画像処理のためのソフトウェア：

(7) 色数削減のアルゴリズム

中野 司*

Softwares for Digital Image Processing :

(7) An Algorithm for Color Reduction

Tsukasa NAKANO*

Abstract : An algorithm to reduce the number of colors in a color image is explained. To bring the appearance frequency of the colors close the same, the number of colors in an image is reduced by dividing three-dimensional space (RGB-space) in which the colors are distributed. The amount of information included in the reduced image is left by this algorithm as much as possible. A software library to reduce the number of colors both in the RGB and palette color images is introduced for the realization of this algorithm. For the application of this library, the UNIX command “tiffreduce” is introduced. This command is available for the color reduction of a TIFF image file.

Key Words : Image Processing, Color Reduction, RGB Color, Palette Color, TIFF

1. はじめに

自然な質感を表すコンピュータグラフィックス画像（たとえば、中野，1992）や微細な密度構造を表現するX線CT画像（たとえば、稲崎・中野，1993）などでは、非常に多数の色を用いることが必要である。ところが、現在のところ、一般に使われているディスプレイ（フレームバッファ）などのハードウェアでは限られた色数しか取り扱えない場合が多いため、作成した画像を容易に出力できないことがしばしば生じる（たとえば、UNIX ワークステーションで標準的な仕様となっている X Window System でも 256 色程度しか表示できない Pseudo Color タイプのものがほとんどである）。このような出力ハードウェアの性能を考慮して作成の段階で画像に限られた色数だけしか用いないことは、そのソフトウェアの負担を増大させ、また、応用の幅をせばめることになる。それゆえ、画像を作成する段階では自由に色を用い、その後、適当な画像処理によって画像中の色数を削減することが望ましい。

本文では、多数の色を用いた画像から、それに含まれる情報量をできる限り残して色数を削減するアルゴリズムについて説明する。また、そのアルゴリズムの実現例として作成したC言語によるライブラリソフトウェアを紹介し、

さらに、その利用例として、画像ファイルとして広く一般に用いられている Rev. 5.0 の TIFF (Tag Image File Format ; Aldus and Microsoft, 1988) 画像ファイル中の色数を削減するソフトウェア “tiffreduce” を紹介する。

2. 色数削減のアルゴリズム

画像中の色数を削減するアルゴリズムの概略を第1図に示した。ここで紹介するアルゴリズムは、以下の2つのステップに分けることができる。まず、色情報の格納の仕方（RGB カラーとパレットカラー；後述）の異なる2種類の画像に対して、それらを一方に統合し、また、冗長な色情報の削除を行う。その後、画像に含まれる色の情報量をできる限り残して、使用する色の数を減少させる。

2-1. RGB カラーとパレットカラー

物体の色の表現法には様々なものがあるが（たとえば、金子，1988），コンピュータ上での画像の色づけは、現在ではほとんどすべて RGB 方式と呼ばれるもので行われている（Harrington，1987）。この方式では、画像中のそれぞれの画素が色の R（赤），G（緑），B（青）成分の強度データを持ち、これらが加色混合された結果が各画素の色として用いられる。さらに、コンピュータで取り扱う画像は色成分のデータの格納の仕方によって、以下で述べる RGB カラーとパレットカラー画像に分類できる（第1図に示したように、本文の色数削減のアルゴリズムではこれ

* 地質調査所 Geological Survey of Japan, 1-1-3, Higashi, Tsukuba 305, Japan.

第1図 色数削減のアルゴリズムの概略

ら2種類の画像を入力画像として取り扱う)。

各画素がR, G, B成分のデータを他の画素と独立に持つ画像は、RGBカラー画像と呼ばれている。この画像では、たとえば各画素のR, G, B成分のデータがそれぞれ1バイトで表される256段階の強度の値を持つとき、 $256 \times 256 \times 256$ 種類の色が表現可能で、また、画素数 $\times 3$ バイトの総容量を必要とする(このような画像はフルカラー画像とも呼ばれている)。なお、RGBカラーの画像は、各成分のデータをどのように格納するか(ベクトル的に3成分まとめて格納するか、各成分を独立なプレーンとして格納するか)などによってさらに分類可能であるが、以下ではそれは考慮しない。陰影づけを行う場合に色づけの処理が容易になるため、RGBカラー画像はコンピュータグラフィックスの画像を作成する際にしばしば用いられる(たとえば、中野, 1992)。

RGBカラー画像は作成の際に自由度があるが、その反面無駄も多い。たとえば、先に例としてあげた各成分1バイト容量の画像では、 $256 \times 256 \times 256$ 個以上の画素(正方形の場合これは縦横が4096 $\times$ 4096画素以上の画像に相当)がない場合には、そこに一度も現れていない色が少なくとも1つはあることになる。そこで、出現する色だけを表(カラーパレット表もしくはカラーマップと呼ばれる)にして、各画素の値(ピクセル値)として色そのものの情報でなく表上の色へのインデックス(表上で色につけられた番号; パレット番号と呼ばれる)を格納する方式がしばしば用いられる(同じ色の画素は同じ色番号をピクセル値として持つ)。これをパレットカラー(もしくはカラーマップ)画像と言う。色の番号づけを0から始まる連続した整数とすれば、パレットカラー画像の各画素にはピクセル値として色の総数がおさまる容量のメモリを割り当てただけでよい。それゆえ、パレットカラー画像ではRGBカラー画像

よりも少ない総容量しか必要としない(カラーパレット表のために余分のメモリが必要になるが、それは普通は画像そのものに要するものと比べて非常に少ない)。このように、パレットカラー画像を用いればメモリの節約にはなるが、前もってカラーパレット表を作成しなければならないために画像作成の際の色の選択の自由度は失われる。

RGBカラー画像とパレットカラー画像は相互に変換可能である。すなわち、パレットカラー画像をRGB画像に直すことは言うに及ばず、その逆も以下のようにして行うことができる。たとえば、先の例の各成分1バイト容量のRGB画像の場合、まず各画素にそれぞれのR, G, B成分 r, g, b から以下の式で計算されるパレット番号を割り当てる。

$$\text{パレット番号} = r + g \times 256 + b \times 256 \times 256 \quad (1)$$

その後、すべての画素を調べて、パレット番号(=色)のうち出現しないものを除外した連続した番号をつけ直せばよい。

色数削減のアルゴリズム(第1図)では、RGBカラーの画像が入力画像として与えられた場合は、まず上記のようにしてパレットカラーの画像に変換している。また、画像作成の際に作ったカラーパレット表は冗長なものである可能性もあるので、パレットカラー画像を入力した場合にはそのチェックを行っている。すなわち、入力したカラーパレット表に画像中で使用されていない色や重複する色が登録されているか否かを調べ、もしそれが見つかったなら表から削除するようにしている。

2-2. RGB空間の分割

先にも述べたように、パレットカラー画像では、画像に出現する色の数を M とすると各画素に $0 \sim M-1$ のパレット番号が格納できるメモリを用意すればよい。すなわち、各画素には $\log_2 M$ ビットの容量が必要になる。ところで、情報理論によれば、ある画像の画素当たりの平均情報量 I (単位はビット)は以下の式で表される(たとえば、堀部, 1989)。

$$I = - \sum_{j=0}^{M-1} f_j \log_2 f_j \quad (2)$$

ここで f_j は画像中でのパレット番号 j の色の画素の個数の割合である。すべての色が同じ頻度で用いられている画像ではパレット番号にかかわらず $f_j = 1/M$ だから、画素当たりの平均情報量 I_0 は、

$$I_0 = -M \times \{ (1/M) \times \log_2 (1/M) \} = \log_2 M \quad (3)$$

第2図. グレースケール画像のヒストグラムイコライゼーションの例.

となる. さらに, I と I_0 の間には以下の不等式が常に成り立つ.

$$I \leq I_0 \tag{4}$$

すなわち, 色の出現頻度に偏りがある画像 (たいていの画像がそうである) では, それに含まれる色の平均情報量は画素が保持できるものよりも常に少ない. 言い替えると, 人間の目 (もしくは脳) は画像の平均情報量をとらえているので, たいていの画像では出現頻度を考慮してその色数を減らしても画像全体としての情報量は保持される.

この原理を応用してグレースケール (白黒濃淡) 画像の強調 (グレースケールのつけ直し) を行う手法は, ヒストグラムイコライゼーションと呼ばれている (第2図; 稲崎・中野, 1993). ヒストグラムイコライゼーションでは, 画素にピクセル値として保持された輝度の出現頻度が一樣になるように, 輝度のつけ直しを行う. すなわち, まず各輝度の画素数のヒストグラムを作成し (第2図下半分), それから計算した積算分布の値が等間隔になるように輝度の値を分割する (第2図上半分). たとえば第2図では, 積算分布の上で C1~C2 の間にある輝度 P1~P2 の画素を同じ輝度にするなどによって, 画像に保持される情報量を最大限残して, 画像に用いられている輝度 (グレースケール) の数を減らすことができる.

カラー画像の色の削減の場合には, ピクセル値として色の R, G, B 成分の3つの値 (ただしパレットカラー画像ではピクセル値の示すカラーパレット表にある3つの値) を持つので, グレースケール画像のヒストグラムイコライゼーションの手法をそのまま用いることはできない. すなわち, 第2図に示されたような1次元に並んだピクセル値上の頻度分布の分割でなく, R, G, B 成分の分布する3次元空間 (RGB 空間) 上での分割を行う必要がある. 本文で紹介する色数削減のアルゴリズムでは, 以下のようにして RGB 空間の分割を行っている. なお, ここでは問題を単純化するために, 3次元の RGB 空間ではなく画像の B 成分がすべて同じ値を持つ場合に相当する2次元の RG 空間を用いて分割法の説明を行う.

B 成分が同じ値を持つ画像に出現する色のヒストグラムを RG 空間上に描くと, 第3図 (a) のようになる (ただし, この画像は, もともとは少数の原色に明るさの割合を乗ずることで陰影表現を行っているコンピュータグラフィックス画像であるため, RG 空間上で色は値0から放射状に並んだ線上に現れており, また, 画像に黄色 (RGB 空間なら灰色) が多く使われているため, 対角線上に多数の画素がある). さて, このような RG 空間を正方形であると仮定する. この正方形を同じサイズの4つの正方形 (以下ではこれらをセルと呼ぶ) に分割して, 各セルに含まれ

第3図 RG空間でのヒストグラムイコライゼーションの例。(a) もとの画像の色の出現頻度のヒストグラム、(b) RG空間を分割した例、(c) 色数削減後の画像のヒストグラム。

る画素の個数を調べる。そして、これらのセルのうち画素の個数が最大のものをさらに4分割する。このように、分割によって様々なサイズをとることになるセルから最大個数の画素を含むものを探し出し、それをさらに分割することを繰り返す。ただし、あるセルが1種類の色しか含まない場合には、それは分割の対象から除外する。最終的には、セルが指定した個数に達するか、すべてのセルに1種類の色だけしか含まなくなったときに分割を終了する。

このようにして得られたセル毎に、そこに含まれる色の出現頻度（画素数）を考慮した重心を計算してその座標値を新しい色の各成分とする。これらの処理の結果の例として、第3図(b)には第3図(a)のRG空間を26個のセルに分割したものを示した。また、第3図(c)には、各セルの重心上にそれに含まれる画素数を示したヒストグラムを描いた。第3図(a)に比べると、このヒストグラムはそれら各々の高低差が小さくなっていることがわかる。すなわち、新しい画像では、色の出現頻度が以前よりずっと一様になっている。また、これらの図で、多数の画素があるRG空間の中央付近は細かい分割が行われている。これはすなわち、もとの画像で出現頻度の高い（広い面積を占める）色が新しい画像でも独立した色となることを示している。さらに、各セルの重心を新しい色と見なす

ため、大きなサイズのセルを代表する色は、そのセル内で最も出現頻度が高い色に近いものになる。

以上がRGB空間に対してヒストグラムイコライゼーションを拡張した色数削減の手法である。ただし、RG空間でなく実際のRGB空間では、各ステップでセルを8分割することが必要になる。このようなRGB空間の分割は、Quick Sortのアルゴリズム（たとえば、石畑, 1989）を応用すれば容易に実現可能である。

なお、色数削減を行う上記の手法には以下の2つの問題点がある。第1の問題点は、RGB空間の歪みに関わるものである。ここで述べたRGB空間の分割のアルゴリズムでは、空間内の2点間の距離が通常のユークリッド距離として、2つの色の類似性をRGB空間でのそれらの間の距離の大小で計っている。しかしながら、実際にはRGB空間は歪んでいて、2つの色の類似性とそれらの間の距離は線形な比例関係にない（それらの位置も関係する）。これを正すためには、たとえばガンマ補正(Harrington, 1987)などが用いられる。ここで述べたアルゴリズムや後で示すソフトウェアではこのような補正を考慮していないが、それを行うことは容易である（RGB空間を分割する前に、RGB空間の歪みをなくすような色の修正を行えばよい）。

第2の問題点は、分割をおこなうセルの個数の決定に関

するものである。セルの個数（削減後の色数）が少なすぎると、もとの画像の情報は大きく損なわれてしまう。この個数は、理論的には第（2）式で得られるその画像の平均情報量（*I*ビット）を保持可能な個数（ 2^I 個）以上あればよいが、実際にはここで示した RGB 空間分割の手法の冗長さのためにより多くのものが必要である。それゆえ、現在のところ、この個数は試行錯誤で決定している。

3. サンプルソフトウェアの説明

前節で説明した画像中の色数削減のアルゴリズムの実現例として、C 言語で書いたソフトウェアライブラリとコマンド `tiffreduce` のソースコードを付録に示した（これらの入ったファイル名の一覧は第 1 表に示されている）。これらはすべて UNIX 上でコンパイルして利用することができる。以下ではその詳細を説明する。

3-1. データ構造と関数

付録にソースコードを示した画像中の色数削減のためのソフトウェアライブラリには、第 2 表のデータ構造と第 3、4、5 表の関数（もしくはマクロ宣言）が用意されている。

コンピュータ上でカラー画像のデータを取り扱うために、構造体 `Image` を用意した（第 2 表）。この構造体は、色の 3 成分の容量が各々 1 バイト以下の RGB 画像と、各画素のピクセル値が 3 バイトにおさまる値を持つカラーパレット画像のどちらに対しても利用できる。関数 `AllocImage`（第 3 表）で画像の縦横サイズを指定して初期化（メモリ確保）した後、構造体 `Image` は以下のように使用する。

第 1 表 ソースコードなどの入ったファイル名一覧

ファイル名	内 容
cmimage.h	変数（第 2 表）や関数（第 3、4 表）の型などの定義
cmimage.c	関 数 <code>AllocImage</code> , <code>FreeImage</code> , <code>AllocCM</code> , <code>FreeCM</code> のソースコード
rgb2cm.c	関数 <code>RGB2CM</code> （第 4 表）のソースコード
cm2cm.c	関数 <code>CM2CM</code> （第 4 表）のソースコード
reducecm.c	関数 <code>ReduceCM</code> （第 4 表）のソースコード
readtiff.h	関数 <code>ReadTIFF</code> , <code>ReadTIFFError</code> （第 5 表）の型の定義
readtiff.c	関数 <code>ReadTIFF</code> , <code>ReadTIFFError</code> （第 5 表）のソースコード
writetiff.h	関数 <code>WriteTIFF</code> , <code>WriteTIFFError</code> （第 5 表）の型の定義
writetiff.c	関数 <code>WriteTIFF</code> , <code>WriteTIFFError</code> （第 5 表）のソースコード
copytag.c	関数 <code>CopyTAG</code> （第 5 表）のソースコード
tiffreduce.c	コマンド “ <code>tiffreduce</code> ” のソースコード
Makefile	コマンド “ <code>tiffreduce</code> ” の <code>make</code> 用スクリプト
tiffio.h	TIFF ライブラリのヘッダファイル
libtiff.a	TIFF ライブラリ

（注）“`tiffio.h`”と“`libtiff.a`”のソースコードは付録に示していない。

まず、RGB カラー画像に対して用いる場合には、構造体の要素 `r`, `g`, `b` を色の `R`, `G`, `B` 成分をおさめた 2 次元配列として取り扱う（これらの配列の各要素、たとえば `r[x][y]` が画像上の位置 `(x, y)` の画素の色の `(R)` 成分に対応）。また、パレットカラー画像の場合には、それらの配列の要素を第（1）式のように組み合わせて各画像のピクセル値を計算する。ただし、第（1）式の計算はめんどうな上に書き間違えと致命的なエラーとなりかねない。それで、この計算を行うために、マクロ宣言 `GETPIXEL` と `PUTPIXEL` が用意されている（第 3 表）。

画像がパレットカラーである場合には、色データ（カラーパレット表）を構造体 `CM` に格納して使用する（第 2 表）。関数 `AllocCM`（第 3 表）で色の個数を指定してこの構造体にメモリを割り当てた後、その要素 `r`, `g`, `b` に色の `R`, `G`, `B` 成分を格納する。これらの要素はそれぞれ 1 次元配列になっているので、そのインデックスをパレットカラー画像の各画素のピクセル値とすればよい。なお、パレットカラー画像の色の各成分のデータを 2 バイト整数で表すソフトウェアが多いため、構造体 `Image` の場合と異なり、構造体 `CM` の要素 `r`, `g`, `b` は 2 バイトの容量を持っている。それで、後述する関数 `RGB2CM` で RGB カラー画像をパレットカラー画像に変換する際には、RGB カラー画像の色成分のデータの大きさを適当に調節してカラーパレット表に格納している（このサイズを RGB カラー画像のものと合わせたい場合には、構造体 `CM` の色成分のデータを 256 で割ればよい）。

前節で説明した画像中の色数削減のアルゴリズムは、関数 `RGB2CM`, `CM2CM` および `ReduceCM`（第 4 表）で実

第 2 表 変数の型（宣言は “`cmimage.h`” および “`tiffio.h`” で行われている）

型名	説明と定義
byte	バイトデータ（1 バイト長）を格納する変数の型。 定義： <code>unsigned char</code>
word	ワードデータ（2 バイト長）を格納する変数の型。 定義： <code>unsigned short</code>
Image	RGB およびパレットカラーの画像を格納する構造体。 定義： <code>struct {</code> <code>int xSize,ySize; ... 画像のサイズ</code> <code>byte **r,**g,**b; ... 画像データ</code> <code>}</code>
CM	カラーパレットを格納する構造体。 定義： <code>struct {</code> <code>int pixels; ... カラーパレットの総数</code> <code>word *r,*g,*b; ... カラーパレットデータ</code> <code>}</code>
TIFF	TIFF ファイルのアクセスのための構造体（TIFF ファイルポインタ）。 UNIX のファイルアクセス用構造体 <code>FILE</code> とほぼ同じ用途に用いる。

第3表 ソースコード “cmimage.c” の関数
 (宣言は “cmimage. h” で行われている)

関数名	説明と定義
AllocImage	新しい画像(Image 構造体)にメモリを割り付ける。 処理に失敗すると NULL を、成功すると新しい Image 構造体へのポインタを返す。 定義： Image *AllocImage(xSize, ySize) int xSize, ySize; ... 画像のサイズ
FreeImage	画像に割り付けられたメモリを解放する。 定義： void FreeImage(image) Image *image; ... Image 構造体へのポインタ
PUTPIXEL	画像の指定した画素にピクセル値をおさめる (これは関数ではなくマクロ)。 定義： void PUTPIXEL(image, x, y, pixel) Image *image; ... Image 構造体へのポインタ int x, y; ... 画素の位置 int pixel; ... ピクセル値
GETPIXEL	画像の指定した画素のピクセル値を調べる (これは関数ではなくマクロ)。 定義： int GETPIXEL(image,x,y) Image *image; ... Image 構造体へのポインタ int x,y; ... 画素の位置
AllocCM	新しいカラーパレット (CM 構造体) にメモリを割り付ける。 処理に失敗すると NULL を、成功すると新しい CM 構造体へのポインタを返す。 定義： CM *AllocCM(pixels) int pixels; ... カラーパレットの総数
FreeCM	カラーパレットに割り付けられたメモリを解放する。 定義： void FreeCM(cm) CM *cm; ... CM 構造体へのポインタ

第4表 ソースコード “reducecm. c”, “rgb2cm.c” および
 “cm2cm. c” の関数 (宣言は “cmimage. h” で行
 われている)

関数名	説明と定義
RGB2CM	RGB カラーの画像をパレットカラーとする。 処理に失敗すると NULL を、成功すると新しい CM 構造体へのポインタを返す。 定義： CM *RGB2CM(image) Image *image; ... Image 構造体へのポ インタ
CM2CM	パレットカラー画像の冗長なカラーパレットを除去する。 処理に失敗すると NULL を、成功するともとの CM 構造体が消去されて新しいものを返す。 定義： CM *CM2CM(image, cm) Image *image; ... Image 構造体へのポ インタ CM *cm; ... もとの CM 構造体へのポ インタ
ReduceCM	パレットカラー画像の色数を削減する。 処理に失敗すると NULL を、成功するともとの CM 構造体が消去されて新しいものを返す。 定義： CM *ReduceCM(image,cm,pixels) Image *image; ... Image 構造体へのポ インタ CM *cm; ... もとの CM 構造体へのポ インタ int pixels; ... 削減後の色数の上限

第5表 TIFF ライブラリ “libtiff. a” およびソースコード
 “readtiff. c”, “writetiff. c”, “copytag. c” の関数
 (宣言は “tiffio. h”, “readtiff. h”, “ writetiff. h”
 で行われている)

関数名	説明と定義
TIFFOpen	TIFF ファイルをオープンする。 使い方は、C 言語の標準関数 fopen とほぼ同じ。 定義： TIFF *TIFFOpen(name, mode) char *name; ... TIFF ファイルの名前 char *mode; ... ファイルの入出力モード
TIFFClose	TIFF ファイルをクローズする。 使い方は、C 言語の標準関数 fclose とほぼ同じ。 定義： void TIFFClose(tiff) TIFF *tiff; ... TIFF ファイルポインタ
ReadTIFF	TIFF ファイルの画像データを読み込む。 処理に失敗するとエラーコード (正の整数) を、成功すると 0 を返す。 定義： int ReadTIFF(tiff, image, cm) TIFF *tiff; ... TIFF ファイルポインタ Image **image; ... Image 構造体へのポ インタへのポインタ CM **cm; ... CM 構造体へのポインタへの ポインタ 注意： カラーパレットが存在しない場合は *cm = NULL となる。
ReadTIFF- Error	関数 ReadTIFF で返されたエラーコードのメッ セージを返す。 定義： char *ReadTIFFError(code) int code; ... ReadTIFF で返されたエラー コード
WriteTIFF	TIFF ファイルを作成し、画像データを書き込む。 処理に失敗するとエラーコード (正の整数) を、成功すると 0 を返す。 定義： int WriteTIFF(tiff, image, cm) TIFF *tiff; ... TIFF ファイルポインタ Image *image; ... Image 構造体へのポインタ CM *cm; ... CM 構造体へのポインタ 注意： カラーパレットが存在しない場合は cm = NULL とする。
WriteTIFF- Error	関数 WriteTIFF で返されたエラーコードのメッ セージを返す。 定義： char *WriteTIFFError(code) int code; ... WriteTIFF で返されたエ ラーコード
CopyTAG	TIFF ファイルのすべての TAG をコピーする。 定義： void CopyTAG(org, new) TIFF *org; ... もとの TIFF ファイルポ インタ TIFF *new; ... 新しい TIFF ファイルポ インタ

現されている（第1図参照）。関数 **RGB2CM** は、引き数として渡された構造体 **Image** 中の **RGB** カラー画像をパレットカラー画像に変換する。そして、各画素にピクセル値としてパレット番号を入れた画像を同じ構造体 **Image** に、また、カラーパレット表を新たな構造体 **CM** に格納する。関数 **CM2CM** は、引き数として渡された構造体 **CM** 中のカラーパレット表を引き数の構造体 **Image** を用いてチェックし、冗長な色情報を削除したカラーパレット表を新しい構造体 **CM** として返す。関数 **ReduceCM** は、引き数で指定した個数以下のセルに **RGB** 空間を分割して（引き数として渡された構造体 **Image** と **CM** におさめられた）パレットカラー画像中の色数を削減する。そして、得られたパレットカラー表を新しい構造体 **CM** として返す（構造体 **Image** 中の画像もそれに合うように変換される）。なお、関数 **ReduceCM** では、削減後の色の個数が指定した個数そのものにはならない場合が多いことに注意してほしい。これは、**RGB** 空間を8分割する前に分割後のセルの総数を知ることができないので、その繰り返しを事前に終了するようにしているためである。

これらの構造体や関数で取り扱う画像（**RGB** カラーとパレットカラー画像）は、色の各要素の容量に注意すれば任意のものでよい。すなわち、関数 **AllocImage** や **AllocCM** で構造体 **Image** や **CM** を作成して、それらの各要素に色成分の値をおさめて用いることができる。ところで、ファイル化されている画像データを読み込んで処理したり、また、処理結果をファイルに書き込むことも多いので、そのために使用する関数も付録に示したソフトウェアライブラリに用意した。画像データファイルの形式には様々なものがあるが、ここでは **Rev. 5.0** の **TIFF** 画像データファイルを取り扱うものだけを示した（第5表）。

3－2. TIFF 画像ファイル

TIFF (**Tag Image File Format**) は、**Aldus and Microsoft** (1988) がその仕様を公開している画像データファイルの形式で、多くの市販およびパブリックドメインソフトウェアで取り扱うことができる。**TIFF** 画像ファイルでは、**RGB** カラーやパレットカラー画像はもちろんのこと、それ以外にもグレースケールや白黒2値画像などをおさめることができる。また、様々な方法で画像データの圧縮を行うことや画像データ作成時の諸情報をファイルに付随させることも可能である。画像に付随したこのような情報を **TIFF** ではタグ (**Tag**) と呼んでいる。**TIFF** のおもなタグの一覧を第6表に示した。

第6表のタグを参照したり、また、画像データを読み書きするためのライブラリソフトウェアは、パブリックドメインソフトウェアとして広く流通している（これらは

第6表 TIFF 画像ファイルのおもなタグ

タグ名	格納してある情報
ImageLength	画像の縦幅
ImageWidth	画像の横幅
SamplesPerPixel	1画素あたりのサンプル数
BitsPerSample	サンプルあたりのビット数
PlanarConfiguration	画像データのレイアウト (画像プレーンの配置)
PhotometricInterpretation	画像の表示時の黒白・カラーモデルの種類
ColorMap	カラーパレット (RGB マップ) データ
ColorResponseCurve	カラー (RGB 値) の応答曲線
GrayResponseCurve	グレースケールの応答曲線
ResolutionUnit	画像の解像度の単位
XResolution	画像の横方向の解像度の単位
YResolution	画像の縦方向の解像度の単位
RawPerStrip	入出力時のデータストリップあたりのライン数
StripByteCounts	データストリップの長さ (バイト数)
StripOffset	データストリップのファイル中での位置
Compression	画像データ圧縮の手法
ImageDescription	画像に関する説明・記述
Artist	画像作成者の名前
DateTime	画像作成の日時
HostComputer	画像作成に用いたホストコンピュータ名
Make	画像作成に用いたスキャナ等のメーカー名
Model	画像作成に用いたスキャナ等のモデル名
Software	画像作成に用いたソフトウェアの情報

JUNET などのネットワークを通して配布されている)。本文のソフトウェアでは、**Leffler** (1990) の作成した **TIFF** サポートソフトウェア中に含まれていた **C** 言語用の **TIFF** ライブラリ (“**libtiff. a**”) を利用した。このライブラリでは、関数 **TIFFOpen** (第5表) で初期化したポインタ **TIFF** (第2表) を用いて、ファイル上の画像データやタグにアクセスする。その詳細については付録のソースリストや **Leffler** (1990) の説明書などを参照していただきたい。

TIFF 画像ファイルから画像データやカラーパレットのデータを構造体 **Image** や **CM** に読み込む関数 **ReadTIFF** と、それらを **TIFF** 画像ファイルに書き込む関数 **WriteTIFF** を作成した (第5表)。これらは、関数 **TIFFOpen** で初期化した **TIFF** ファイルポインタを引き数とする。関数 **ReadTIFF** では、ファイルとして格納されている画像がグレースケールもしくは白黒2値画像である場合にそれらをパレットカラー画像であるとして取り扱う（この場合には各色の成分は同じ値を持つ）。また、ファイルに **RGB** カラー画像が格納されている場合には、この関数は構造体 **CM** を作成しない（返される構造体 **CM** へのポインタが **NULL** となる）。関数 **WriteTIFF** では、**RGB** カラー画像をファイルとして格納する場合に、引き数として引き渡す構造体 **CM** のポインタに **NULL** を指定する。これらの関数内で発生したエラーに関する情報は、関数の返す整数値で表現される。そして、この値は、関数 **ReadTIFFError** や **WriteTIFFError** を用いてエラーメッセージに翻訳することができる (第5表参照)。

第6表に示したように、TIFFのタグには画像の縦横サイズのようなファイルの読み書きに必要な基本的な情報と、それ以外の画像の説明のような補助的な情報がどちらもおさめられている。関数 `ReadTIFF` や `WriteTIFF` を用いれば、前者の情報はきちんと取り扱うことができる。ところで、TIFF 画像ファイルから読み込んだ画像に何らかの処理を加えて、その結果を再び TIFF 画像ファイルとして格納するとき、後者の情報をそのままコピーしたい場合がしばしばある。第5表に示した関数 `CopyTAG` は、この用途に用いることができる。すなわち、この関数では、引き数で TIFF ポインタとして与えられたもとの TIFF 画像ファイルのすべてのタグを (TIFF ポインタとして与えられた) 新しい TIFF 画像ファイルにとりあえずコピーする。その後、関数 `WriteTIFF` を呼び出せば、基本的な情報をおさめたタグはそこで指定した画像に合ったものに上書きされる。

3-3. `tiffreduce`

上で説明したソフトウェアライブラリの利用例として、既存の TIFF 画像ファイルを読み込んで、その画像中の色数を指定した個数以下に削減した画像を新たな TIFF 画像ファイルとして書き込む UNIX コマンド `tiffreduce` を作成した。そして、このコマンドとソフトウェアライブラリのコンパイルの仕方を、付録に示した “`Makefile`” に記述した (ただし、この “`Makefile`” には、`tiffreduce` と同様にして色数を削減した後、画像データを LZW 法 (Lempel-Ziv & Welch 法; Welch, 1984) で圧縮してファイルに格納するようにライブラリに指定したコマンド `tiffreduceZ` のコンパイル法も記述してある)。

UNIX コマンド `tiffreduce` は以下のようにして起動する (`tiffreduceZ` の起動法もまったく同じである)。

```
tiffreduce OrgTIFF colors NewTIFF
```

ここで、パラメータ `OrgTIFF` と `NewTIFF` は、それぞれ、もとの TIFF 画像ファイルと色数削減後の TIFF 画像ファイルの名前である。また、`colors` は、削減後の色数の上限 (自然数) である。パラメータ (ファイル名など) の指定の誤り以外で `tiffreduce` がエラーを起こすのは、おそらく、十分なメモリが確保できなかったときだけであると思われる。

4. 色数削減の例

本文の色数削減のアルゴリズムの使用例として、第4図に地質調査所で準備中の100万分の1日本地質図第3版 CD-ROM 版 (1994) のためのテストデータから作成した

地形陰影つき地質図画像への適用結果を示した (なお、この画像の詳細については、中野 (1994) の説明書を参照していただきたい)。

第4図の3枚はどれも 1590×960 画素 (東西4度、南北2度) からなる陰影つき地質図画像である。第4図 (a) はオリジナルのフルカラー画像で、4992色が用いられている。また、第4図 (b) と (c) は、コマンド `tiffreduce` を用いて、それぞれ色数を250色 (`tiffreduce` で256色以下に削減を指定)、120色 (128色以下を指定) に削減した結果の画像である。これらの図からも明らかなように、本文で示したアルゴリズムは非常にうまく色数の削減を行っている。

より詳しく見ると、250色に色数を削減した第4図 (b) では第4図 (a) のもとの画像に比べて、全体の色合いがやや変化している。さらに、120色にした第4図 (c) では、九州九重山周辺の火山岩の色合いが明らかにおかしい。もとの画像の情報量 (第(2)式) を調べてみるとおよそ5.54ビットであるため、原理的には6ビット (64色) 程度に色数を削減しても画像の平均情報量は保持されるはずである。しかしながら、先にも述べた RGB 空間の分割の冗長さのため、本文のアルゴリズムではその倍以上の色数にしか削減できないようである。

なお、これらの色数削減処理に要した時間はどちらも、SUN SPARC station 2 でおよそ100秒程度であった。その処理時間の大部分は RGB カラー画像をパレットカラー画像に変換する処理 (第1図) に費やされていた。この処理では、入力した RGB カラー画像をパレットカラー画像に変換する際に Quick Sort のアルゴリズムで色のチェックとパレット番号のつけ直しを行っているので、画像の総画素数を N としておよそ $N \times \log_2 N$ に比例する時間が必要である (たとえば、画像の縦横の画素数がどちらも半分 (795×480 画素) になると、処理に要する時間はおよそ4分の1 (～30秒) になる)。

5. おわりに

市販もしくはパブリックドメインのソフトウェアでは、画像を表示する際にディスプレイハードウェアに合わせて色数を削減する処理を行っているものも多い。これらで行われている処理法は様々だが (たとえば, Heckbert, 1982), 本文で説明したものと同様な考え方をもとにしているようである (本文のアルゴリズムはそれらとは独立に作成されたものであるが)。本文のソフトウェアライブラリを画像表示の際に用いると、総画素数の多い画像に対してはやや長すぎる処理時間を要する (中野, 1994)。コーディングの際にアルゴリズム全体 (第1図) で重複する処理の無駄を省けばそれをもっと短くできると思われるので、今後改

第4図 色数削減の例. (a) もとの画像（使用されている色は4992色）, (b) 色数を250色に削減した画像, (c) 色数を120色に削減した画像.

良していきたい。

なお、付録に示したソフトウェアのソースリストはすべて、地質調査所で出版予定の CD-ROM (地質調査所, 1994) におさめる予定である。また、ネットワークを通して配布することも可能なので、これらを必要とされる方は御連絡いただきたい。

文 献

- Aldus and Microsoft (1988) Tag Image File Format specification - Reversion 5.0, An Aldus / Microsoft technical Memorandum.
- Harrington, S. (1987) Computer graphics - A programming approach, 2nd ed., McGraw-Hill (邦訳, アルゴリズムとプログラムによるコンピュータグラフィックス (I) (II), マグロウヒル).
- Heckbert, P. (1982) Color image quantization for frame buffer display, SIGGRAPH proceedings, 297 - 307.
- 堀部 安一 (1989) 情報エントロピー論, 森北出版.
- 稲崎 富士・中野 司 (1993) 地質試料解析のための X 線 C T 画像データ処理システム, 情報地質, 4, 9 - 23.
- 石畑 清 (1989) アルゴリズムとデータ構造, 岩波講座ソフトウェア科学, 3, 岩波書店.
- 金子 隆芳 (1988) 色彩の科学, 岩波新書, 岩波書店.
- Leffler, S. (1990) Version 2.3 public domain TIFF support softwares.
- 中野 司 (1992) 2 次元データの処理システム：(4) Z バッファを使った三次元立体図, 情報地質, 3, 65 - 82.
- 中野 司 (1994) 地質図画像データベース, 文書ファイル “TIFFDOC. JP”, 100 万分の 1 日本地質図第 3 版 CD-ROM 版, 地質調査所, 準備中.
- 地質調査所 (1994) 100 万分の 1 日本地質図第 3 版 CD-ROM 版, 準備中.
- Welch, T. A. (1984) A technique for high performance data compression, IEEE Computer, 17, 8 - 19.

要旨：

画像処理のためのソフトウェア：(7)色数削減のアルゴリズム

：中野 司

多数の色を用いた画像から、それに含まれる情報量をできるだけ残して色数を削減するアルゴリズムについて説明する。このアルゴリズムでは、色の出現頻度を一様に近づけるように色の分布する 3 次元空間 (RGB 空間) の分割を行って色数を削減する。その実現として作成した、RGB カラーとパレットカラー画像の色数を削減するためのソフトウェアライブラリについて説明する。また、その応用として作成した、UNIX 上で TIFF 画像ファイルの色数削減の処理を行うコマンド “tiffreduce” を紹介する。

キーワード：画像処理, 色数削減, RGB カラー, パレットカラー, TIFF

(1994年 8 月29日 受付, 1994年10月31日 受理)

付録：ソフトウェアのソースリスト

Makefile :

```

1: #
2: TIFFI    =-I../libtiff
3: TIFFL    =../libtiff/libtiff.a
4:
5: tiffreduce:    tiffreduce.c readtiff.h readtiff.c \
6:                writetiff.h writetiff.c copytag.c \
7:                cmimage.h cmimage.c reducecm.c cm2cm.c rgb2cm.c
8:    cc $(TIFFI) -o tiffreduce \
9:        tiffreduce.c readtiff.c writetiff.c copytag.c \
10:        cmimage.c reducecm.c cm2cm.c rgb2cm.c $(TIFFL)
11:    rm tiffreduce.o readtiff.o writetiff.o copytag.o \
12:        cmimage.o reducecm.o cm2cm.o rgb2cm.o
13:
14: tiffreduceZ:    tiffreduce.c readtiff.h readtiff.c \
15:                writetiff.h writetiff.c copytag.c \
16:                cmimage.h cmimage.c reducecm.c cm2cm.c rgb2cm.c
17:    cc $(TIFFI) -DLZW -o tiffreduceZ \
18:        tiffreduce.c readtiff.c writetiff.c copytag.c \
19:        cmimage.c reducecm.c cm2cm.c rgb2cm.c $(TIFFL)
20:    rm tiffreduce.o readtiff.o writetiff.o copytag.o \
21:        cmimage.o reducecm.o cm2cm.o rgb2cm.o

```

tiffreduce.c :

```

1:
2: #include <stdio.h>
3: #include "tiffio.h"
4: #include "cmimage.h"
5: #include "readtiff.h"
6: #include "writetiff.h"
7:
8: #define STRLEN  256
9:
10: static void      Error(msg)
11: char             *msg;
12: {
13:     fputs(msg,stderr); fputc('\n',stderr); exit(1);
14: }
15:
16: int              main(argc,argv)
17: int              argc;
18: char             **argv;
19: {
20:     int          colors,code;
21:     TIFF         *org,*new;
22:     Image        *image;
23:     CM           *cm;
24:     char         str[STRLEN];
25:
26:     if (argc!=4) Error("usage : tiffreduce orgTIFF colors.newTIFF");
27:
28:     if ((colors=atoi(argv[2]))<2) Error("bad reduction colors.");
29:
30:     if ((org=TIFFOpen(argv[1],"r"))==NULL) Error("TIFF file not found.");
31:
32:     if ((code=ReadTIFF(org,&image,&cm))>0) {
33:         sprintf(str,"TIFF %s",ReadTIFFError(code)); Error(str);
34:     }
35:
36:     if (cm==NULL) {
37:         if ((cm=RGB2CM(image))==NULL) Error("error in RGB2CM.");
38:     }
39:     else
40:         if ((cm=CM2CM(image,cm))==NULL) Error("error in CM2CM.");
41:
42:     if ((cm=ReduceCM(image,cm,colors))==NULL) Error("error in ReduceCM.");
43:
44:     if ((new=TIFFOpen(argv[3],"w"))==NULL) Error("TIFF file not created.");

```

```

45:
46:     CopyTAG(org,new);
47:
48: #ifdef LZW
49:     (void)TIFFSetField(new,TIFFTAG_COMPRESSION,COMPRESSION_LZW);
50: {
51:     int      tss=TIFFScanlineSize(new);
52:
53:     (void)TIFFSetField(new,TIFFTAG_ROWSPERSTRIP,
54:                         (tss>8192)?1L:(long)(8192/tss));
55: }
56: #endif
57:
58:     if ((code=WriteTIFF(new,image,cm))>0) {
59:         sprintf(str,"TIFF %s",WriteTIFFError(code)); Error(str);
60:     }
61:
62:     TIFFClose(new); TIFFClose(org); return(0);
63: }

```

cmimage.h :

```

1:
2: typedef unsigned char  byte;
3: typedef unsigned short word;
4:
5: typedef struct {
6:     int      xSize,ySize;
7:     byte     **r,**g,**b;
8: } Image;
9:
10: typedef struct {
11:     int      pixels;
12:     word     *r,*g,*b;
13: } CM;
14:
15: extern Image      *AllocImage(/* int xSize, int ySize */);
16: extern void       FreeImage(/* Image *image */);
17:
18: extern CM         *AllocCM(/* int pixels */),
19:                  *RGB2CM(/* Image *image */),
20:                  *CM2CM(/* Image *image, CM *cm */),
21:                  *ReduceCM(/* Image *image, CM *cm, int pixels */);
22: extern void       FreeCM(/* CM *cm */);
23:
24: #define PUTPIXEL(image,x,y,pixel)      {\
25:     image->r[x][y]=(byte)((pixel & 0xff);\
26:     image->g[x][y]=(byte)((pixel>> 8)&0xff);\
27:     image->b[x][y]=(byte)((pixel>>16)&0xff);\
28: }
29: #define GETPIXEL(image,x,y)            ((int)image->r[x][y] | \
30:                                         ((int)image->g[x][y]<< 8) | \
31:                                         ((int)image->b[x][y]<<16))

```

cmimage.c :

```

1:
2: #include <stdio.h>
3: #include "cmimage.h"
4:
5: #define R      (image->r)
6: #define G      (image->g)
7: #define B      (image->b)
8: #define XSIZE  (image->xSize)
9: #define YSIZE  (image->ySize)
10:
11: Image      *AllocImage(xSize,ySize)
12: int      xSize,ySize;
13: {
14:     Image      *image=(Image *)malloc(sizeof(*image));

```



```

15:     int      x;
16:
17:     if (image==NULL) return(NULL);
18:
19:     if ((R=(byte **)calloc(xSize,sizeof(*R)))==NULL ||
20:         (G=(byte **)calloc(xSize,sizeof(*G)))==NULL ||
21:         (B=(byte **)calloc(xSize,sizeof(*B)))==NULL) return(NULL);
22:
23:     for (x=0; x<xSize; x++)
24:         if ((R[x]=(byte *)calloc(ySize,sizeof(**R)))==NULL ||
25:             (G[x]=(byte *)calloc(ySize,sizeof(**G)))==NULL ||
26:             (B[x]=(byte *)calloc(ySize,sizeof(**B)))==NULL) return(NULL);
27:
28:     XSIZE=xSize; YSIZE=ySize; return(image);
29: }
30:
31: void FreeImage(image)
32: Image *image;
33: {
34:     int      x;
35:
36:     for (x=XSIZE-1; x>=0; x--) {
37:         free(B[x]); free(G[x]); free(R[x]);
38:     }
39:     free(B); free(G); free(R); free(image);
40: }
41:
42: #define PIXELS (cm->pixels)
43: #define CMR (cm->r)
44: #define CMG (cm->g)
45: #define CMB (cm->b)
46:
47: CM *AllocCM(pixels)
48: int pixels;
49: {
50:     CM      *cm=(CM *)malloc(sizeof(*cm));
51:
52:     if (cm==NULL) return(NULL);
53:
54:     if ((CMR=(word *)calloc(pixels,sizeof(*CMR)))==NULL ||
55:         (CMG=(word *)calloc(pixels,sizeof(*CMG)))==NULL ||
56:         (CMB=(word *)calloc(pixels,sizeof(*CMB)))==NULL) return(NULL);
57:
58:     PIXELS=pixels; return(cm);
59: }
60:
61: void FreeCM(cm)
62: CM *cm;
63: {
64:     free(CMB); free(CMG); free(CMR); free(cm);
65: }

```

rgb2cm.c :

```

1:
2: #include <stdio.h>
3: #include "cmimage.h"
4:
5: typedef struct {
6:     word      x,y;
7:     int       pixel;
8: } XYPixel;
9:
10: static XYPixel *xyP;
11:
12: static void QSort(lower,upper)
13: int lower,upper;
14: {
15:     int i,j,pixel;
16:     XYPixel tmp;

```

```

17:
18:     if (lower>=upper) return;
19:
20:     pixel=xyP[(lower+upper)/2].pixel; i=lower; j=upper;
21:     do {
22:         while (xyP[i].pixel<pixel) ++i;
23:         while (xyP[j].pixel>pixel) --j;
24:         if (i<=j) {
25:             if (i<j) { tmp=xyP[i]; xyP[i]=xyP[j]; xyP[j]=tmp; }
26:             ++i; --j;
27:         }
28:     } while (i<=j);
29:
30:     QSort(lower,j);
31:     QSort(i,upper);
32: }
33:
34: #define B2W(rgb)          (((word)rgb<<8)|(word)rgb)
35:
36: CM      *RGB2CM(image)
37: Image   *image;
38: {
39:     int      size=image->xSize*image->ySize,
40:             x,y,pixel,pixels,idx;
41:     XYPixel *ptr;
42:     CM       *cm;
43:     byte     rgb;
44:
45:     if ((xyP=(XYPixel *)calloc(size,sizeof(*xyP)))==NULL) return(NULL);
46:
47:     ptr=xyP;
48:     for (x=0; x<image->xSize; x++)
49:     for (y=0; y<image->ySize; y++) {
50:         ptr->x=(word)x;
51:         ptr->y=(word)y; ptr->pixel=GETPIXEL(image,x,y); ++ptr;
52:     }
53:     QSort(0,size-1);
54:
55:     ptr=xyP; pixel=ptr->pixel; ptr->pixel=pixels=0;
56:     for (++ptr, idx=1; idx<size; idx++, ptr++) {
57:         if (ptr->pixel!=pixel) { pixel=ptr->pixel; ++pixels; }
58:         ptr->pixel=pixels;
59:     }
60:     if ((cm=AllocCM(++pixels))==NULL) return(NULL);
61:
62:     pixel=(-1);
63:     for (ptr=xyP, idx=0; idx<size; idx++, ptr++) {
64:         if (ptr->pixel!=pixel) {
65:             pixel=ptr->pixel;
66:             rgb=image->r[ptr->x][ptr->y]; cm->r[pixel]=B2W(rgb);
67:             rgb=image->g[ptr->x][ptr->y]; cm->g[pixel]=B2W(rgb);
68:             rgb=image->b[ptr->x][ptr->y]; cm->b[pixel]=B2W(rgb);
69:         }
70:         PUTPIXEL(image,ptr->x,ptr->y,pixel);
71:     }
72:
73:     free(xyP); return(cm);
74: }

```

cm2cm.c :

```

1:
2: #include <stdio.h>
3: #include "cmimage.h"
4:
5: static int      *lut;
6: static word     *cmR,*cmG,*cmB;
7:
8: static void     QSort(lower,upper)
9: int             lower,upper;

```

```

10: {
11:     int    tmp,r,g,b,i,j,dif;
12:
13:     if (lower>=upper) return;
14:
15:     tmp=lut[(lower+upper)/2]; r=(int)cmR[tmp];
16:                               g=(int)cmG[tmp];
17:                               b=(int)cmB[tmp]; i=lower; j=upper;
18:     do {
19:         for (;;) { tmp=lut[i];
20:             if ((dif=(int)cmB[tmp]-b)<0) continue; if (dif>0) break;
21:             if ((dif=(int)cmG[tmp]-g)<0) continue; if (dif>0) break;
22:             if ((dif=(int)cmR[tmp]-r)<0) continue;          break;
23:         }
24:         for (;;) { tmp=lut[j];
25:             if ((dif=(int)cmB[tmp]-b)>0) continue; if (dif<0) break;
26:             if ((dif=(int)cmG[tmp]-g)>0) continue; if (dif<0) break;
27:             if ((dif=(int)cmR[tmp]-r)>0) continue;          break;
28:         }
29:         if (i<=j) {
30:             if (i<j) { tmp=lut[i]; lut[i]=lut[j]; lut[j]=tmp; }
31:             ++i; --j;
32:         }
33:     } while (i<=j);
34:
35:     QSort(lower,j);
36:     QSort(i,upper);
37: }
38:
39: CM    *CM2CM(image,cm)
40: Image *image;
41: CM    *cm;
42: {
43:     int    pixel,x,y,pixels,*ptr,
44:            *pxl=(int *)calloc(cm->pixels,sizeof(*pxl));
45:     CM    *tmp;
46:     word   r,g,b;
47:
48:     if (pxl==NULL ||
49:         (lut=(int *)calloc(cm->pixels,sizeof(*lut)))==NULL) return(NULL);
50:
51:     for (ptr=lut, pixel=0; pixel<cm->pixels; pixel++, ptr++) *ptr=(-1);
52:
53:     for (x=0; x<image->xSize; x++)
54:         for (y=0; y<image->ySize; y++) {
55:             pixel=GETPIXEL(image,x,y); lut[pixel]=0;
56:         }
57:     pixels=0;
58:     for (ptr=lut, pixel=0; pixel<cm->pixels; pixel++, ptr++)
59:         if (*ptr==0) { *ptr=pixels; ++pixels; }
60:
61:     if (pixels<cm->pixels) {
62:         if ((tmp=AllocCM(pixels))==NULL) return(NULL);
63:
64:         for (ptr=lut, pixel=0; pixel<cm->pixels; pixel++, ptr++)
65:             if (*ptr>=0) {
66:                 tmp->r[*ptr]=cm->r[pixel];
67:                 tmp->g[*ptr]=cm->g[pixel];
68:                 tmp->b[*ptr]=cm->b[pixel];
69:             }
70:         FreeCM(cm); cm=tmp;
71:         for (x=0; x<image->xSize; x++)
72:             for (y=0; y<image->ySize; y++) {
73:                 pixel=lut[GETPIXEL(image,x,y)]; PUTPIXEL(image,x,y,pixel);
74:             }
75:     }
76:
77:     for (ptr=lut, pixel=0; pixel<cm->pixels; pixel++, ptr++) *ptr=pixel;
78:     cmR=cm->r;
79:     cmG=cm->g;

```

```

80:         cmB=cm->b; QSort(0,cm->pixels-1);
81:
82:         ptr=lut; r=cmR[*ptr];
83:         g=cmG[*ptr];
84:         b=cmB[*ptr]; pxl[*ptr]=pixels=0;
85:         for (++ptr, pixel=1; pixel<cm->pixels; pixel++, ptr++) {
86:             if (cmR[*ptr]!=r ||
87:                 cmG[*ptr]!=g ||
88:                 cmB[*ptr]!=b) { r=cmR[*ptr];
89:                                 g=cmG[*ptr];
90:                                 b=cmB[*ptr]; ++pixels; }
91:             pxl[*ptr]=pixels;
92:         }
93:
94:         if (++pixels<cm->pixels) {
95:             if ((tmp=AllocCM(pixels))==NULL) return(NULL);
96:
97:             for (ptr=pxl, pixel=0; pixel<cm->pixels; pixel++, ptr++) {
98:                 tmp->r[*ptr]=cm->r[pixel];
99:                 tmp->g[*ptr]=cm->g[pixel];
100:                 tmp->b[*ptr]=cm->b[pixel];
101:             }
102:             FreeCM(cm); cm=tmp;
103:             for (x=0; x<image->xSize; x++)
104:                 for (y=0; y<image->ySize; y++) {
105:                     pixel=pxl[GETPIXEL(image,x,y)]; PUTPIXEL(image,x,y,pixel);
106:                 }
107:         }
108:
109:         free(lut); free(pxl); return(cm);
110: }

```

reducecm.c :

```

1:
2: #include <stdio.h>
3: #include "cmimage.h"
4:
5: typedef struct {
6:     int      number,lo,up,size,r0,g0,b0;
7: } Cell;
8:
9: static Cell      *cell;
10: static int      *pxl,*lut,pixels;
11:
12: static void      Store(lo,up,size,r0,g0,b0)
13: int              lo,up,size,r0,g0,b0;
14: {
15:     Cell      *ptr;
16:     int        pixel;
17:
18:     for (ptr=cell; ptr->number!=0; ptr++) ;
19:
20:     for (pixel=lo; pixel<up; pixel++) ptr->number+=pxl[lut[pixel]];
21:     if ((ptr->lo=lo)==(ptr->up=up)-1)
22:         ptr->size=1;
23:     else {
24:         ptr->size=size; ptr->r0=r0; ptr->g0=g0; ptr->b0=b0;
25:     }
26:     ++pixels;
27: }
28:
29: static int      Divide(lo,up,rgb,pivot)
30: int              lo,up;
31: word            *rgb,pivot;
32: {
33:     int      top,idx,tmp;
34:
35:     for (top=lo; top<up; top++) if (rgb[lut[top]]>=pivot) break;
36:     for (idx=top+1; idx<up; idx++)

```



```

37:         if (rgb[lut[idx]]<pivot) {
38:             tmp=lut[top]; lut[top]=lut[idx]; lut[idx]=tmp; ++top;
39:         }
40:     return(top);
41: }
42:
43: CM      *ReduceCM(image,org,limit)
44: Image   *image;
45: CM      *org;
46: int     limit;
47: {
48:     int     pixel,x,y,number,lo,up,size,r0,g0,b0,rP,gP,bP,rC,gC,bC,i,j;
49:     Cell   *ptr;
50:     CM      *new;
51:     double  r,g,b,cnt;
52:
53:     if ((lut=(int *)calloc(org->pixels,sizeof(*lut)))==NULL ||
54:         (pxl=(int *)calloc(org->pixels,sizeof(*pxl)))==NULL ||
55:         (cell=(Cell *)calloc(limit,sizeof(*cell)))==NULL) return(NULL);
56:
57:     for (pixel=0; pixel<org->pixels; pixel++) {
58:         lut[pixel]=pixel; pxl[pixel]=0;
59:     }
60:     for (x=0; x<image->xSize; x++)
61:     for (y=0; y<image->ySize; y++) ++(pxl[GETPIXEL(image,x,y)]);
62:
63:     ptr=cell; ptr->number=image->xSize*image->ySize;
64:     ptr->lo=0; ptr->up=org->pixels;
65:     ptr->size=0x10000; ptr->r0=ptr->g0=ptr->b0=0; pixels=1;
66:     for (++ptr, pixel=1; pixel<limit; pixel++, ptr++) ptr->number=0;
67:
68:     while (pixels<=limit-7) {
69:         number=0;
70:         for (pixel=0; pixel<pixels; pixel++)
71:             if (cell[pixel].number>number && cell[pixel].size>1) {
72:                 ptr=&(cell[pixel]); number=ptr->number;
73:             }
74:         if (number==0) break;
75:
76:         ptr->number=0; lo=ptr->lo; up=ptr->up;
77:         size=ptr->size>>1; rP=(r0=ptr->r0)+size;
78:         gP=(g0=ptr->g0)+size;
79:         bP=(b0=ptr->b0)+size; --pixels;
80:         bC=Divide(lo,up,org->b,(word)bP);
81:         if (bC>lo) {
82:             gC=Divide(lo,bC,org->g,(word)gP);
83:             if (gC>lo) {
84:                 rC=Divide(lo,gC,org->r,(word)rP);
85:                 if (rC>lo) Store(lo,rC,size,r0,g0,b0);
86:                 if (rC<gC) Store(rC,gC,size,rP,g0,b0);
87:             }
88:             if (gC<bC) {
89:                 rC=Divide(gC,bC,org->r,(word)rP);
90:                 if (rC>gC) Store(gC,rC,size,r0,gP,b0);
91:                 if (rC<bC) Store(rC,bC,size,rP,gP,b0);
92:             }
93:         }
94:         if (bC<up) {
95:             gC=Divide(bC,up,org->g,(word)gP);
96:             if (gC>bC) {
97:                 rC=Divide(bC,gC,org->r,(word)rP);
98:                 if (rC>bC) Store(bC,rC,size,r0,g0,bP);
99:                 if (rC<gC) Store(rC,gC,size,rP,g0,bP);
100:            }
101:            if (gC<up) {
102:                rC=Divide(gC,up,org->r,(word)rP);
103:                if (rC>gC) Store(gC,rC,size,r0,gP,bP);
104:                if (rC<up) Store(rC,up,size,rP,gP,bP);
105:            }
106:        }

```

```

107:     }
108:     if ((new=AllocCM(pixels))==NULL) return(NULL);
109:
110:     for (ptr=cell, pixel=0; pixel<pixels; pixel++, ptr++) {
111:         r=g=b=0.0;
112:         for (i=ptr->lo; i<ptr->up; i++) {
113:             j=lut[i]; cnt=(double)pxl[j]; r+=((double)org->r[j]*cnt);
114:             g+=((double)org->g[j]*cnt);
115:             b+=((double)org->b[j]*cnt);
116:             pxl[j]=pixel;
117:         }
118:         cnt=(double)ptr->number; new->r[pixel]=(word)(r/cnt);
119:         new->g[pixel]=(word)(g/cnt);
120:         new->b[pixel]=(word)(b/cnt);
121:     }
122:     for (x=0; x<image->xSize; x++)
123:     for (y=0; y<image->ySize; y++) {
124:         pixel=pxl[GETPIXEL(image,x,y)]; PUTPIXEL(image,x,y,pixel);
125:     }
126:
127:     free(cell); free(pxl); free(lut); FreeCM(org); return(new);
128: }

```

readtiff.h :

```

1:
2: extern int      ReadTIFF(/* TIFF *tiff, Image **image, CM **cm */);
3: extern char     *ReadTIFFError(/* int code */);

```

readtiff.c :

```

1:
2: #include <stdio.h>
3: #include "tiffio.h"
4: #include "cmimage.h"
5: #include "readtiff.h"
6:
7: #define ERROR_NO      11
8: #define ERROR_SL      1
9: #define ERROR_SIZE    2
10: #define ERROR_IMAGE   3
11: #define ERROR_NO_PI   4
12: #define ERROR_PI      5
13: #define ERROR_BPS     6
14: #define ERROR_SPP     7
15: #define ERROR_PC      8
16: #define ERROR_NO_CM   9
17: #define ERROR_CM      10
18:
19: static char          *Error[ERROR_NO]={
20:     "read error.",
21:     /* SL      */    "scanline data allocation error.",
22:     /* SIZE    */    "no image size description.",
23:     /* IMAGE   */    "image data allocation error.",
24:     /* NO_PI   */    "no photometric interpretation.",
25:     /* PI      */    "unacceptable photometric interpretation.",
26:     /* BPS     */    "unacceptable bits per sample.",
27:     /* SPP     */    "unacceptable samples per pixel.",
28:     /* PC      */    "unacceptable planar configuration.",
29:     /* NO_CM   */    "no color map.",
30:     /* CM      */    "color map data allocation error."
31: };
32:
33: char      *ReadTIFFError(code)
34: int       code;
35: {
36:     if (code<1 || code>=ERROR_NO) return(*Error); return(Error[code]);
37: }
38:
39: #define MSB2LSB(bps,ptr,bit,pxl)      {\

```

```

40:         int idx; for (pxl=idx=0; idx<bps; idx++) {\
41:             pxl<=1; if (bit&(*ptr)) pxl|=1;\
42:             if ((bit>=1)==0) { ++ptr; bit=0x80; }\
43:         }\
44:     }
45:
46: static void      RGBContigImage(tiff,image,tsl,bps,spp)
47: TIFF            *tiff;
48: Image           *image;
49: u_char          *tsl;
50: u_short         bps,spp;
51: {
52:     int      y,x,pxl;
53:     u_char   *ptr,bit;
54:
55:     for (y=0; y<image->ySize; y++)
56:         if (TIFFReadScanline(tiff,ptr=tsl,y,0)>=0)
57:             if (bps==8)
58:                 for (x=0; x<image->xSize; x++) {
59:                     image->r[x][y]=(*ptr++);
60:                     image->g[x][y]=(*ptr++);
61:                     image->b[x][y]=(*ptr++);
62:                 }
63:             else
64:                 for (bit=0x80, x=0; x<image->xSize; x++) {
65:                     MSB2LSB(bps,ptr,bit,pxl); image->r[x][y]=(byte)pxl;
66:                     MSB2LSB(bps,ptr,bit,pxl); image->g[x][y]=(byte)pxl;
67:                     MSB2LSB(bps,ptr,bit,pxl); image->b[x][y]=(byte)pxl;
68:                     if (spp==4)
69:                         for (pxl=0; pxl<bps; pxl++)
70:                             if ((bit>=1)==0) { ++ptr; bit=0x80; }
71:                 }
72: }
73:
74: static void      RGBSeparateImage(tiff,image,tsl,bps)
75: TIFF            *tiff;
76: Image           *image;
77: u_char          *tsl;
78: u_short         bps;
79: {
80:     int      sample,y,x,pxl;
81:     byte     **rgb;
82:     u_char   *ptr,bit;
83:
84:     for (sample=0; sample<3; sample++) {
85:         switch(sample) {
86:             case 0 : rgb=image->r; break;
87:             case 1 : rgb=image->g; break;
88:             case 2 : rgb=image->b; break;
89:         }
90:         for (y=0; y<image->ySize; y++)
91:             if (TIFFReadScanline(tiff,ptr=tsl,y,sample)>=0)
92:                 if (bps==8)
93:                     for (x=0; x<image->xSize; x++) rgb[x][y]=(*ptr++);
94:                 else
95:                     for (bit=0x80, x=0; x<image->xSize; x++) {
96:                         MSB2LSB(bps,ptr,bit,pxl); rgb[x][y]=(byte)pxl;
97:                     }
98:         }
99: }
100:
101: static void      CMImage(tiff,image,tsl,bps)
102: TIFF            *tiff;
103: Image           *image;
104: u_char          *tsl;
105: u_short         bps;
106: {
107:     int      y,x,pxl;
108:     u_char   *ptr,bit;
109:

```

```

110:         for (y=0; y<image->ySize; y++)
111:             if (TIFFReadScanline(tiff,ptr=tsl,y,0)>=0)
112:                 if (bps==8)
113:                     for (x=0; x<image->xSize; x++) {
114:                         pxl=(int)(*(ptr++)); PUTPIXEL(image,x,y,pxl);
115:                     }
116:                 else
117:                     for (bit=0x80, x=0; x<image->xSize; x++) {
118:                         MSB2LSB(bps,ptr,bit,pxl); PUTPIXEL(image,x,y,pxl);
119:                     }
120:     }
121:
122: #define TGF      TIFFGetField
123:
124: int      ReadTIFF(tiff,image,cm)
125: TIFF     *tiff;
126: Image    **image;
127: CM       **cm;
128: {
129:     u_char *tsl=(u_char *)malloc(TIFFScanlineSize(tiff));
130:     u_short iw,il,pi,*r,*g,*b,
131:             bps=1,
132:             spp=1,
133:             pc=PLANARCONFIG_CONTIG;
134:     int     idx,pxl;
135:
136:     if (tsl==NULL) return(ERROR_SL);
137:
138:     if (!TGF(tiff,TIFFTAG_IMAGEWIDTH, &iw) ||
139:         !TGF(tiff,TIFFTAG_IMAGELENGTH,&il)) return(ERROR_SIZE);
140:
141:     if ((*image=AllocImage((int)iw,(int)il))==NULL) return(ERROR_IMAGE);
142:
143:     if (!TGF(tiff,TIFFTAG_PHOTOMETRIC,&pi)) return(ERROR_NO_PI);
144:
145:     (void)TGF(tiff,TIFFTAG_BITSPERSAMPLE,&bps);
146:     (void)TGF(tiff,TIFFTAG_SAMPLESPERPIXEL,&spp);
147:     (void)TGF(tiff,TIFFTAG_PLANARCONFIG,&pc);
148:
149:     switch(pi) {
150:         case PHOTOMETRIC_RGB :
151:             if (bps>8) return(ERROR_BPS);
152:             if (spp!=3 && spp!=4) return(ERROR_SPP);
153:
154:             *cm=NULL;
155:             switch(pc) {
156:                 case PLANARCONFIG_CONTIG :
157:                     RGBContigImage(tiff,*image,tsl,bps,spp); break;
158:                 case PLANARCONFIG_SEPARATE :
159:                     RGBSeparateImage(tiff,*image,tsl,bps); break;
160:                 default :
161:                     return(ERROR_PC);
162:             }
163:             break;
164:
165:         case PHOTOMETRIC_MINISWHITE :
166:             if (bps>24) return(ERROR_BPS);
167:             if (spp!=1) return(ERROR_SPP);
168:             if (pc!=PLANARCONFIG_CONTIG) return(ERROR_PC);
169:
170:             if ((*cm=AllocCM(1<bps))==NULL) return(ERROR_CM);
171:
172:             pxl=(*cm)->pixels-1;
173:             for (idx=0; idx<=pxl; idx++)
174:                 (*cm)->r[idx]=
175:                 (*cm)->g[idx]=
176:                 (*cm)->b[idx]=(word)(0xffff-(0xffff*idx)/pxl);
177:             CMImage(tiff,*image,tsl,bps);
178:             break;
179:

```

```

180:         case PHOTOMETRIC_MINISBLACK :
181:         case PHOTOMETRIC_MASK :
182:             if (bps>24) return(ERROR_BPS);
183:             if (spp!=1) return(ERROR_SPP);
184:             if (pc!=PLANARCONFIG_CONFIG) return(ERROR_PC);
185:
186:             if ((*cm=AllocCM(1<<bps))==NULL) return(ERROR_CM);
187:
188:             pxl=(*cm)->pixels-1;
189:             for (idx=0; idx<=pxl; idx++)
190:                 (*cm)->r[idx]=
191:                 (*cm)->g[idx]=
192:                 (*cm)->b[idx]=(word)((0xffff*idx)/pxl);
193:             CMImage(tiff,*image,tsl,bps);
194:             break;
195:
196:         case PHOTOMETRIC_PALETTE :
197:             if (bps>24) return(ERROR_BPS);
198:             if (spp!=1) return(ERROR_SPP);
199:             if (pc!=PLANARCONFIG_CONFIG) return(ERROR_PC);
200:
201:             if ((*cm=AllocCM(1<<bps))==NULL) return(ERROR_CM);
202:
203:             if (!TGF(tiff, TIFFTAG_COLORMAP, &r, &g, &b)) return(ERROR_NO_CM);
204:
205:             for (idx=0; idx<(*cm)->pixels; idx++) {
206:                 (*cm)->r[idx]=(word)r[idx];
207:                 (*cm)->g[idx]=(word)g[idx];
208:                 (*cm)->b[idx]=(word)b[idx];
209:             }
210:             CMImage(tiff,*image,tsl,bps);
211:             break;
212:
213:         default :
214:             return(ERROR_PI);
215:     }
216:     free(tsl); return(0);
217: }

```

writetiff.h :

```

1:
2: extern int      WriteTIFF(/* TIFF *tiff, Image *image, CM *cm */);
3: extern char     *WriteTIFFError(/* int code */);

```

writetiff.c :

```

1:
2: #include <stdio.h>
3: #include "tiffio.h"
4: #include "cmimage.h"
5: #include "writetiff.h"
6:
7: #define ERROR_NO      10
8: #define ERROR_SL      1
9: #define ERROR_SIZE    2
10: #define ERROR_PI      3
11: #define ERROR_BPS     4
12: #define ERROR_SPP     5
13: #define ERROR_PC      6
14: #define ERROR_CM      7
15: #define ERROR_NO_CM   8
16: #define ERROR_WS      9
17:
18: static char          *Error[ERROR_NO]={
19:     "write error.",
20:     /* SL      */      "scanline data allocation error.",
21:     /* SIZE    */      "image size description write error.",
22:     /* PI      */      "photometric interpretation write error.",
23:     /* BPS     */      "bits per sample write error.",

```

```

24: /* SPP */      "samples per pixel write error.",
25: /* PC */      "planar configuration write error.",
26: /* CM */      "color map data allocation error.",
27: /* NO_CM */   "color map write error.",
28: /* WS */      "scanline data write error."
29: };
30:
31: char *WriteTIFFError(code)
32: int code;
33: {
34:     if (code<1 || code>=ERROR_NO) return(*Error); return(Error[code]);
35: }
36:
37: static int RGBImage(tiff,image,tsl)
38: TIFF *tiff;
39: Image *image;
40: u_char *tsl;
41: {
42:     int y,x;
43:     u_char *ptr;
44:
45:     for (y=0; y<image->ySize; y++) {
46:         ptr=tsl;
47:         for (x=0; x<image->xSize; x++) {
48:             *(ptr++)=image->r[x][y];
49:             *(ptr++)=image->g[x][y];
50:             *(ptr++)=image->b[x][y];
51:         }
52:         if (TIFFWriteScanline(tiff,tsl,y,0)<0) return(0);
53:     }
54:     return(1);
55: }
56:
57: static int CMImage(tiff,image,tsl,bps)
58: TIFF *tiff;
59: Image *image;
60: u_char *tsl;
61: int bps;
62: {
63:     int y,x,pxl,msk;
64:     u_char *ptr,bit;
65:
66:     for (y=0; y<image->ySize; y++) {
67:         ptr=tsl;
68:         if (bps==8)
69:             for (x=0; x<image->xSize; x++) *(ptr++)=GETPIXEL(image,x,y);
70:         else
71:             for (bit=0x80, x=0; x<image->xSize; x++) {
72:                 pxl=GETPIXEL(image,x,y);
73:                 for (msk=1<<(bps-1); msk!=0; msk>>=1) {
74:                     if (pxl&msk) *ptr|=bit; else *ptr&=(-bit);
75:                     if ((bit>>=1)==0) { ++ptr; bit=0x80; }
76:                 }
77:             }
78:         if (TIFFWriteScanline(tiff,tsl,y,0)<0) return(0);
79:     }
80:     return(1);
81: }
82:
83: #define TSF TIFFSetField
84:
85: int WriteTIFF(tiff,image,cm)
86: TIFF *tiff;
87: Image *image;
88: CM *cm;
89: {
90:     int pxl,spp=3,bps=8;
91:     u_char *tsl;
92:     u_short *r,*g,*b;
93:

```

```

94:         if (cm!=NULL) {
95:             spp=1; for (bps=1; (1<<bps)<cm->pixels; bps++) ;
96:         }
97:
98:         if (!TSF(tiff,TIFFTAG_IMAGEWIDTH, (u_short)image->xSize) ||
99:             !TSF(tiff,TIFFTAG_IMAGELENGTH, (u_short)image->ySize))
100:             return(ERROR_SIZE);
101:
102:         if (!TSF(tiff,TIFFTAG_BITSPERSAMPLE, (u_short)bps)) return(ERROR_BPS);
103:
104:         if (!TSF(tiff,TIFFTAG_SAMPLESPERPIXEL, (u_short)spp)) return(ERROR_SPP);
105:
106:         if (!TSF(tiff,TIFFTAG_PLANARCONFIG, (u_short)PLANARCONFIG_CONTIG))
107:             return(ERROR_PC);
108:
109:         if (cm==NULL) {
110:             if (!TSF(tiff,TIFFTAG_PHOTOMETRIC, (u_short)PHOTOMETRIC_RGB))
111:                 return(ERROR_PI);
112:
113:             if ((tsl=(u_char *)malloc(TIFFScanlineSize(tiff)))==NULL)
114:                 return(ERROR_SL);
115:
116:             if (!RGBImage(tiff,image,tsl)) return(ERROR_WS);
117:         }
118:         else {
119:             if (!TSF(tiff,TIFFTAG_PHOTOMETRIC, (u_short)PHOTOMETRIC_PALETTE))
120:                 return(ERROR_PI);
121:
122:             if ((r=(u_short *)calloc(1<<bps,sizeof(*r)))==NULL ||
123:                 (g=(u_short *)calloc(1<<bps,sizeof(*g)))==NULL ||
124:                 (b=(u_short *)calloc(1<<bps,sizeof(*b)))==NULL)
125:                 return(ERROR_CM);
126:
127:             for (pxl=0; pxl<cm->pixels; pxl++) {
128:                 r[pxl]=cm->r[pxl];
129:                 g[pxl]=cm->g[pxl];
130:                 b[pxl]=cm->b[pxl];
131:             }
132:             if (!TSF(tiff,TIFFTAG_COLORMAP,r,g,b)) return(ERROR_NO_CM);
133:
134:             if ((tsl=(u_char *)malloc(TIFFScanlineSize(tiff)))==NULL)
135:                 return(ERROR_SL);
136:
137:             if (!CMImage(tiff,image,tsl,bps)) return(ERROR_WS);
138:         }
139:         free(tsl); return(0);
140:     }

```

copytag.c :

```

1:
2: #include <stdio.h>
3: #include "tiffio.h"
4:
5: #define TGF      TIFFGetField
6: #define TSF      TIFFSetField
7:
8: #define CF1(t,v1)      if (TGF(org,t,&v1)) TSF(new,t,v1)
9: #define CF2(t,v1,v2)   if (TGF(org,t,&v1,&v2)) TSF(new,t,v1,v2)
10: #define CF3(t,v1,v2,v3) if (TGF(org,t,&v1,&v2,&v3)) TSF(new,t,v1,v2,v3)
11:
12: #define CF6(t,v1,v2,v3,v4,v5,v6) \
13:     if (TGF(org,t,&v1,&v2,&v3,&v4,&v5,&v6)) TSF(new,t,v1,v2,v3,v4,v5,v6)
14:
15: void    CopyTAG(org,new)
16: TIFF    *org,*new;
17: {
18:     char    *cpl;
19:     u_long   ull;
20:     u_short  us1,us2,*usp1,*usp2,*usp3;

```

```

21:         float    f1,f2,f3,f4,f5,f6;
22:
23:         CF1(TIFFTAG_ARTIST,                cp1);
24:         CF1(TIFFTAG_BADFAXLINES,          ul1);
25:         CF1(TIFFTAG_BITSPERSAMPLE,        us1);
26: /*      CF1(TIFFTAG_CELLENGTH,            us1); */
27: /*      CF1(TIFFTAG_CELLWIDTH,           us1); */
28:         CF1(TIFFTAG_CLEANFAXDATA,         us1);
29:         CF3(TIFFTAG_COLORMAP,             usp1,usp2,usp3);
30:         CF3(TIFFTAG_COLORRESPONSECURVE,   usp1,usp2,usp3);
31:         CF1(TIFFTAG_COLORRESPONSEUNIT,    us1);
32:         CF1(TIFFTAG_COMPRESSION,          us1);
33:         CF1(TIFFTAG_CONSECUTIVEBADFAXLINES, ul1);
34:         CF1(TIFFTAG_DATETIME,             cp1);
35:         CF1(TIFFTAG_DOCUMENTNAME,         cp1);
36:         CF1(TIFFTAG_FILLOREDER,          us1);
37: /*      CF1(TIFFTAG_FREEBYTECOUNTS,      ul1); */
38: /*      CF1(TIFFTAG_FREEOFFSETS,          ul1); */
39:         CF1(TIFFTAG_GRAYRESPONSECURVE,    usp1);
40:         CF1(TIFFTAG_GRAYRESPONSEUNIT,     us1);
41:         CF1(TIFFTAG_GROUP3OPTIONS,        ul1);
42:         CF1(TIFFTAG_GROUP4OPTIONS,        ul1);
43:         CF1(TIFFTAG_HOSTCOMPUTER,         cp1);
44:         CF1(TIFFTAG_IMAGEDESCRIPTION,     cp1);
45:         CF1(TIFFTAG_IMAGELENGTH,          us1);
46:         CF1(TIFFTAG_IMAGEWIDTH,           us1);
47:         CF1(TIFFTAG_MAKE,                 cp1);
48:         CF1(TIFFTAG_MATTEING,             us1);
49:         CF1(TIFFTAG_MAXSAMPLEVALUE,       us1);
50:         CF1(TIFFTAG_MINSAMPLEVALUE,       us1);
51:         CF1(TIFFTAG_MODEL,                cp1);
52:         CF1(TIFFTAG_ORIENTATION,          us1);
53: /*      CF1(TIFFTAG_OSUBFILETYPE,         us1); */
54:         CF1(TIFFTAG_PAGENAME,             cp1);
55:         CF2(TIFFTAG_PAGENUMBER,           us1,us2);
56:         CF1(TIFFTAG_PHOTOMETRIC,          us1);
57:         CF1(TIFFTAG_PLANARCONFIG,         us1);
58:         CF1(TIFFTAG_PREDICTOR,            us1);
59: /*      CF6(TIFFTAG_PRIMARYCHROMATICITIES, f1,f2,f3,f4,f5,f6); */
60:         CF1(TIFFTAG_RESOLUTIONUNIT,       us1);
61:         CF1(TIFFTAG_ROWSPERSTRIP,         ul1);
62:         CF1(TIFFTAG_SAMPLESPERPIXEL,      us1);
63:         CF1(TIFFTAG_SOFTWARE,             cp1);
64:         CF1(TIFFTAG_STRIPBYTECOUNTS,     ul1);
65:         CF1(TIFFTAG_STRIPOFFSETS,         ul1);
66:         CF1(TIFFTAG_SUBFILETYPE,          ul1);
67:         CF1(TIFFTAG_THRESHOLDING,         us1);
68: /*      CF2(TIFFTAG_WHITEPOINT,           f1,f2); */
69:         CF1(TIFFTAG_XPOSITION,            f1);
70:         CF1(TIFFTAG_XRESOLUTION,          f1);
71:         CF1(TIFFTAG_YPOSITION,            f1);
72:         CF1(TIFFTAG_YRESOLUTION,          f1);
73: }

```