イメージスキャナーと画像処理（その3）

河西秀夫*

Image Processing with Image Scanner (3)

Hideo KASAI*

0. はじめに

この資料は日本情報地質学会技術講習会（1993. 9. 28, 東京）で講演したものに修正・加筆したものである。この講習会では「安価な画像処理」をテーマにした。このため、講習会では市販の高価な画像処理ソフトを使用していない。但し、画像入力装置は必要なので、比較的低価格であるイメージスキャナーを中心に考えた。本講座はこれらの内容を以下のように3回に分けて行ってきた。

目次（第1回～第3回）

1. PC9801のグラフィックメモリについて（第1回）

- 1-1. グラフィックメモリの概要
- 1-2. アドレス（番地）の表記方法
- 1-3. GVRAMのデータの読み書き
- 1-4. メモリアクセスのアルゴリズム
- 1-5. 画像をディスクに記録する方法
- 1-6. ウィンドウ（窓開け）のアルゴリズム

2. イメージスキャナー（第2回）

- 2-1. イメージスキャナー
- 2-2. 2値化画像
- 2-3. ディザ法
- 2-4. 線分抽出

3. イメージスキャナーの使用例（第3回）

- 3-1. 波形解析
- 3-2. 等高線読み取り
- 3-3. 面積測定
- 3-4. 輪郭抽出
- 3-5. 輪郭追跡
- 3-6. 画像の合成
- 3-7. ディザ法で入力した画像の処理について

4. 結語

第1回目（河西, 1994a）では画像処理の基礎知識であるハードウェアの説明を中心に解説した。第2回目（河西, 1994b）ではイメージスキャナーを使用した画像処理に必要な基礎知識の説明を中心に解説した。そして、第3回目（最終回）の今回はイメージスキャナーの使用例について解説する。また、解説とともにプログラム作成の一助となるように必要なリストを掲載する。使用する言語はベーシック言語とC言語である。ベーシック言語はマイクロソフト系のN88DISK BASIC, 日本語ベーシック(86), QUICK BASIC, C言語はTurbo Cである。3種のベーシックの命令はほとんど同じである。また、処理の一部はN88DISK BASIC 及び日本語ベーシック(86)では速度の点で実用的でないものもある。使用した機種はNEC製のPC98シリーズである。

なお、以下の解説では、第1回目で述べたように2進数と16進数を使用している。ハードウェアに関する数値は2進数と16進数を使用の方が合理的であるからである。

また、第2回目で述べたように本文ではビット位置の数え方を情報学での数え方と逆にしてある。情報学では右から左に向かって、第0ビット、第1ビット、・・・、第7ビットと数えていく。PC9801シリーズのグラフィックメモリの場合、1つのアドレスに横方向に連続して並ぶ8ビット（8点）分の情報が格納されているので、この8ビットを1ビットずつ分解して配列に格納する必要がある。配列の添字の順序に合わせるためには、左側から第0ビット、第1ビット、・・・、第7ビットと数えていくのが便利である。本文中で示したプログラムではこの数え方にしている。

3. イメージスキャナーの使用例

イメージスキャナーを使用した例をいくつか紹介する。イメージスキャナーは基本的には画像を入力してCRTに表示する装置である。第1回で説明したように、入力画像はGVRAMにロードされる。第1回及び第2回で説明し

* 山梨学院大学 Yamanashi Gakuin University, Sakaori
2-4-4, Kofu, Yamanashi 400, Japan

たように、CRT 画面は GVRAM のモニターにすぎないので、GVRAM にアクセスすることにより画面に表示されている画像の情報を得ることができる。言い換えれば、CRT を入力装置として使用することが可能になる。

3-1. 波形解析

イメージスキャナーで読み込んだ波形を解析する。地震波形、常時微動波形等のチャート紙（記録用紙）に記録された既存のアナログ波形記録の解析を行うことが目的である（河西，1987）。

ここで説明するのは CRT 画面にデジタイザーと同じ役割をさせ、画面上の波形画像からデータを読み取っていく方法である。イメージスキャナーを使用して入力した画像は原画像よりも拡大されるので、詳細な読み取りが可能になる。第21図はペンレコーダーのチャートをイメージスキャナーで入力したものである。この図には振幅時間の基準線と振幅のゼロ位置が記入してある。この基準線は単位変換や0点補正などに使用される。

第21図 ペンレコーダーチャートの転送図形

処理順序は次のように行う。

①波形画像を2値化画像として読み込む。

②基準となる情報（時間、振幅）を読み取る。入力する波形画像に、あらかじめ時間や振幅の基準となる線（第22図④、⑤）を引いておく。読み取り間隔 dt は次の式で求まる。

$$dt = d / y$$

d は時間基準線の長さ（ドット数）、 y は時間基準線が示す時間間隔である。

③グラフィックカーソル（リスト1～3）で波形を構成する点の座標を順次読み取る。読み取りは等間隔で行っていく。この段階ではドット単位のデータ（スクリーン座標値）が読み取られる。この段階でのデータ列は次の式で表すことができる。

$$D_1 = \{(x_1, y_1), (x_2, y_2) \cdots (x_i, y_i) \cdots (x_n, y_n)\}$$

n はデータ数である。

④チャート上にあらかじめ記入してある振幅ゼロ線（第

第22図 波形画像読み取りの原理

●印は読み取り点、 Δt は読み取り間隔

22図)の y 座標値を読み取り、③で求めた y_i の値（振幅に相当する）をゼロ線を基準にした値に変換する。変換式は次のようになる。

$$y'_i = -(y_i - z)$$

z はゼロ線の y 座標値、 y'_i は変換後の値である。但し、 y'_i の値はまだドット単位の値である。例えば、 $y_i = 30$, $z = 100$ のとき $y'_i = 70$ となり、 $y_i = 180$, $z = 100$ のときは $y'_i = -80$ となる。スクリーン座標の原点は左上隅なので、変換には負号が必要である。

⑤単位変換を行い、ドット単位の値を mv, gal 等の値に変換する。変換式は次のとおりである。

$$y''_i = ys \times y'_i \div y_1$$

$$x''_i = xs \times x_i \div x_1$$

y''_i は mv, gal 等を単位とする振幅値、 x''_i は秒を単位とする値（経過時間）である。 ys は振幅基準線の長さ（ドット単位）、 xs は時間基準線の長さ（ドット単位）、 y_1 は振幅基準線の大きさ（mv, gal 等）、 x_1 は時間基準線の大きさ（秒）である。

この段階で測定値（mv, gal 等）と経過時間の2つの値からなるデータ列に変換される。

$$D_2 = \{(x''_1, y''_1), (x''_2, y''_2) \cdots (x''_i, y''_i) \cdots (x''_n, y''_n)\}$$

n はデータ数である。

⑥0点補正を行う。振幅の平均値が0になるように振幅値を次の式で補正する。

$$y^*_i = y''_i - y_m$$

y_m は振幅の平均値で次の式で求まる。

$$y_m = \left(\sum_{i=1}^n y''_i \right) / n$$

⑦変換後の座標点をシーケンシャルファイルに格納する。

⑧シーケンシャルファイルからデータを入力してFFT等の波形解析をおこなう。

第23図 転送画像から読み取った波形データの再現

(a) エルセントロ地震波の入力画像
(b) 再現画像

第23図 (a) はエルセントロ地震波 (インペリアル・バレイ地震, 1940; 大崎, 1976) の波形画像をイメージスキャナーで入力したものの, 同図 (b) は読み取った結果を再現したものである。時間軸の長さが両図で異なっているが, これはCRTに表示するときのx軸の表示間隔の設定の問題である。再現性はかなり良好である。

リスト1はN88DISKBASICのグラフィックカーソルルーチン, リスト2はTurbo Cのグラフィックカーソルルーチン, リスト3はリスト2で使用するキーコードのヘッダーである。

3-2. 等高線読み取り

イメージスキャナーで入力した地形図上の等高線を追跡して, 等高線を構成する各点の座標値をグラフィックカーソル (リスト1~3) で読み取る方法を説明する。各等高線を構成する各点の座標値が得られるとこの等高線の立体表示が可能になる (河西, 1988)。

等高線を構成する各点の座標値の読み取り方法は「3-1. 波形解析」と同様な方法で実現できる。

第24図はイメージスキャナーで読み込んだ地形図 (富士山) である。この図では枠内を処理領域として設定してある。図中の白丸は追跡点で黒丸は領域境界線上のデータである。グラフィックカーソルを使用してこれらの点の座標値 (スクリーン座標値) を順次読みとる。

第24図 イメージスキャナーで入力した地形図

A: 処理領域を設定するための指定点 B: 縮尺
黒丸印: 領域境界線上のデータ点
白丸印: 等高線上のデータ点

1本の等高線のデータ形式は次のようになる。

$$D_k = \{(x_1, y_1, h_1), (x_2, y_2, h_2), \dots, (x_n, y_n, h_n), (*, *, *)\}$$

(x_i, y_i, h_i) は等高線を構成する点 i の座標値で, x_i は横座標値, y_i は縦座標値, h_i は標高値である。等高線の場合 $h_1 = h_2 = \dots = h_n$ となる。このデータ形式を使用すると, 等高線以外の線 (例えば, 谷, 道路) も表現できる。 $(*, *, *)$ はデータの終端記号である。添字 k は等高線の番号である。閉曲線の場合, $x_i = x_n, y_i = y_n, h_i = h_n$ となる。グラフィックカーソルで読み取ることができるのはスクリーン座標値 (x_i, y_i) のみで, 標高データの inputs は別途工夫する必要がある。等高線の場合は標高値は全て同じなので, 各点のスクリーン座標値の読み取りが終了した時点で標高値をキー入力するか, 最初にキー入力すればよい。等高線のみの場合にはデータ形式は次のようにしてもよい。

$$D_k = \{(h, 0), (x_1, y_1), (x_2, y_2), \dots, (x_n, y_n), (*, *)\}$$

ここで, $(h, 0)$ の h は標高データ, 0 は形式を整えるためのダミーデータである。

第25図は第24図から読み取った等高線を立体表示した例である。この立体図では隠線処理を簡単にするために, 低標高の等高線から描いている。

第25図 立体地形図例 (富士山)

第24図の地形図から読み取った等高線を立体表示したもの

3-3. 面積測定

イメージスキャナーで入力した2値化画像では複雑な形状の面積測定が可能である。2値化画像はGVARM上で0と1からなる画像である。従って、0または1のどちらかの数を数えていけば面積が求まる。但し、測定できるのは画面上の0または1の画素の数であるので、実際の面積に換算する必要がある。換算は画面上の1ドットが実際の長さでどの程度になるかがわかれば良い。このために、入力画像上に長さの基準線を引いておき、これを読み取ればよい。

2値化画像はGVRAM 0 (青), GVRAM 1 (赤), GVRAM 2 (緑)のいずれかの画面に入力すれば、プログラムが簡単になり、また処理速度も向上する。例えば、GVRAM 0 (青)に2値化画像を入力した場合次のようにすればよい。

①セグメントベースをGVRAM 0の先頭アドレスに設定する。

②関数PEEKを使用してオフセットアドレスの内容を読む。

③オフセットアドレスの内容は8ビットなので、ビットごとに分解して配列に格納する。ビットごとに分解する方法は第1回で解説してある。例えば、オフセットアドレスの内容が&HFFの場合、分解結果は次のようになる。

BT (0)=BT (1)=・・・=BT (6)=BT (7)=1
また、オフセットアドレスの内容が&HF0の場合は次のようになる。

BT (0)=BT (1)=BT (2)=BT (3)=1

BT (4)=BT (5)=BT (6)=BT (7)=0

④1のビットまたは0のビットを数える。アドレスの内容が&HFFの場合の1のビットの数は8、&HF0の場合は4となる。

⑤対象領域について、②～④の処理を繰り返す。最終的な合計値が面積となる。GVARM01画面分のオフセットアドレスの範囲は&H0～&H7CFFである。関数PEEKとビットごとに分解する方法は第1回(河西, 1994a)で解説してある。

26図 面積測定用図形

である。リスト4ではまず青色の図形と赤色の枠線を描き(第26図)、その後この青色の図形の面積を測定する。青色の図形はGVRAM0(青画面)上に描かれた2値化画像に相当する。赤色の枠はGVRAM1上に描かれているので面積に加算されない。このリストではセグメントベースとオフセットアドレス(変数AD)を用いてGVRAM0をアクセスしている。対象とする領域はCRT画面全域であり、オフセットアドレスの範囲は0～&H7CFFになる。&Hは16進数を示す記号である。これらについては第1回(河西, 1994a)で解説してある。

第26図の青色の部分の面積は42670ドットで、PC9801RAを使用した場合の所要時間は2分30秒である。このリストをQUICK BASICで実行した場合の所要時間は1分(インタプリタ)及び32秒(コンパイル後)である。

Turbo CとQUICK BASICと場合、直接画素(x, y)の値を直接調べる関数getpixel(TurboC)とPOINT(QUICK BASIC)が用意されているので、これらの関数を使用すればプログラムは簡単になる。これらの関数の戻り値は色番号である。

3-4. 輪郭抽出

イメージスキャナーで入力した2値化画像の輪郭抽出を考える。第2回で説明したように線分の抽出は微分操作により行える。この方法は画素の濃度値の変化が急変する場所を図形の境界としている。2値化画像の場合は画素の値は0と1のどちらかなので図形境界は画素の値が0から1または1から0に変化する点である。ここではこの値の変化を直接調べて図形の境界(輪郭)を抽出する方法を解説する。この方法は微分操作を行なう方法に比べてプログラムが単純になる。以下の説明ではGVRAM0(青)に2値化画像を入力するものとする。

8ビットデータ

第27図 輪郭抽出の原理

図中の0と1は画素の値を示す

第27図に輪郭抽出の原理を示す。図中の0と1はGVRAMの各ビットの値を示す。GVRAMは8ビットメモリ

リスト4は面積測定

のプログラム(N88DISKBASIC)

なのでアドレスの内容は8桁の2進数（ビット列）となる。破線で囲まれた8桁の2進数がアドレスの内容である。これらのビットはCRT画面の画素に対応しているので、これらの0と1は画素の値でもある。0と1の境界が輪郭線になる。しかし、図中の0と1は画素の値なので実際に0と1の間に境界線を引くことは物理的に不可能である。このため、0と1のどちらかを輪郭点とする必要がある。ここでは1の側を輪郭点としてここに白色の点を打点することにする。図中のPは1を黒丸で0を白丸で表したものであり、点線で囲んだ部分に対応している。輪郭点は灰色の丸で示されている。

輪郭を抽出するにはX軸方向とY軸方向の2方向を調べる必要がある。X軸方向で輪郭を抽出する場合は次の式で変化点を調べることになる。

$$DN=N(X-1, Y)-N(X, Y)$$

$N(X-1, Y)$ は座標 $(X-1, Y)$ の画素の値、 $N(X, Y)$ は座標 (X, Y) の画素の値である。両者とも0または1のどちらかの値を取る。DN=1の場合は1から0に変化したことを示し、DN=-1のときは0から1に変化したことを示す。DN=1の場合は $(X-1, Y)$ に白色の点を打点し、DN=-1の場合は (X, Y) に打点する。

Y軸方向で輪郭を抽出する場合は次の式で変化点を調べることになる。

$$DN=N(X, Y-1)-N(X, Y)$$

同様に、 $N(X, Y-1)$ は座標 $(X, Y-1)$ の画素の値、 $N(X, Y)$ は座標 (X, Y) の画素の値である。DN=1の場合は1から0に変化したことを示し、DN=-1のときは0から1に変化したことを示す。DN=1の場合は $(X, Y-1)$ に白色の点を打点し、DN=-1の場合は (X, Y) に打点する。

輪郭を抽出後、青色の部分を消去し輪郭線のみを残すことになる。この場合、GVRAM0に直接0を書き込むことになる。その方法を示す(N88DISKBASIC)。

```
DEF SEG=&HA800
FOR AD%=0 TO &H7CFF . . . . 画面全領域
POKE AD%, 0 . . . . 0を書き込む
NEXT AD%
```

輪郭抽出のリストをリスト5（Turbo C）、リスト6（N88DISKBASIC）リスト7（Quick Basic）に示す。これらのリストではまず青色の図形と赤色の枠線を描き（第28図-1）、その後この青色の図形の輪郭を抽出する。前述の様に青色の図形はGVRAM0（青画面）上に描かれた2値化画像に相当する。第28図2は抽出結果である。PC9801RAを使用した場合の所要時間は5分程度である。リスト6をそのままQUICK BASICで実行した場合の所要時間は2分半程度（インタプリタ）及び1分半程度（コンパイ

第28図 輪郭抽出例

1は輪郭抽出を行う図形 2は抽出結果

ル後）、Turbo Cでは40秒程度である。

Turbo CとQUICK BASICと場合、前述の様に画素 (x, y) の色番号を直接調べる関数getpixel（Turbo C）とPOINT（QUICK BASIC）が用意されているので、この関数を使用すればほぼ原理通りにプログラム化が可能である。青画面以外を使用した2値化画像の場合は色番号が1以上の場合は画素の値を1とすればよい。しかし、QUICK BASICでは処理時間がかかりかかり、関数POINTを使用したプログラムの実行時間は8分30秒（インタプリタ）、5分（コンパイラ）で、前述のN88DISKBASICよりも処理時間が大きい。

N88DISKBASICと日本語ベーシック(86)の場合は画素 (x, y) の色番号を直接調べる関数がないので直接GVRAMをアクセスすることになり若干のアルゴリズム上の工夫が必要になる。X軸方向で輪郭を抽出する場合は次のような手順になる。

- ①関数PEEKであるオフセットアドレスの内容を読む。
- ②アドレスの内容は8ビットなので、ビットごとに分解して配列BTに格納する。
- ③ビットの変化を調べる。1から0に変化した点或いは0から1に変化した点を輪郭点とする。変化点は次の式で調べる。

$$DN=BT(I-1)-BT(I)$$

Iは右側から数えたビットの位置である。DNが1或いは-1の場合が変化点である。この場合変化点に白色の点を打

点するが、打点するスクリーン座標値 (DX, DY) は次の式でオフセットアドレスADから求める。

$$DX = (AD \text{ MOD } 80) * 8 + I$$

$$DY = \text{INT} (AD / 80)$$

INTは () 内の計算結果を整数化する BASIC 言語の関数、MODはAD÷80の剰余を求める BASIC 言語の演算子である。DN=1 の場合は (DX-1, DY) に、DN=-1 の場合は (DX, DY) に打点する。

最左位置のビットBT (0) は1つ前のアドレスの最右位置のビットFBT (7) と比較する必要がある。このため、変化を調べる式は次のようになる。

$$DN = \text{FBT} (7) - \text{BT} (0)$$

④対象領域について、①～③の処理を繰り返す。

Y 軸方向で輪郭を抽出する場合は次のような手順になる。

①関数 PEEK を使用して上下の位置にある2つのオフセットアドレスの内容を読む。例えば &H0 (X=0~7, Y=0) と &H50 (X=0~7, Y=1) の2つのオフセットアドレスをよむ (第1回, 第3図参照)。アドレスの内容をそれぞれ M1, M2 とする。

②次の論理計算を行う。

$$R = M1 \text{ OR } M2$$

OR は論理和を行うビット演算子である。M1 と M2 の両方が共に0ならば演算結果 R は0になる。この場合は両者の間には輪郭点が存在しないことを意味しているので①の操作に戻り、次の2つのオフセットアドレスを読む。R が0以外の場合は③に進む。

③アドレスの内容は8ビットなので、ビットごとに分解して配列 BT に格納する。例えば、&H0 の分解結果を配列 BT1 に格納し、&H50 の分解結果を BT2 に格納したとする。

④ビットの変化を調べる。1 から0に変化した点あるいは0 から1に変化した点を輪郭点とする。変化点は次の式で調べる。

$$DN = \text{BT} (I-1) - \text{BT} (I)$$

I は右側から数えたビットの位置である。DNが1あるいは-1 の場合が変化点である。この場合変化点に白色の点を打点するが、打点するスクリーン座標値 (DX, DY) は次の式でオフセットアドレス AD から求める。

$$DX = (AD \text{ MOD } 80) * 8 + I$$

$$DY = \text{INT} (AD / 80)$$

INT は () 内の計算結果を整数化する BASIC 言語の関数、MOD は AD÷80の剰余を求める BASIC 言語の演算子である。DN=1 の場合は (DX-1, DY) に、DN=-1 の場合は (DX, DY) に打点する。

最左位置のビットBT (0) は1つ前のアドレスの最右位置のビットFBT (7) と比較する必要がある。このため、変化を調べる式は次のようになる。

$$DN = \text{FBT} (7) - \text{BT} (0)$$

⑤対象領域について、①～④の処理を繰り返す。

3-5. 輪郭追跡

「3-4. 輪郭抽出」で抽出した輪郭を自動的に追跡し、輪郭線を構成する各点の座標値を記録する方法を考える。

光の三原色と輝度の重ね合わせにより CRT 画面の色は表現されている。「3-4. 輪郭抽出」では抽出した輪郭に白色の点を打点しているが、白色は青、赤、緑の三色を重ね合わせたものなのでこれらのどれかの画面を用いると2値化画像になる。前述の様に、2値化画像は画素の値が0と1の組み合わせで表現されているので、1の値の画素を追跡すればよいことになる。

追跡は次のように行う。

①輪郭線上の1点を指定する。この点の座標値を (Xs, Ys) とする。

②この点の周辺8近傍の画素の値を調べる。これらの画素のうちで1の値を示す画素の方向が追跡方向になる。但し、周辺8近傍の画素をどのような順序で調べるかをあらかじめ決めておく必要がある。時計まわりに追跡する場合と反時計まわりに追跡する場合では調べる順序が異なる。ここでは時計まわりの方向に追跡するものとする。

③画素の値が1である隣接画素に移動する。この点の座標を (X, Y) とする。新たにこの点を中心とする周辺8近傍の画素の値を調べて次の移動方向を決める。但し、周辺8近傍の画素を調べる順序は移動してきた方向を考慮する必要がある。このとき移動してきた方向の画素は調べない。このため調べる画素の数は7となる。第29図は移動方向と調べる順序の関係である。移動してきた方向は次式で調べる。

第29図 移動方向と周辺8近傍の調べる順序
矢印は移動方向 番号は調べる順序

$$xd = x - xf$$

$$yd = y - yf$$

(x, y) は移動後の座標、(xf, yf) は移動前の座標である。例えば、左から右へ移動してきた場合は $xd > 0$, $yd = 0$ となる。

④画素の値が1である隣接画素に再び移動する。この点の

座標を新たに (X, Y) とする。③の操作を行いつぎの移動点を捜し移動する。

⑤開曲線あるいは直線の場合、周辺7近傍の画素の値が全て0の時に追跡が終わる。閉曲線の場合は移動点の座標が追跡開始点の座標と同じになったときに追跡が終了する。

リスト8は輪郭追跡のプログラム (Quick Basic) で、第30図は追跡結果である。第30図では上段の図形の追跡をし下段にその追跡結果を再表示してある。このプログラムでは関数 POINT を使用している。

第30図 輪郭追跡例
上段の図は抽出を行う図形。下段は抽出結果の再現

3-6. 画像の合成

イメージスキャナーで読み込んだ2つの画像を合成する。画像の合成方法は次の順序で行う (第31図)。

第31図 画像合成の原理

- ①合成する2つの画像 (画像1, 画像2) を用意する。画像2を背景用とする。
- ②画像1をCRTに表示し、画像1から合成する部分を抜き出す。抜き出す領域を矩形とするとプログラムが簡単になる。抜き出す部分の情報を関数 PEEK で読み、ディ

スク上にファイルとして記録する。

③背景用の図形2をCRTに表示する。はめ込む位置を指定して、ディスク上に記録したファイルの内容を POKE でCRT (GVRAM) に書き込む。

④ はめ込んだ部分の境界を修正する。

第32図は画像の合成結果の例である。この合成図は富士山北麓の青木ヶ原溶岩流の流出直後の様子をイメージしたものである。背景は朝霧高原から撮影した富士山の写真をイメージスキャナーで入力 (ディザ法) したものである。溶岩流は富士山と同成分であるハワイのキラウエア火山の

第32図 画像合成例

もの (荒牧, 1983) であり、同様にディザ法で入力した。

リスト9は画像合成のプログラムである。このプログラムでは抜き出す領域をグラフィックカーソルを用いて指定する。

3-7. ディザ法で入力した画像の処理について

ディザ法で入力した写真等の画像 (以下、ディザ画像と称する) は濃淡で階調が表現されている。イメージスキャナーでは、ディザ画像作成処理はスキャナー内部の回路で処理され、結果だけが画面に表示される。このため、画像を構成する各画素の濃度値を直接出力することができない。このCRT画面に表示されたディザ画像に何らかの処理ができる方法を考える。

CRT上の画像の濃淡は第2回で解説したようにドットの密度の変化で表現されている。このドット密度は原画像の明るさに比例している。この関係を利用すれば、精度は落ちるものの、ある程度画像処理を行うことができる。

第33図 ディザ画像の処理
原画像の画素をまとめて新しい画素の単位を作る

具体的には次のようにする（第33図）。

①画像から4ドット×4ドットの領域（1ブロック）を取り出す。

②このブロックを新たな1画素として、ブロック内の点の数を数える。

③ブロック内の点の数を新たな画素（ブロック）の濃度値とする。1ブロック内には16個の原画像の画素があるので、新画素の濃度値は0～16の17段階になる。

④画像全域について上記の作業を繰り返す。

この方法で使用する画素の大きさは現画像の画素の4倍となり、精度は低下する。しかし、大まかな傾向を把握することは可能である。

具体的な方法は次のようになる。

①4行分のデータを関数PEEKでGVRAMから読む。

$$D_0 = \{M_{00}, M_{01}, M_{02}, M_{03}, \dots, M_{0n}\}$$

$$D_1 = \{M_{10}, M_{11}, M_{12}, M_{13}, \dots, M_{1n}\}$$

$$D_2 = \{M_{20}, M_{21}, M_{22}, M_{23}, \dots, M_{2n}\}$$

$$D_3 = \{M_{30}, M_{31}, M_{32}, M_{33}, \dots, M_{3n}\}$$

D_i は i 行目の1行分のデータ、 M_{ij} はGVRAMから読んだ8ビットデータである。添字 j はアドレス単位の順番を表す。CRT画面全領域を対象とする場合は $n=79$ である。X座標は640ドット分あるので $640/8=80$ となり、0から数えているので最大値は79になる。

②4ビットごとに分解

GVRAMからデータを読む場合、GVRAMは8ビット単位のメモリである点に注意する必要がある。関数PEEKで読むことのできるデータは横並び8ビット分である。このため、4ビットずつ分解する必要がある。分解は次のようにすればよい（N88DISKBASIC）。

$A = \text{PEEK}(AD) \dots\dots$ GVRAMから読む

$UA = (A \text{ AND } \&HF0) / 16 \dots\dots$ 上位4ビット分

$LA = A \text{ AND } \&H0F \dots\dots$ 下位4ビット分

AD はオフセットアドレス、 AND は論理積、 $\&H$ は16進数であることを表す記号である。 $(A \text{ AND } \&HF0)$ は A の下位4ビットを0にする計算である。例えば、 $(\&HDE \text{ AND } \&HF0)$ の結果は $\&HD0$ となる。この値を16で割ると4ビット右にシフトされ $\&HD$ となる。

4行分のデータは次のようになる。

$$D_0 = \{M_{00U}, M_{00L}, M_{01U}, M_{01L}, \dots, M_{0nU}, M_{0nL}\}$$

$$D_1 = \{M_{10U}, M_{10L}, M_{11U}, M_{11L}, \dots, M_{1nU}, M_{1nL}\}$$

$$D_2 = \{M_{20U}, M_{20L}, M_{21U}, M_{21L}, \dots, M_{2nU}, M_{2nL}\}$$

$$D_3 = \{M_{30U}, M_{30L}, M_{31U}, M_{31L}, \dots, M_{3nU}, M_{3nL}\}$$

M_{iJU} は M_{iJ} の上位4ビット、 M_{iJL} は M_{iJ} の下位4ビットを表す。

③4ビットのデータを1ビットずつ分解して1の数を数える。

1の数を合計するプログラムは次のようになる（N88DISKBASIC）。

FOR I=0 TO 3 $\dots\dots$ 4回ループ

BIT=M AND &H1 $\dots\dots$ 右端のビットを取り出す

IF BIT=1 THEN K=K+1

M=INT(M/2) $\dots\dots$ 1ビット右へシフト

NEXT I

例えば、 $\&HF$ の場合1の数は4（ $\&HF=1111$ ）、 $\&HE$ の場合の1の数は3（ $\&HE=1110$ ）、 $\&H3$ の場合の1の数は2（ $\&H3=0011$ ）となる。

4行分のデータは次のようになる。

$$D_0 = \{C_{00U}, C_{00L}, C_{01U}, C_{01L}, \dots, C_{0nU}, C_{0nL}\}$$

$$D_1 = \{C_{10U}, C_{10L}, C_{11U}, C_{11L}, \dots, C_{1nU}, C_{1nL}\}$$

$$D_2 = \{C_{20U}, C_{20L}, C_{21U}, C_{21L}, \dots, C_{2nU}, C_{2nL}\}$$

$$D_3 = \{C_{30U}, C_{30L}, C_{31U}, C_{31L}, \dots, C_{3nU}, C_{3nL}\}$$

C_{iJU} は M_{iJU} の1の数、 C_{iJL} は M_{iJU} の1の数を表す。

④4ドット×4ドットの領域（1ブロック）内の点の数を数える。このブロックを単位としたデータは次のようになる。

$$N0 = \{B0, B1, \dots, Bn\}$$

$B_0 = C_{00U} + C_{10U} + C_{20U} + C_{30U}$, $B_1 = C_{00L} + C_{10L} + C_{20L} + C_{30L}$ である。このデータが新しい画素値になる。

第34図は第31図のディザ画像を処理し、求められた新画素の値を微分操作して線分を抽出した例である。ほぼ輪郭が抽出できている。この他、新画素の値に応じたカラー表示を行うことも可能である。

第34図 新画素を使用した線分抽出結果

示を行うことも可能である。

4. 結語

3回にわたってイメージスキャナーを使用した画像処理について解説した。第1回で解説したようにパソコンでは画素の値をそのまま配列に格納すると必要なメモリが大きくなりプログラムの実行が困難になることもある。例えば1画面全部の画素を配列に格納した場合 $640 \times 400 = 256,000$ バイト $= 256$ キロバイト（1画素の大きさを1バイト $= 8$ ビットとする）の大きさになる。この大きさはパソコンではかなりの大きさになり、プログラム中で使用する他の変

数の容量を考慮するとメモリーオーバーになる可能性が高い。配列を使用する場合は処理する画像の大きさが小さくなり、実用面から考えると不便である。このため GVRAM に直接アクセスする必要がある。

Turbo C 及び Quick Basic では画素の値を直接調べる関数が準備されている。このため、これらの言語を使用すると見掛け上は GVRAM にアクセスする必要はない。このため、これらの関数を通して GVRAM にアクセスすることになる。N88DISKBASIC と日本語ベーシック（86）ではこの関数が存在しないので、GVRAM を直接アクセスする必要がある。また、Quick Basic の画素の値を調べる関数の処理速度はかなり遅く、実用的な処理速度ではない。むしろ、N88DISKBASIC と日本語ベーシック（86）と同様に直接 GVRAM に直接アクセスした方が処理時間は速い。

処理速度や機能から考えると、ベーシック言語（N88 DISKBASIC, 日本語ベーシック（86）, Quick Basic）は画像処理に不向きな言語といえる。画像処理を行う場合は C 言語を使用した方がよいと思われる。しかし、処理時間に考慮しないならばアルゴリズムを工夫することによりベーシック言語でもかなりの画像処理は行えるものと思われる。第3回の今回説明した使用例は基本的なものであり、実用化するにはさらに工夫を要するものもある。

文 献

- 荒巻重夫（1983）火山の驚異，福武書店，23
- 河西秀夫（1987）イメージスキャナーを利用したアナログ波形画像データの処理方法，情報地質，no.12，31-41.
- 河西秀夫（1988）等高線方式による立体地形図，情報地質，no.13，17-41.
- 河西秀夫（1994a）イメージスキャナーと画像処理（その1），情報地質，5，21-38.
- 河西秀夫（1994b）イメージスキャナーと画像処理（その2），情報地質，5，101-109.
- 大崎順彦（1976）地震動のスペクトル入門，鹿島出版会，3

リスト1 グラフィックカーソルルーチン (N88DISKBASIC)

```

610 *CURSOL: '(CX,CY)
620 'カーソル移動
630 LINE(0,10)-(20,10),4:LINE(10,0)-(10,20),4
640 BYTE=((20+7)¥8)*20*3+4:FACT=BYTE¥2+1
650 DIM CS%(FACT)
660 GET@(6,6)-(14,14),CS%
670 LINE(0,0)-(20,20),0,BF
680 '
690 CX=CSX:CY=CSY
700 PUT@(CX,CY),CS%,XOR
710 CH=ASC(INPUT$(1)):CCX=CX:CCY=CY
720 IF CH=13 THEN GOTO 810
730 IF CH=28 THEN CX=CX+1
740 IF CH=29 THEN CX=CX-1
750 IF CH=30 THEN CY=CY-1
760 IF CH=31 THEN CY=CY+1
770 GOSUB *LIMIT
780 PUT@(CCX,CCY),CS%,XOR
790 PUT@(CX,CY),CS%
800 GOTO 710
810 PUT@(CX,CY),CS%,XOR
820 CX=CX+4 :CY=CY+4
830 ERASE CS%
840 RETURN
850 '
860 *LIMIT: '(CX,CY)
870 'カーソル移動限界
880 IF CX<0 THEN CX=599
890 IF CX>600 THEN CX=0
900 IF CY<0 THEN CY=399
910 IF CY>399 THEN CY=0
920 RETURN

```

リスト2 グラフィックカーソルルーチン (Turbo C)

```

/*****/
/** グラフィック カーソル ****/
/** 分割コンパイル用 ****/
/** gcursol.c ****/
/** [list] ****/
/** void gcursol_make(buf) **/
/** int gcursol_mov(buf,px,py) **/
/** void gcursol_area(buf,area,fg) */
/*****/
#include "graphics.h"
#include "conio.h"
#include "stdio.h"
#include "keycode.h"
/*****/
/** グラフィックカーソル作成 **/
/*****/
void gcursol_make(buf)
char buf[];/**バッファ**/
{
    int x1,x2;
    int y1,y2;
    x1=580;y1=56;
    x2=x1+20;y2=y1+20;
    /**十字を描く**/
    setcolor(WHITE);
    line(x1,y1,x2,y2);
    line(x1,y1,x1+10,y1);
    line(x1,y1,x1,y1+10);
    /**バッファにカーソルを読む**/
    getimage(x1,y1,x2,y2,buf);
    setfillstyle(SOLID_FILL,BLACK);
}

```

```

        bar(x1,y1,x2,y2);
    }
/*****
** グラフィックカーソル移動 **
*****/
int gcursor_mov(buf,px,py)
char buf[];
int px[],py[]; /*select point*/
{
    int ky;      /*キーコード*/
    int cx,cy;   /*カーソル座標*/
    int dx,dy;   /*カーソル移動幅*/
    int fcx,fcy;
    int xx,yy;   /*求める点の座標*/
    int efg;     /*終了フラグ*/
    int pn;      /*select point number*/
    int lx,ly;
    /**初期設定**/
    cx=300;cy=100;
    dx=dy=1;
    pn=0;
    lx=70;ly=8;
    putimage(cx,cy,buf,XOR_PUT);
    efg=0;
    /**カーソル移動**/
    while(efg==0)
    {
        ky=getch();
        fcx=cx;fcy=cy;
        switch(ky)
        {
            case DOWN:
                cy=cy+dy;
                if(cy>390) {cy=390;}
                break;
            case UP:
                cy=cy-dy;
                if(cy<0) {cy=0;}
                break;
            case RIGHT:
                cx=cx+dx;
                if(cx>500) {cx=500;}
                break;
            case LEFT:
                cx=cx-dx;
                if(cx<0) {cx=0;}
                break;
            case RET:
                xx=cx;yy=cy;
                px[pn]=xx;py[pn]=yy;
                pn++;
                gotoxy(lx,ly);printf("%3d¥n",xx);
                gotoxy(lx+4,ly);printf("%3d¥n",yy);
                ly++;
                circle(xx,yy,5);
                break;
            case ESC:
                efg=1;break;
            case SPACE:
                if(dx==10 && dy==10) {dx=1;dy=1;}
                else
                    {dx=10;dy=10;}
                break;
        }
        putimage(fcx,fcy,buf,XOR_PUT);
        putimage(cx,cy,buf,XOR_PUT);
    }
    return(pn);
}
/*****/

```

```

/** グラフィックカーソル移動 **/
/** 調査領域設定 **/
/*****/
void gcursol_area(buf,area,fg)
char buf[];
int area[]; /*指定領域*/
int fg;
{
    int ky; /*キーコード*/
    int cx,cy; /*カーソル座標*/
    int dx,dy; /*カーソル移動幅*/
    int fcx,fcy;
    int xx,yy; /*求める点の座標*/
    int efg; /*終了フラグ*/
    int pn; /*select point number*/
    int px[2],py[2];
    /**初期設定**/
    cx=300;cy=100;
    dx=dy=1;
    pn=0;
    putimage(cx,cy,buf,XOR_PUT);
    efg=0;
    /**カーソル移動**/
    while(efg==0)
    {
        ky=getch();
        fcx=cx;fcy=cy;
        switch(ky)
        {
            case DOWN:
                cy=cy+dy;
                if(cy>390) {cy=390;}
                break;
            case UP:
                cy=cy-dy;
                if(cy<0) {cy=0;}
                break;
            case RIGHT:
                cx=cx+dx;
                if(cx>500) {cx=500;}
                break;
            case LEFT:
                cx=cx-dx;
                if(cx<0) {cx=0;}
                break;
            case RET:
                xx=cx;yy=cy;
                px[pn]=xx;py[pn]=yy;
                if(pn>=1)
                {
                    if (fg==0)
                        {rectangle(px[0],py[0],px[1],py[
1]);}
                    efg=1;
                    break;
                }
            else
            {
                pn++;
                if (fg==0)
                    {circle(xx,yy,5);}
                break;
            }
            case ESC:
                px[0]=py[0]=px[1]=py[1]=-1;
                efg=1;break;
            case SPACE:
                if(dx==10 && dy==10) {dx=1;dy=1;}
                else
                    {dx=10;dy=10;}

```

```

        break;
    }
    putimage(fcx,fcy,buf,XOR_PUT);
    putimage(cx,cy,buf,XOR_PUT);
}
putimage(cx,cy,buf,XOR_PUT);
area[0]=px[0];area[1]=py[0];
area[2]=px[1];area[3]=py[1];
}

```

リスト3 リスト2で使用するキーコードヘッダー

```

/*****
/** キーコードヘッダー ****
*****/
#define DOWN 0x0a
#define UP 0x0b
#define RIGHT 0x0c
#define LEFT 0x08
#define RET 0x0d
#define PLUS 43
#define MINUS 45
#define SPACE 0x20
#define ESC 0x1b
#define DEL 0x44
#define INS 0x50

```

リスト4 面積測定プログラム (N88DISKBASIC)

```

10 '*****
20 '** 面積測定プログラム **
30 '*****
40 SCREEN 3,0
50 CONSOLE 0,24,0,1
60 CLS 3
70 DIM BT(8)
80 '図形を描く
90 LINE(0,0)-(639,399),2,B
100 CIRCLE(200,200),170,1,...3
110 PAINT(200,200),1,1
120 CIRCLE(340,220),100,1,...1.5
130 PAINT(340,300),1,1
140 '面積を求める
150 SUM=0
160 DEF SEG=&HA800 '青画面
170 FOR AD=0 TO &H7CFF
180 M=PEEK(AD) 'メモリを読む
190 IF M<>0 THEN GOSUB *BIT.SUM
200 NEXT
201 PRINT "面積 = ";SUM;"dot"
210 END
220 '
230 '*****
240 '*** ビットの合計 ***
250 '*****
260 *BIT.SUM
270 'ビットの分解
280 DT=M
290 FOR J=7 TO 0 STEP -1
300 BT(J)=DT AND &H1
310 DT=INT(DT/2)
320 NEXT
330 'ビット1を数える
340 BSUM=0
350 FOR J=0 TO 7

```

```

360 IF BT(J)=1 THEN BSUM=BSUM+1
370 NEXT
371 SUM=SUM+BSUM
380 RETURN

```

リスト5 輪郭抽出プログラム (Turbo C)

```

/*****
***      輪 郭 抽 出      ***
*****/
#include "dos.h"
#include "graphics.h"
#include "time.h"
/**関数宣言**/
void blue_clear(void);
void gsample(void);
void main(void)
{
    int x,y;
    int j;
    int cn1,cn2;          /*色 番 号 */
    int bt1,bt2;
    int dn;               /*差 */
    int gerr;
    long st,et;           /*time*/
    int gdriver=DETECT,gmode=PC98C16;
    /*グラフィック画面初期化*/
    textmode(0);
    initgraph(&gdriver,&gmode,"");
    if((gerr=graphresult())!=grOk)
    {
        printf("error:%s\n",grapherrormsg(gerr));
        exit(1);
    }
    for(j=0;j<16;j++) {setpalette(j,j);}
    gsample();           /*作業用図形*/
    /**水平方向輪郭抽出**/
    time(&st);           /*time strat*/
    for(y=0;y<399;y++)
    {
        cn1=getpixel(0,y);
        if(cn1==1) {bt1=1;}
        else      {bt1=0;}
        for(x=1;x<639;x++)
        {
            cn2=getpixel(x,y);
            if(cn2==1) {bt2=1;}
            else      {bt2=0;}
            dn=bt1-bt2;
            switch (dn) {
                case 1:
                    putpixel(x,y,WHITE);
                    break;
                case -1:
                    putpixel(x-1,y,WHITE);
            }
            bt1=bt2;
        }
    }
    /**垂直方向輪郭抽出**/
    for(x=0;x<639;x++)
    {
        cn1=getpixel(0,y);
        if(cn1==1) {bt1=1;}
        else      {bt1=0;}
        for(y=1;y<399;y++)
        {

```

```

        cn2=getpixel(x,y);
        if(cn2==1) {bt2=1;}
        else      {bt2=0;}
        dn=bt1-bt2;
        switch (dn) {
            case 1:
                putpixel(x,y-1,WHITE);
                break;
            case -1:
                putpixel(x,y,WHITE);
        }
        bt1=bt2;
    }
    time(&et);
blue_clear();
printf("time=%fn ㉞n",difftime(et,st));
}

void blue_clear(void)
{
    int ad;    /*offset address*/
    for(ad=0;ad<=0x7cff;ad=ad+2)
    {
        poke(0xa800,ad,0);
    }
}

void gsample()
{
    setcolor(RED);
    rectangle(0,0,639,399);    /*枠線*/
    setcolor(BLUE);
    setfillstyle(SOLID_FILL,1);
    fillellipse(200,200,170,51);
    fillellipse(340,200,50,100);
    circle(600,200,30);
    floodfill(600,200,1);
    setcolor(BLACK);
    circle(200,200,20);
    setfillstyle(SOLID_FILL,0);floodfill(200,200,0);
}

```

リスト6 輪郭抽出プログラム (N88DISKBASIC)

```

10 '*****
20 '** 輪郭抽出プログラム **
30 '*****
40 SCREEN 3,0
50 CONSOLE 0,24,0,1
60 CLS 3
70 DIM BT(8),BT1(8),BT2(8)
80 '図形を描く
90 LINE(0,0)-(639,399),2,B
100 CIRCLE(200,200),170,1,...3
110 PAINT(200,200),1,1
120 CIRCLE(340,220),100,1,...1.5
130 PAINT(340,300),1,1
131 CIRCLE(200,200),20,0
132 PAINT(200,200),0,0
133 CIRCLE(600,200),30,1
134 PAINT(600,200),1,1
140 '輪郭を求める
150 PRINT TIME$
160 SUM=0
170 DEF SEG=&HA800 '青画面
180 GOSUB *OUTLINEH '輪郭抽出水平方向
190 GOSUB *OUTLINEV '輪郭抽出垂直方向
191 GOSUB *BCLEAR
195 PRINT TIME$

```

```

200 END
210 '
220 '*****
230 '*** 輪郭抽出作業 *****
240 '*****
250 *OUTLINEH '水平方向
260 FOR AD%=0 TO &H7CFF
270   M%=PEEK(AD%) 'メモリを読む
275   IF RB=1 AND M%=0 THEN GOSUB *BIT.CHECK:RB=BIT[7]
280   IF M%<>0 THEN GOSUB *BIT.CHECK:RB=BT[7] ELSE RB=0
286   FM%=M%
290 NEXT
300 RETURN
310 '
320 *OUTLINEV '垂直方向
330 FOR K=0 TO &H4F
340   M1%=PEEK(K) 'メモリを読む(0行目)
350   FOR L=80 TO 399*80 STEP 80
360     AD%=K+L
370     M2%=PEEK(AD%) 'メモリを読む
380     R%=M1% OR M2%
390     IF R%<>0 THEN GOSUB *BIT.CHECKV
395     M1%=M2%
400   NEXT L
410 NEXT K
420 RETURN
430 '
440 '*****
450 '***ビットの分解. 輪郭抽出***
460 '*****
470 *BIT.CHECK
480 'ビット分解
490 DT%=M%
500 FOR J=7 TO 0 STEP -1
510   BT(J)=DT% AND &H1
520   DT%=INT(DT%/2)
530 NEXT
540 '変化点を見つける
550 NC1=RB '1つ前のアドレスの最右ビット
560 FOR I=0 TO 7
570   NC2=BT(I)
580   DN=NC1-NC2
590   IF DN<>0 THEN GOSUB *WHITE.DOTH
600   NC1=NC2
610 NEXT
620 RETURN
630 '
640 *BIT.CHECKV
650 'ビット分解
660 DT%=M1%
670 FOR J=7 TO 0 STEP -1
680   BT1(J)=DT% AND &H1
690   DT%=INT(DT%/2)
700 NEXT
710 DT%=M2%
720 FOR J=7 TO 0 STEP -1
730   BT2(J)=DT% AND &H1
740   DT%=INT(DT%/2)
750 NEXT
760 FOR I=0 TO 7
770   DN=BT1(I)-BT2(I)
771   IF DN<>0 THEN GOSUB *WHITE.DOTV
772 NEXT I
780 RETURN
790 '*****
800 '***変化点に白点を打つ***
810 '*****
820 'オフセットアドレスからスクリーン座標値を求める
830 *WHITE.DOTH
840 DY=INT(AD%/80)

```



```

850 DX=AD% MOD 80
860 DX=DX*8+I
870 IF DN=1 THEN PSET(DX-1,DY),7 ELSE PSET(DX,DY),7
880 RETURN
890 '
900 *WHITE.DOTV
910 DY=L/80          'Y座 標 値
920 DX=AD% MOD 80
930 DX=DX*8+I        'X座 標 値
940 IF DN=1 THEN PSET(DX,DY-1),7 ELSE PSET(DX,DY),7
950 RETURN
960 '
970 *BCLEAR
980 FOR AD%=0 TO &H7CFF
990   POKE(AD%),0 'メモリを読む
1000 NEXT
1010 RETURN

```

リスト7 輪郭抽出プログラム (Quick Basic)

```

'*****
'** 輪郭抽出プログラム **
'*****
SCREEN 0
CLS
DIM BT(8), BT1(8), BT2(8)
'図形を描く
LINE (0, 0)-(639, 399), 2, B
CIRCLE (200, 200), 170, 1, , , .3
PAINT (200, 200), 1, 1
CIRCLE (340, 220), 100, 1, , , 1.5
PAINT (340, 300), 1, 1
CIRCLE (200, 200), 20, 0
PAINT (200, 200), 0, 0
CIRCLE (600, 200), 30, 1
PAINT (600, 200), 1, 1
'輪郭を求める
PRINT TIME$
GOSUB OUTLINEH: '輪郭抽出水平方向
GOSUB OUTLINEV: '輪郭抽出垂直方向
GOSUB BCLEAR:
PRINT TIME$
END
'*****
'*** 輪郭抽出作業 ***
'*****
OUTLINEH: '水平方向
FOR Y = 0 TO 399
  FD = POINT(0, Y)
  FOR X = 1 TO 639
    D = POINT(X, Y)
    DN = FD - D
    IF DN = 1 THEN PSET (X - 1, Y), 7
    IF DN = -1 THEN PSET (X, Y), 7
    FD = D
  NEXT
NEXT
RETURN
OUTLINEV: '垂直方向
FOR X = 0 TO 639
  FD = POINT(X, 0)
  FOR Y = 1 TO 399
    D = POINT(X, Y)
    DN = FD - D
    IF DN = 1 THEN PSET (X, Y - 1), 7
    IF DN = -1 THEN PSET (X, Y), 7
    FD = D
  NEXT
NEXT

```

```

NEXT
RETURN
'
BCLEAR:
  DEF SEG = &HA800
  FOR AD% = 0 TO &H7CFF
    POKE (AD%), 0 'メモリを読む
  NEXT
RETURN

```

リスト8 輪郭追跡プログラム (Quick Basic)

```

'*****
'** 輪郭追跡プログラム **
'*****
SCREEN 0
CLS
DIM NB(8)
'図形を描く
LINE (0, 0)-(639, 399), 2, B
LINE (100, 50)-(250, 150), 1, B: X = 150: Y = 60: GOSUB TRACE:
CIRCLE (350, 100), 40, 1: X = 350: Y = 100: GOSUB TRACE:
CIRCLE (500, 100), 40, 1, , , .5: X = 500: Y = 100: GOSUB TRACE:
'輪郭を求める
PRINT TIME$
END
'*****
'*** 輪郭追跡作業 ***
'*** 時計廻り ***
'*****
TRACE:
  CL = 0
  WHILE (CL <> 1)
    CL = POINT(X, Y) '色番号
    IF CL <> 1 THEN Y = Y - 1:
  WEND
  XE = X: YE = Y '開始時の座標値
  XF = X: YF = Y
  GOSUB DIRECT1:
  GOSUB AROUND:
  GOSUB DIRECTION:
  X = XL(DR): Y = YL(DR): PSET (X, Y + 150), 2
  XD = X - XF: YD = Y - YF
  DO
    IF XD > 0 AND YD > 0 THEN GOSUB DIRECT2:
    IF XD > 0 AND YD = 0 THEN GOSUB DIRECT3:
    IF XD > 0 AND YD < 0 THEN GOSUB DIRECT4:
    IF XD < 0 AND YD = 0 THEN GOSUB DIRECT5:
    IF XD < 0 AND YD > 0 THEN GOSUB DIRECT6:
    IF XD < 0 AND YD < 0 THEN GOSUB DIRECT7:
    IF XD = 0 AND YD < 0 THEN GOSUB DIRECT8:
    IF XD = 0 AND YD > 0 THEN GOSUB DIRECT9:
    GOSUB AROUND: '周辺画素の値
    GOSUB DIRECTION: '追跡方向
    X = XL(DR): Y = YL(DR)
    PSET (X, Y + 150), 2
    XD = X - XF: YD = Y - YF
    XF = X: YF = Y
    IF XD = 0 AND YD = 0 THEN EXIT DO
    IF X = XE AND Y = YE THEN EXIT DO
    XF = X: YF = Y
  LOOP
RETURN
'
DIRECT1:
  XL(1) = X + 1: YL(1) = Y
  XL(2) = X + 1: YL(2) = Y - 1
  XL(3) = X: YL(3) = Y - 1

```

```

XL(4) = X - 1: YL(4) = Y - 1
XL(5) = X - 1: YL(5) = Y
XL(6) = X - 1: YL(6) = Y + 1
XL(7) = X + 1: YL(7) = Y + 1
XL(8) = X: YL(8) = Y - 1
RETURN

```

DIRECT2:

```

XL(1) = X + 1: YL(1) = Y + 1
XL(2) = X + 1: YL(2) = Y
XL(3) = X: YL(3) = Y + 1
XL(4) = X + 1: YL(4) = Y - 1
XL(5) = X - 1: YL(5) = Y + 1
XL(6) = X: YL(6) = Y - 1
XL(7) = X - 1: YL(7) = Y
XL(8) = X - 1: YL(8) = Y - 1
RETURN

```

DIRECT3:

```

XL(1) = X + 1: YL(1) = Y
XL(2) = X + 1: YL(2) = Y - 1
XL(3) = X + 1: YL(3) = Y + 1
XL(4) = X: YL(4) = Y - 1
XL(5) = X: YL(5) = Y + 1
XL(6) = X - 1: YL(6) = Y - 1
XL(7) = X - 1: YL(7) = Y + 1
XL(8) = X - 1: YL(8) = Y
RETURN

```

DIRECT4:

```

XL(1) = X + 1: YL(1) = Y - 1
XL(2) = X + 1: YL(2) = Y
XL(3) = X: YL(3) = Y - 1
XL(4) = X + 1: YL(4) = Y + 1
XL(5) = X: YL(5) = Y - 1
XL(6) = X - 1: YL(6) = Y - 1
XL(7) = X - 1: YL(7) = Y + 1
XL(8) = X - 1: YL(8) = Y
RETURN

```

DIRECT5:

```

XL(1) = X - 1: YL(1) = Y
XL(2) = X - 1: YL(2) = Y + 1
XL(3) = X - 1: YL(3) = Y - 1
XL(4) = X: YL(4) = Y + 1
XL(5) = X: YL(5) = Y - 1
XL(6) = X + 1: YL(6) = Y - 1
XL(7) = X + 1: YL(7) = Y + 1
XL(8) = X + 1: YL(8) = Y
RETURN

```

DIRECT6:

```

XL(1) = X - 1: YL(1) = Y + 1
XL(2) = X: YL(2) = Y + 1
XL(3) = X - 1: YL(3) = Y
XL(4) = X - 1: YL(4) = Y - 1
XL(5) = X + 1: YL(5) = Y + 1
XL(6) = X: YL(6) = Y - 1
XL(7) = X + 1: YL(7) = Y
XL(8) = X + 1: YL(8) = Y - 1
RETURN

```

DIRECT7:

```

XL(1) = X - 1: YL(1) = Y - 1
XL(2) = X - 1: YL(2) = Y
XL(3) = X: YL(3) = Y - 1
XL(4) = X - 1: YL(4) = Y + 1
XL(5) = X + 1: YL(5) = Y - 1
XL(6) = X + 1: YL(6) = Y
XL(7) = X: YL(7) = Y + 1

```

```
XL(8) = X + 1: YL(8) = Y + 1
RETURN
```

```
DIRECT8:
```

```
XL(1) = X: YL(1) = Y - 1
XL(2) = X + 1: YL(2) = Y - 1
XL(3) = X - 1: YL(3) = Y - 1
XL(4) = X + 1: YL(4) = Y
XL(5) = X - 1: YL(5) = Y
XL(6) = X + 1: YL(6) = Y + 1
XL(7) = X - 1: YL(7) = Y + 1
XL(8) = X: YL(8) = Y + 1
RETURN
```

```
DIRECT9:
```

```
XL(1) = X: YL(1) = Y + 1
XL(2) = X + 1: YL(2) = Y + 1
XL(3) = X - 1: YL(3) = Y + 1
XL(4) = X - 1: YL(4) = Y
XL(5) = X + 1: YL(5) = Y
XL(6) = X - 1: YL(6) = Y - 1
XL(7) = X + 1: YL(7) = Y - 1
XL(8) = X: YL(8) = Y - 1
RETURN
```

```
AROUND: '周辺画素の値
```

```
AR(1) = POINT(XL(1), YL(1))
AR(2) = POINT(XL(2), YL(2))
AR(3) = POINT(XL(3), YL(3))
AR(4) = POINT(XL(4), YL(4))
AR(5) = POINT(XL(5), YL(5))
AR(6) = POINT(XL(6), YL(6))
AR(7) = POINT(XL(7), YL(7))
AR(8) = POINT(XL(8), YL(8))
```

```
RETURN
```

```
DIRECTION: '追跡方向
```

```
DR = 0
FOR I = 1 TO 7
  IF AR(I) = 1 THEN DR = I: EXIT FOR
NEXT
RETURN
```

リスト9 画像合成プログラム (N88DISKBASIC)

```
10 '*****
20 '*** 簡易画像合成プログラム ***
30 '*** By Hideo KASAI ***
40 '*****
50 SCREEN 3,0,0,1
60 CONSOLE 0,25,0,1:SCREEN 3,0:DEFINT A-Z:WIDTH 80,25
70 *MENU
80 CLS
90 COLOR 5:LOCATE 17,0:PRINT "**** 画像合成 ****":COLOR 4
100 LOCATE 20,2:PRINT "1.画像ロード"
110 LOCATE 20,3:PRINT "2.画像保存 "
120 LOCATE 20,4:PRINT "3.取り出し "
130 LOCATE 20,5:PRINT "4.はめ込み "
135 LOCATE 20,6:PRINT "5.画面消去 "
140 LOCATE 20,7:PRINT "6.終了 "
150 LOCATE 20,9:INPUT "番号は ",BG
160 ON BG GOSUB *GLOAD,*GSAVE,*GGET,*GPUT,*GCLS,*PEND
170 GOTO *MENU
180 *PEND
190 END
200 '
210 *GLOAD: '()
220 '画像ロード (青画面)
```

```

230 CLS :INPUT "テ-タメイ =" ;FILENAME$
240 DEF SEG=&HA800: BLOAD FILENAME$
250 RETURN
260 '
270 *GSAVE: ' ( )
280 '画像保存
290 CLS :INPUT "テ-タメイ =" ;FILENAME$
300 DEF SEG=&HA800: BSAVE FILENAME$ , &H0 , &H8000
310 RETURN
320 '
330 *GGET: ' ( )
340 '画像の一部取出し
350 *INIT
360 CLS :INPUT "テ-タメイ =" ;FILENAME$
370 CLS:PRINT "*** 取り出し範囲指定 ***"
380 CSX=100:CSY=200:GOSUB *CURSOR:Y1=CY:X1=CX:PSET(CX,CY),7
390 CSX=CX-4:CSY=CY-4:GOSUB *CURSOR:Y2=CY:X2=CX:PSET(CX,CY),7
395 CLS:PRINT "*** 取り出し中 ***"
400 OPEN FILENAME$ FOR OUTPUT AS #1:WRITE #1,Y1,Y2,X1,X2
410 FOR I=Y1 TO Y2:FOR J=X1 TO X2 STEP 8
420 X=J:Y=I:GOSUB *ADRESCALC:WRITE #1,A:PRINT A
430 NEXT:NEXT
440 CLOSE
450 RETURN
460 '
470 *GPUT: ' ( )
475 '画面のはめ込み
480 CLS:INPUT "テ-タメイ =" ;FLNAME$
490 INPUT "色番号";CL
500 ON CL GOTO *C1,*C2,*C3,*C4,*C5,*C6,*C7
510 *C1:DEF SEG=&HA800:LP=1:GOSUB *FINPUT:RETURN
520 *C2:DEF SEG=&HB000:LP=1:GOSUB *FINPUT:RETURN
530 *C3:DEF SEG=&HB000:LP=1:GOSUB *FINPUT:LP=2:DEF SEG=&HA800:GOSUB *FINPUT:RETU
RN
540 *C4:DEF SEG=&HB800:LP=1:GOSUB *FINPUT:RETURN
550 *C5:DEF SEG=&HA800:LP=1:GOSUB *FINPUT:LP=2:DEF SEG=&HB800:GOSUB *FINPUT:RETU
RN
560 *C6:DEF SEG=&HB800:LP=1:GOSUB *FINPUT:LP=2:DEF SEG=&HB000:GOSUB *FINPUT:RETU
RN
570 *C7:DEF SEG=&HB800:LP=1:GOSUB *FINPUT:LP=2:DEF SEG=&HB000:GOSUB *FINPUT:LP=3
:DEF SEG=&HA800:GOSUB *FINPUT
580 RETURN
590 '
591 *GCLS
592 ' 画面消去
593 CLS 3
594 RETURN
595 '
600 '----- サブルーチン -----
610 *CURSOR: ' (CX,CY)
620 'カーソル移動
630 LINE(0,10)-(20,10),4:LINE(10,0)-(10,20),4
640 BYTE=((20+7)¥8)*20*3+4:FACT=BYTE¥2+1
650 DIM CS%(FACT)
660 GET@(6,6)-(14,14),CS%
670 LINE(0,0)-(20,20),0,BF
680 '
690 CX=CSX:CY=CSY
700 PUT@(CX,CY),CS%,XOR
710 CH=ASC(INPUT$(1)):CCX=CX:CCY=CY
720 IF CH=13 THEN GOTO 810
730 IF CH=28 THEN CX=CX+1
740 IF CH=29 THEN CX=CX-1
750 IF CH=30 THEN CY=CY-1
760 IF CH=31 THEN CY=CY+1
770 GOSUB *LIMIT
780 PUT@(CCX,CCY),CS%,XOR
790 PUT@(CX,CY),CS%
800 GOTO 710
810 PUT@(CX,CY),CS%,XOR

```

```
820 CX=CX+4 :CY=CY+4
830 ERASE CS%
840 RETURN
850 '
860 *LIMIT: '(CX,CY)
870 'カーソル移動限界
880 IF CX<0 THEN CX=599
890 IF CX>600 THEN CX=0
900 IF CY<0 THEN CY=399
910 IF CY>399 THEN CY=0
920 RETURN
930 '
1000 *FINPUT: '(lp)
1010 OPEN FLNAME$ FOR INPUT AS #1:INPUT #1,Y1,Y2,X1,X2
1020 IF LP<>1 THEN 1160
1030 PRINT "はめ込む画面の大きさは"
1032 PRINT "(";X1;"",";Y1;")から(";X2;"",";Y2;")まで 横";X2-X1;"ドット ";Y2-Y1"ドットです"
1040 INPUT "始点を縦方向に何ドット下げますか=";NN
1070 SY=Y1+NN:EY=Y2+NN
1110 INPUT "始点を横方向に何ドットずらしますか=";NN
1120 SX=X1+NN:EX=X2+NN
1130 PRINT "(";SX;"",";SY;")から(";EX;"",";EY;")までとなります"
1140 INPUT "sure(y/n)";YN$
1150 IF YN$="n" OR YN$="N" THEN CLS:GOTO 1040
1155 LINE(SX,SY)-(EX,EY),0,BF
1160 FOR I=SY TO EY:FOR J=SX TO EX STEP 8
1170 INPUT #1,DATA.:X=J:Y=I:GOSUB *ADRESCALC:POKE(ADR),DATA.
1180 NEXT:NEXT
1190 CLOSE
1200 RETURN
1210 '
1220 *ADRESCALC: '(X,Y,A,ADR)
1230 'グラフィック座標から番地の計算,データ呼出
1240 ADR=INT(X/8)+Y*80:A=PEEK(ADR)
1250 RETURN
```