JAPLA 研究会資料 2013/1/26

JによるBMP画像データのシステム処理
スキャナ画像からの画像のゴミをきれいに取り除く

西川　利男

現在では、多くの人にとってのパソコン使用の機会は、今や年末恒例行事でもある年賀状の作成印刷のときが唯一ではなかろうか。

わが家では、ワイフが絵をやっているので、手書きの絵を描き、それを元にパソコンソフトの「筆まめ」などで年賀状を作る、ということをやっている。絵の原稿はスキャナから読み取り、それを文面に貼り付けると、絵の白い部分が青っぽいねずみ色になったりして、文字の文面部分の白色となじまない。アーティストでもあるワイフはこれでは満足しない。

いくら真っ白い紙に絵を描いても、スキャナでやる限り、物理的原理からして無理である。さりとて「ペイント」の「消しゴム」で、手ですべてきれいに消すのはあまりに大変である。

私もコンピュータ・サイエンティストの意地をかけて、正月早々のひとつの課題として、取り組むことになってしまった。まず、ビットマップ画像の解析から始めて、Jのプログラムにより何とか実用の一歩手前までこぎつけた。

下の画面で、へびの首の部分の周りが、この処理でゴミがとれて真っ白になったのが分かるだろう。

1.いろいろな画像ファイル

画像ファイルにはいろいろあるが、大別すると3通りに分かれる。

・**BMP ファイル**　ピクセルとメモリとが直接対応、画像圧縮処理をしていないため、ファイルの大きさは非常に大きくなるが、画像の劣化はない。

・**GIF ファイル、PNG ファイル**　ホームページで使われる画像圧縮のファイル。

・**JPEG ファイル**　フルカラーの写真などを非常に高い圧縮方法で効率的に保存するので、ファイルは小さくなるが、画像の忠実さはさけられない。

さらに BMP(ビットマップ)ファイルにも、

24bit ビットマップ
256 色ビットマップ
モノクロビットマップ

のようにあるが、今回は最も良く使われる 24bit ビットマップについてのみ、解析およびデータ処理を行った。

2.J のシステム処理のための基本のプログラム・ツール

ファイルやメモリの内容を調べるには、バイナリデータを扱う基本のプログラム・ツールが必要である。J ではコンパクトなプログラムでこのようなシステム処理が行える。

・値(10 進) → 文字　　chr 97 65 => aA
・文字 → 値(10 進)　　val 'aA' => 97 65
・文字 → 値(16 進)　　hdump 'aA' => 61 41
・10 進表示 → 16 進表示　hfd 97 65 => 61 41
・16 進表示 → 10 進表示　dfh '61' => 97 , dfh '41' => 65

プログラム定義は以下のとおりである。

```
h=.   '0123456789ABCDEF'
dfh=: 16&#. @ (h&i.) f.
d2h2=: (16,16)&#. @ (h&i.) f.        NB. for 2-digits
hfd=: dfh ^:_1 f.
h2d2=: d2h2 ^:_1 f.                  NB. for 2-digits

val=: a. & i.
chr=: val ^:_1
hdump=: }:@,@(,"1' '"_)@h2d2@val f.  NB. revised for 2-digits
```

MS-DOS システムで重宝されたファイルのデバッグ DEBUG コマンドに準拠した J のツール bmp_dump を作った。これを今回の解析に利用したが、実際の使用例は後に示す。

また、J のシステムには、BMP ファイルを扱うためのいろいろな便利なツールが備えられているので、以下のように require により参照しておくのがよい。

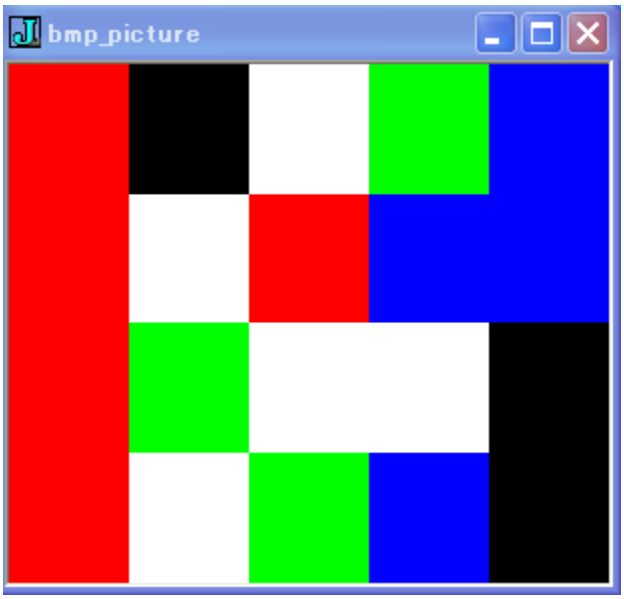
```
require 'system¥packages¥graphics¥bmp.ijs'
```

3.24 ビット BMP ファイルの構造と解析

具体的に、小さな画像ファイルを作り、それをダンプすることで BMP ファイルの構造を調べてみよう。

まず、上の bmp ライブラリにある viewbmp により、例として小さな画像ファイル bm24.bmp を見てみる。このファイルは通常のペイントでは小さくて良く見えない。

```
viewbmp 'user¥bm24.bmp'
```



つぎに、この内容ををを先のツール bmp_dump によりダンプしてみる。

```
bmp_dump 'user¥bm24.bmp'
 _0 _1 _2 _3 _4 _5 _6 _7 _8 _9 _A _B _C _D _E _F
0: 42 4D 76 00 00 00 00 00 00 00 36 00 00 00 28 00
1: 00 00 05 00 00 00 04 00 00 00 01 00 18 00 00 00
2: 00 00 40 00 00 00 CE 0E 00 00 C4 0E 00 00 00 00
3: 00 00 00 00 00 00 00 00 FF FF FF FF 00 FF 00 FF
4: 00 00 00 00 00 00 00 00 FF 00 FF 00 FF FF FF FF
5: FF FF 00 00 00 00 00 00 FF FF FF FF 00 00 FF FF
6: 00 00 FF 00 00 00 00 00 FF 00 00 00 FF FF FF 00
7: FF 00 FF 00 00 00
```

ファイルの始めの部分は次の内容を示している。

```
00 01       | 42 4D        文字'B M'により、BMP ファイルであることを示す。
02 03 04 05 | 76 00 00 00  ファイル長　76 バイトを示す。
06 07 08 09 | 00 00 00 00  予備、現在使用していない。
0A 0B 0C 0D | 36 00 00 00  ピクセル開始番地
0E 0F 10 11 | 28 00 00 00  ヘッダー開始番地
```

12 13 14 15 | 05 00 00 00 画像の幅 Width
16 17 18 19 | 04 00 00 00 画像の高さ Height
1A 1B | 01 00 カラープレイン数、常に 01
1C 1D | 18 00 １ピクセル当たりのビット数、24 ビットを示す。(=18x)

これらのパラメータはツール header により次のように示される。

```
  header 'user¥bm24.bmp'
File size = 00 00 00 76 (hex)
        = 118 bytes (dec)
Type = 24-bits BMP
Start Address of Pixel Data = 36 (hex)
Start Address of Header     = 28 (hex)
Width  = 5
Height = 4
```

それでは、ピクセル値の部分をツール bmp_pixel により、整理して、表示する。

```
  bmp_pixel 'user¥bm24.bmp'
00 00 FF FF FF FF 00 FF 00 FF 00 00 00 00 00 00
00 00 FF 00 FF 00 FF FF FF FF FF FF 00 00 00 00
00 00 FF FF FF FF 00 00 FF FF 00 00 FF 00 00 00
00 00 FF 00 00 00 FF FF FF 00 FF 00 FF 00 00 00
```

これは、4 行、16 列の値を成している。4 行は画像の高さ(Weight)を示す。
ここで 1 行 16 個の値は、つぎのようないきさつから出来ている。
1つのピクセルデータは3バイト(24 ビット)の RGB 値から成っている。従って画像の幅 (Width) 5 から 5 x 3 = 15 となるが、BMP の仕様では1行の値は 4 の倍数でなくてはならない、という規定から、今の場合は1バイトの 00 を追加、調整して1行 16 バイトとしている。

さて、各ピクセル値であるが、1 ピクセルに対して画像の下段から、右方向に 3 バイトずつ、それぞれの色に対応させる。かつ、色対応は RGB ではなく GBR と逆向きとする。

いまの場合、1行目の値は、画像の一番下に対応する。

```
    00 00 FF / FF FF FF / 00 FF 00 / FF 00 00 / 00 00 00 / 00
     あか      しろ        みどり     あお      くろ   ゼロ調整
```

2行目の値は、画像の下から2段目に対応する。

```
   00 00 FF / 00 FF 00 / FF FF FF / FF FF FF / 00 00 00 / 00
     あか     みどり       しろ       しろ       くろ  ゼロ調整
```

3行目の値は、画像の下から3段目に対応する。

```
   00 00 FF / FF FF FF / 00 00 FF / FF 00 00 / FF 00 00 / 00
     あか      しろ        あか       あお       あお      ゼロ調整
```

4行目の値は、画像の一番上に対応する。

```
   00 00 FF / 00 00 00 / FF FF FF / 00 FF 00 / FF 00 00 / 00
     あか      くろ        しろ       みどり    あお      ゼロ調整
```

このように、BMP のフォーマットは原則
1 ピクセル＝3 バイト＝24 ビット

として、行われるものの、ピクセルの配置の位置、ゼロ調整(Zero Pad)と、
あまり合理的とは思えない約束ごとで決められる。

4.BMP ファイルのクリーニング（ごみ取り）

ディジタル装置としてのスキャナとは、紙面で反射した色の光を受光素子で受けて、ディジタル信号としてメモリに記録する。当然、照らす光源、紙面の反射効率などアナログ量に依存する。したがって、現実の絵の中の白い部分も RGB 値 FF FF FF になる保障はない。実際はこれからはずれた RGB 値になり、これが青っぽいねずみ色のごみとなって見えるのだろう。

BMP ファイルのクリーニング(ごみ取り)は、このような RGB 値を強制的に FF FF FF に置き換えることで、真っ白になる。

なお、ここでやっかいなのは、ゼロ調整(Zero Pad)についての取り扱いである。

```
                3 x Width バイト                        Zero Pad
      |-- -- --|-- -- --|-- -- --| ...      |--- --- ---|00 00 ..   |
Height |-- -- --|-- -- --|-- -- --| ...      |--- --- ---|00 00 ..   |
      |-- -- --|-- -- --|-- -- --| ...      |--- --- ---|00 00 ..   |
          ... ... ... ...           ...        ... ...    ... ...
      |-- -- --|-- -- --|-- -- --| ...      |--- --- ---|00 00 ..   |
```

結局、次のように解決して、プログラムを作成した。

まず、ゼロ調整(Zero Pad)の数 NPad を 4 通りの場合に分ける。

(3 x Width) % 4 の余り 0 のとき　NPad = 0

1　NPad = 3

2　NPad = 2

3　NPad = 1

この値を元に、ゼロ調整(Zero Pad)の部分を取り除いた(Height x Width)行、3 列の配列を作り、それについて各行 3 バイトごとに FF FF FF への置き換え（ごみピクセルのクリーニング）を行い、その後、再びゼロ調整(Zero Pad)の部分を元に戻す、ということで処理した。以上の処理は、J の配列の変形で簡単に行えたが、通常の言語のループなどではそう簡単には出来ないだろう。

ここでも、実際の BMP 画像から切り取って小さなファイルを作り、実験を行った。

```
  chapix 'user¥bm24a.bmp'
Input:
A3 65 42 AB 6F 38 C5 96 A2 E7 DB D6 FF FF FF 00
A1 66 35 B4 71 31 CF A7 8A FF FF FF FF FF FF 00
A8 64 34 A3 65 42 B4 71 31 E7 DB D6 FF FF FF 00
NPadz = 1
Output
A3 65 42 AB 6F 38 C5 96 A2 FF FF FF FF FF FF 00
A1 66 35 B4 71 31 CF A7 8A FF FF FF FF FF FF 00
A8 64 34 A3 65 42 B4 71 31 FF FF FF FF FF FF 00
** cleaned fname: user¥bm24az.bmp, length: 102
```

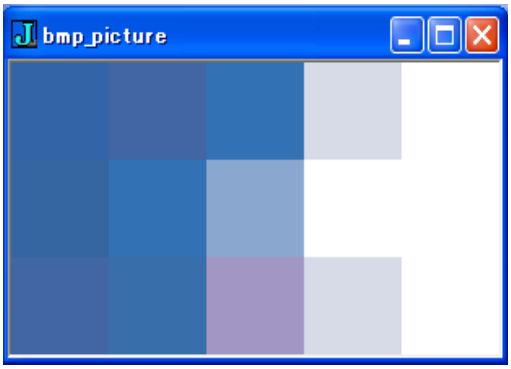
入力した元の画像からの2箇所のピクセル値が、それぞれ

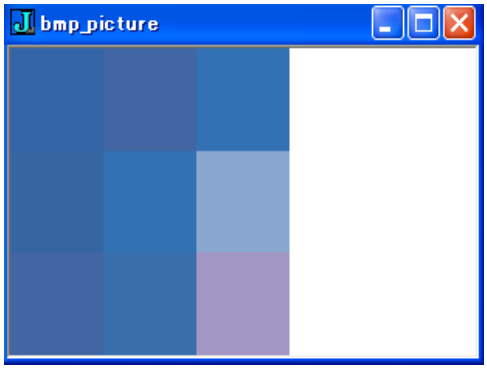
E7 DB D6 => FF FF FF

に置き換えられ、ごみが取り除かれた。

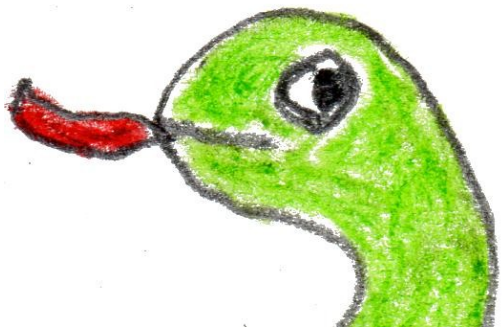
実際に見ると左の画像は右の画像のように修正された。

viewbmp 'user¥bm24a.bmp'　　viewbmp 'user¥bm24az.bmp'





さらに、本物の絵の画像での実際のようすはつぎのようである。左の元の画像にあった地の文章部分との境目は、右のクリーン修正画像ではなくなりきれいになった。



最初のページにあげた絵はこのようにして、ごみを取り除いた画像を元の絵の中に張り付けたものである。

BMP ファイルのクリーニング処理はこのようにうまく出来たが、難点は非常に時間がかかる、ということであった。

しかしながら、いままでのような実験を通して、BMP のファイル構造が具体的に明らかになったのは収穫である。

さらに、このようなシステム処理のプログラミングに J が極めて強力であることが判った。

```
NB. bmp_view_test

require (1!:40 ''), '¥system¥main¥dir.ijs'

NB. Directory of User
NB. e.g. DIR '*.bmp'
DIR =: 3 : 0
dir (1!:40 ''),'¥user¥', y.
)


wr=: 1!:2&2
rd=: 1!:1
each=: &. >

deb=: #~ (+. 1&|.@(> </¥))@(' '&~:)                    NB. delete extra blanks

h=.   '0123456789ABCDEF'
dfh=: 16&#. @ (h&i.) f.
d2h2=: (16,16)&#. @ (h&i.) f.        NB. for 2-digits
hfd=: dfh ^:_1 f.
h2d2=: d2h2 ^:_1 f.                  NB. for 2-digits

NB. hex  (adverb)
NB. e.g. 'FF' + hex '8'
hex=: &. dfh

val=: a. & i.
chr=: val ^:_1
hdump=: }:@,@(,"1' '"_)@h2d2@val f.  NB. revised for 2-digits

require 'system¥packages¥graphics¥bmp.ijs'

NB. Usage: header 'user¥Col16.bmp'
header=: 3 : 0
try. dat=. 1!:1 boxopen y.
catch. 'file read error' return. end.
if. -. 'BM'-:2{.dat do. 'not a bitmap file' return. end.
toi=. 256&#.@(a.&i.)@(|."1)
bits=. toi 28 29 {dat
if. toi 30 31 32 33{dat do.
```

```
  'compressed format not supported' return.
end.
'off shdr cls rws'=. toi (10+i.4 4){dat
wr 'File size = ', (hdump fs =. |. (2 + i.4) {dat), ' (hex)'
wr '          = ', (": 256  #. val fs), ' bytes (dec)'
wr 'Type = ', (":bits), '-bits BMP'
wr 'Start Address of Pixel Data = ', (hfd off), ' (hex)'
wr 'Start Address of Header     = ', (hfd shdr), ' (hex)'
wr 'Width  = ', ":cls
wr 'Height = ', ":rws
''
)

NB. Usage: viewbmp 'user¥bm24.bmp'
viewbmp=: 3 : 0
'f n'=. 2{.(boxopen y.),<'bmp_picture'
if. (<'isipicture') e. <;._2 wd 'qp;' do.
  t=. 'psel isipicture;'
else.
  t=. 'pc isipicture closeok;pn "',n,'";'
  t=. t,'xywh 0 0 150 150;cc p0 isipicture rightmove bottommove;'
  t=. t,'pas 0 0;pcenter;pmove 150 10 -1 -1;'
end.
wd t,'set p0 "',f,'";ptop;pshow;'
)

head=: 3 : 0
try. dat=: 1!:1 boxopen y.
catch. 'file read error' return. end.
if. -. 'BM'-:2{.dat do. 'not a bitmap file' return. end.
toi=. 256&#.@(a.&i.)@(|."1)
bits=. toi 28 29 {dat
if. toi 30 31 32 33{dat do.
  'compressed format not supported' return.
end.
'off shdr cls rws'=. toi (10+i.4 4){dat
cls, rws
)

headN=: 3 : 0  NB. modified by TN
try. dat=: 1!:1 boxopen y.
```

```
catch. 'file read error' return. end.
if. -. 'BM'-:2{.dat do. 'not a bitmap file' return. end.
toi=. 256&#.@(a.&i.)@(|."1)
bits=. toi 28 29 {dat
if. toi 30 31 32 33{dat do.
  'compressed format not supported' return.
end.
'off shdr cls rws'=. toi (10+i.4 4){dat
off, cls, rws NB. addresss of pixel and width, height
)
```

```
NB. dump bmp-file ======================================
NB.   2012/11/4
Col_index =: '_0 _1 _2 _3 _4 _5 _6 _7 _8 _9 _A _B _C _D _E _F '
Col_index =: 1 48$Col_index

NB. dump bmp-file
NB. e.g. bmp_dump 'user¥bm24.bmp'
NB. e.g. bmp_dump 'user¥sc1.bmp'
bmp_dump =: 3 : 0
BD =. 1!:1 boxopen y.
BH =. hdump BD
Col =. 48 NB. Column = 48
Row =. >. ($BH) % Col
RowInd =. hfd i. Row
Row_index =. ' ', RowInd,"(1) ': '
Row_index,"(1) Col_index, (Row, Col)$BH, 48#' '
)

NB. bmp_pixel 'user¥bm24.bmp'
NB.   using headN as subroutine
bmp_pixel =: 3 : 0
BD =. 1!:1 boxopen y.
'Off Width Height' =: headN y.
BMH =: Off {. BD   NB. Header of BMP file
PIX =: Off }. BD   NB. Pixel of BMP file
wr Width
Row =. ($PIX) % Height
PIXX =. (Height, Row)$PIX
NB. PIXY =. ((Height*2), (Row%2))$PIX
hdump"(1) PIXX
)

NB. clean bmp-file 2012/12/22
NB. no change => 0 chapix 'user¥bm24a.bmp'
NB. change    => 1 chapix 'user¥bm24a.bmp'
NB. using to_FF as subroutine
NB. output as bmp-file added 'z' => 'user¥bm24az.bmp'

chapix =: 3 : 0
1 chapix y.
:
```

```
change =. x.
BD =. 1!:1 boxopen y.
'Off Width Height' =: headN y.
BMH =: Off {. BD    NB. Header of BMP file
PIX =: Off }. BD    NB. Pixel of BMP file
Row =. ($PIX) % Height
PIXX =. (Height, Row)$PIX
wr 'Input:'
wr hdump"(1) PIXX
if. 0 = change do. '    ***' return. end.

select. 4 | 3 * Width
  case. 3 do. NPadz =. 1
  case. 2 do. NPadz =. 2
  case. 1 do. NPadz =. 3
  case. 0 do. NPadz =. 0
end.
wr 'NPadz = ',  ": NPadz
PIXY =. (- NPadz) }. "(1) PIXX  NB. remove Padz(00)
NB. wr hdump"(1) PIXY

PIY =. , PIXY
PIYA =. ( ((#PIY)%3), 3) $ PIY
if. change = 1 do. PIZA =. to_FF PIYA else. PIZA =. PIYA end.
hdump"(1) PIZA

wr 'Output'
PIZB =. (Height, (3*Width))$ , PIZA
PIZC =. PIZB,"(1) (NPadz # chr 0)
wr hdump"(1) PIZC
PIZ =: , PIZC
fname =. ((_4) }. y. ), 'z.bmp'
f_length =. (BMH, PIZ) fwrite <fname
'** cleaned fnames: ', fname, ', length: ', ": f_length
)

to_FF =: 3 : 0"(1)
if. * / 180 < val y. do. 3 # chr 255 else. y. end.
)
```