vector[v]].dx;
41:         y+=edgevector[(int)o.grain[g].edge[e].vector[v]].dy;
42:     }
43: }
44: }
```