知能情報処理演習2 — 画像処理プログラミング — レポート

クラス：
担当教員：
学籍番号：
氏名：

実験日:

目 次

1　課題 0. 反転画像の生成

1.1　目的

Portable PixMap(以下 PPM) 形式の 24 ビットカラー画像の原理や扱い方（ファイル入出力，画像処理の基礎）を理解する．また，簡単な画像処理の例として，反転画像の作成方法について理解する．

1.2　内容

PPM 形式の 24 ビットカラー画像をファイルから読み込み，画像中のすべての画素を走査しつつ，各画素値を反転させた画像を生成し，同形式で出力する．

1.2.1　原理

二次元のディジタル画像は，図 1 に示すように，左上のピクセルを原点に，右（i 方向）下（j 方向）へと大きさを持っている．

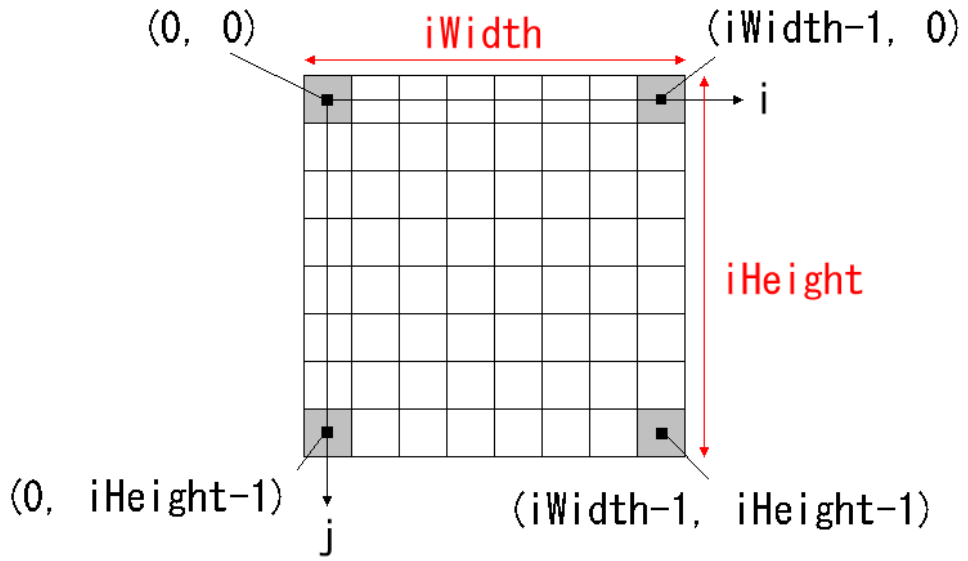

図 1: 二次元画像

ここで，iWidth，iHeight はそれぞれ画像の横と縦のピクセル数である．二次元画像には様々な表現方法があり，一般的に，濃淡（グレースケール）画像は，各画素が 8 ビット（0〜255 の 256 諧調）で表され，カラー画像は 24 ビット（8 ビット ×3．R（赤），G（緑），B（青），

それぞれ 0〜255 の 256 諧調）で表される．また，24 ビットカラー画像を扱うファイル形式には，BMP，PNG，JPEG，GIF，PPM 等，様々な形式が存在し，それぞれ色の記録方法が異なる．本演習では，PPM 形式を利用して 24 ビットカラー画像をファイルから読み込み，各画素の色を反転し，同形式にてファイルに出力する．図 2 は PPM 形式の 24 ビットカラー画像である．カラー画像の各ピクセルは，R（赤），G（緑），B（青）の三色の輝度データを持っている．

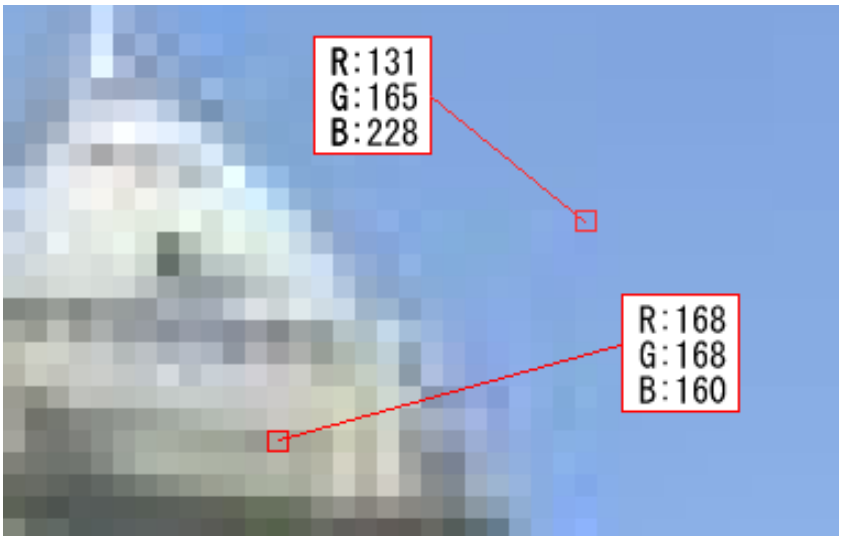

図 2: カラー画像

入力した画像の各画素の色を反転させた画像を作成するためには，画像中の各画素の R,G,B の値を数式 (1) から (3) に示すように，最大階調値の 255 から引いた値に変換すればよい．

$$R_{negative} = 255 - R_{original} \quad (1)$$

$$G_{negative} = 255 - G_{original} \quad (2)$$

$$B_{negative} = 255 - B_{original} \quad (3)$$

ここで，$R_{original}, G_{original}, B_{original}$ は入力画像の輝度値であり，$R_{negative}, G_{negative}, B_{negative}$ が反転画像の輝度値である．この式を使えば，例えば，反転前の画素値が黒色 (0, 0, 0) であれば白色 (255, 255, 255) になり，逆に白色 (255, 255, 255) であれば黒色 (0, 0, 0) になる．この処理を入力した画像のすべての画素に適用すればよい．

また，PPM 形式では，テキストベースで，

86 88 74 85 87 73 86 88 74 87 89 75 90 92 78 …

等のように各画素の R,G,B 値がそれぞれ順番に一次元で 8 ビット（0 から 255 の値）で記述されている．

本演習では，24 ビットカラー画像を PPM 形式で読み込み，図 3 に示すように，各画素の R,G,B 値を以下に示す RGB 構造体を画素の数だけ格納できる一次元のメモリ領域を用いて管理する．

```
// 構造体 RGB 型
struct RGB {
    int iRed;    //赤成分（0～255）
    int iGreen; //緑成分（0～255）
    int iBlue;   //青成分（0～255）
};
```

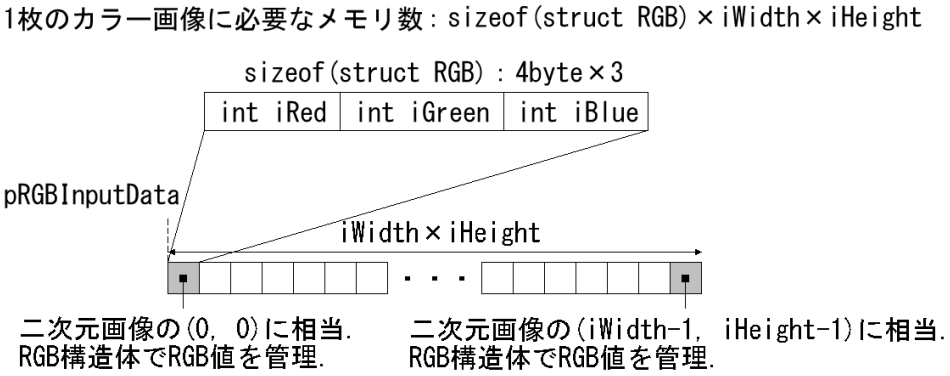

図 3: カラー画像の一次元管理

ここで，iWidth，iHeight は，読み込んだ二次元画像の横と縦のピクセル数である．pRGBInputData は RGB 構造体のポインタ配列である．int 型に必要なメモリ数は sizeof(int)=4byte であるため，RGB 構造体を用いると，1 画素あたりに必要なメモリ数は sizeof(struct RGB)=12byte，1 枚の画像を扱うために必要なメモリ数は（sizeof(struct RGB)×iWidth×iHeight）byte となる．反転画像を格納するためのメモリ数を確保するには，以下のように malloc 関数を用いればよい．

```
struct RGB * pRGBOutputData;        //出力データを格納
// 出力データ（1 次元）のメモリ領域の確保
pRGBOutputData = (struct RGB *)malloc(sizeof(struct RGB)*iWidth*iHeight);
```

すべての画素値を走査するには，一次元の先頭の画素値（0 番目，二次元座標では (0, 0) に相当）から最後尾（(iWidth×iHeight-1）番目，二次元座標では (iWidth-1, iHeight-1）に相当）まで走査すればよい．また，すべての画素値を走査しながら，式 (1)，(2)，(3) を用いて各画素値の値を反転させたものを出力画像として PPM 形式でファイル出力すれば，反転画像を出力できる．

1.2.2　アルゴリズム

図 4 に反転画像生成の処理の流れを示す.

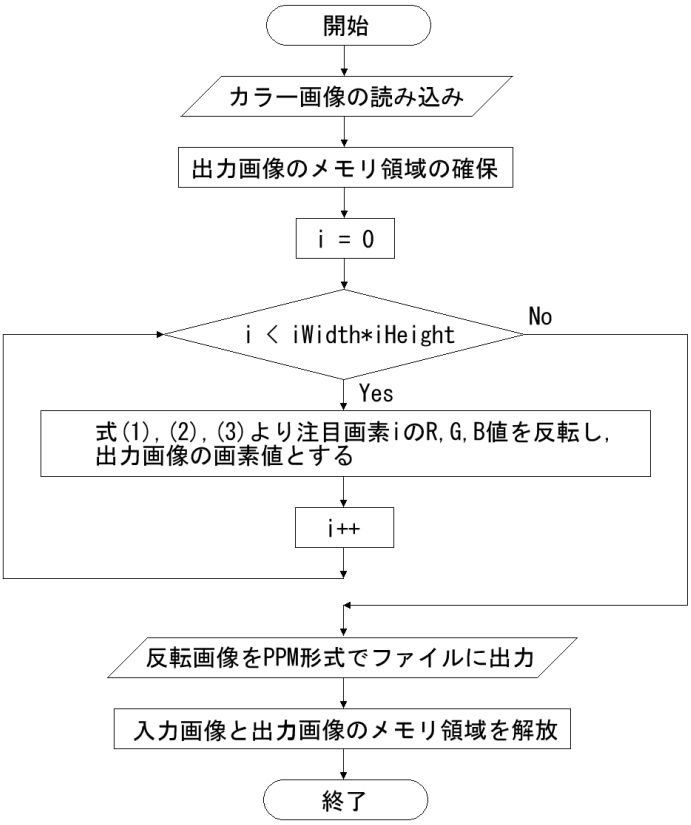

図 4: 反転画像作成のフローチャート

ここで，i は注目画素を表す 1 次元のインデックス，iWidth，iHeight はそれぞれ画像の横と縦の画素数である.

反転画像生成の処理の流れは以下のとおりである.

Step 1. 24 ビットカラー画像（PPM 形式）の読み込み

まず. 反転画像を作成するための元となる画像をファイルから読み込むために，あらかじめライブラリとして用意されている ReadPpm 関数 [1] を用いて 24 ビットカラー画像（PPM 形式）をファイルから読み込む.

Step 2. 出力画像のメモリ領域の確保

次に，作成する反転画像を記憶するために，malloc 関数を用いて RGB 構造体 × 画像サイズ分のメモリ領域を確保する．

Step 3. 全画素数分走査しながら，式 (1),(2),(3) より各画素の R,G,B 値を反転

次に，読み込んだカラー画像の各画素を反転させるため，入力した画像を画素数（iWidth×iHeight）分走査しながら各画素 i の R,G,B 値を式 (1)〜(3) を用いて反転したものを，出力画像の画素値として割り当てる．

Step 4. 反転画像を PPM 形式でファイルに出力

次に，Step 4 では，Step 3 で作成した反転画像（24 ビットカラー画像）をファイルに出力するため，あらかじめライブラリとして用意されている WritePpm 関数 [1] を用いて PPM 形式でファイルに書き込む．

Step 5. 入力画像と出力画像のメモリ領域を解放

最後に，Step 5 では，Step1 と Step2 で確保したメモリ領域を解放するために，あらかじめライブラリとして用意されている FreePpm 関数 [1] を用いて，入力画像と出力画像のメモリ領域を解放する．

1.3 結果

図 5 を反転した結果が図 6 であり, 図 7 を反転した結果が図 8 である．元の画像で明るい部分が反転画像では暗くなっている．これらの結果より, 反転画像が生成できていることがわかる．

図 5: Airport のカラー画像

図 6: Airport の反転画像

図 7: Plane のカラー画像

図 8: Plane の反転画像

1.4 考察

暗い色は明るく，明るい色は暗くなっていることから出力画像は入力画像の補色になっていることが分かる．これは，RGB の値をそれぞれ最大輝度値で引いたため，色相環の反対に位置する色をさすことになったためであると考えられる．

A　課題で作成したソースプログラム

sample1.c

```
1  //--------------------------------------------------------------------//
2  // 例１） //
3  // void NegativeImage(void) //
4  // 内 容 : PPM形式の入力データをネガ画像に変換 //
5  // 更 新 日 : 2004. 2. 11 木村 彰徳 (Computer Graphics Lab.) //
6  // 更 新 日 : 2011. 4. 11 備藤 達郎 (Computer Vision Lab.) //
7  // 更 新 日 : 2013. 9. 27 脇田 航 (Computer Graphics Lab.) //
8  //--------------------------------------------------------------------//
9  void NegativeImage(void) {
10
11     int iWidth, iHeight, iMaxValue; // 画像の幅,高さ,解像度
12     struct RGB * pRGBInputData; // 入力データを格納
13     struct RGB * pRGBOutputData; // 画像処理したデータを格納
14     int i; // ループカウンタ
15
16     // ネガ画像
17     printf("\n****␣Negative␣Image␣*****\n");
18
19     // PPM形式の入力データの読み込み
20     pRGBInputData = ReadPpm(&iWidth, &iHeight, &iMaxValue);
21
22     // 出力データ(１次元)のメモリ領域の確保
23     // pRGBOutputData[iWidth*iHeight]
24     pRGBOutputData = (struct RGB *)malloc(iWidth*iHeight*sizeof(struct RGB));
25
26     // 入力データ (pRGBInputmData)のRGB の値を反転して,
27     // 出力データ (pRGBOutputData)に保存
28     for(i = 0; i < iWidth*iHeight; i++){
29         pRGBOutputData[i].iRed = iMaxValue - pRGBInputData[i].iRed;
30         pRGBOutputData[i].iGreen = iMaxValue - pRGBInputData[i].iGreen;
31         pRGBOutputData[i].iBlue = iMaxValue - pRGBInputData[i].iBlue;
32     }
33
34     // PPM形式のファイルに出力データを出力
35     WritePpm(pRGBOutputData, iWidth, iHeight, iMaxValue);
36
37     // 入力データを保存しているメモリ領域を解放
38     FreePpm(pRGBInputData);
39
40     // 出力データを保存しているメモリ領域を解放
41     FreePpm(pRGBOutputData);
42 }
```

参考文献

[1] 脇田，田中「2013 年度 知能情報処理演習 2 — 画像処理プログラミング —」, 立命館大学情報理工学部知能情報学科, 2013 年.