2017年度 知能情報処理演習２

－ 画像処理プログラミング －

担当：瀬尾 昌孝（D1・D2 クラス，月7～8限）
井本 桂右（D3・D4 クラス，月5～6限）
補助担当：松尾 直志（D1・D2・D3・D4 クラス）

１．目的

画像のパターン認識・理解，コンピュータ・ビジョンなどを志向したデジタル画像処理のための基本アルゴリズムをＣ言語によるプログラミングを通じて習得する．併せてプログラミング用ツール・ソフトウェアの使用法も学ぶ．

２．課題

以下の課題1～5は必須課題である．

課題1：濃淡画像の生成（1週目）
課題2：濃淡画像の輝度値のヒストグラムと2値化（2週目）
課題3：濃淡画像の統計的性質（近接画素間の相関度）抽出（3週目）
課題4：微分フィルタによるエッジ検出（4～5週目）
課題5：パターンの検出（テンプレートマッチング）（6～7週目）

各課題が終わるごとにプログラムの動作チェックを行う．

３．レポート提出

レポートは課題2，4，5が終了した後，計3回提出．指定された日時までに学びステーション及びmanaba+R（PDF，tex，入力画像，出力画像等のすべてのファイルを１つのフォルダにまとめ，zip等で圧縮すること）に提出すること．レポートは原則としてLaTeXで作成すること．レポートのテンプレートは以下からダウンロード可能．

http://www.sys.ci.ritsumei.ac.jp/iipp2g/index.html

やむを得ず Word 等で作成する場合は，サンプルの PDF ファイルを参考に，できる限り書式を合わせること（書式が整っていないレポートは採点の対象外となることがある）．

また，第 1 回目（課題 1，2）と第 2 回目（課題 3，4）のレポートについては 1 度だけ添削の上，不備のあるものは差し戻すので再提出すること．

４．確認テスト

第 3 週目と第 6 週目に各課題の処理内容について，15 分間の筆記による理解度チェックを行う．理解度チェックは各レポート提出後の授業中に行い，不合格の場合は次週に再チェックを受ける(1 回を限度とする)．

５．実技テスト

第 3 週目と第 6 週目に 30 分間の実技テストを行う．各課題のいずれか 1 つについてその場でプログラミングを行ってもらう．

６．その他

各課題を行うにあたって「画像認識・理解」の講義の受講を勧める．また，ソースコードやレポートを他の学生と共有したり，共同制作をしてはならない．悪質なものについては戒告や停学などの懲戒対象にもなりえるので注意すること．

課題の準備

（課題1の授業までに必ず実行しておくこと）

１．サンプルプログラムの取得と解凍

サンプルプログラムは以下のURLからダウンロードすること．

http://www.cv.ci.ritsumei.ac.jp/class/iipp2/iipp2g_program.tgz

以下のようにしてサンプルプログラムを解凍せよ（注“>”は端末エミュレータのコマンド・プロンプトを表す）．

```
> tar zxvf iipp2g_program.tgz
> rm iipp2g_program.tgz
```

カレントディレクトリにiipp2g_program/というディレクトリが出来ていればOK．このディレクトリをカレントディレクトリにせよ．

```
> cd iipp2g_program
```

なお，サンプルプログラムのソースコードは付録５を参照すること．また，本テキストは以下でも参照できる．

http://www.cv.ci.ritsumei.ac.jp/class/iipp2/iipp2g_text.pdf

２．サンプル画像の確認

(2-1) lsコマンドでサンプル画像のファイル名を表示せよ．PGMファイルは濃淡画像（グレースケール画像），PPMファイルはカラー画像である（詳細は付録１参照）．

例）
```
> ls *.pgm
Female.pgm    noisylines.pgm ・・・・・・
> ls *.ppm
Airport.ppm    Bell.ppm    Bike.ppm    Bookshelf.ppm    Building.ppm
Castle.ppm    Characters.ppm    Corridor.ppm ・・・・・・・・
```

(2-2) ImageMagic (displayコマンド．詳細は付録１参照)で上の画像ファイルを表示せよ．

例） > display Castle.ppm

カラー画像（PPM ファイル）

(2-3) emacsまたはless（付録３参照）で上記の画像ファイルの中身を表示せよ．中身はテキスト形式で，数字が並んでいる．付録１を参考にして数字の意味を解釈せよ．

例）

> less Castle.ppm

P3

320 240

255

103 137 198 103 137 198 104 138 199 104 138 199 105 139 200105 139 200 104 138 199 104 138 199 104 138 199 ...

注意

画像の表示が上手く行かない場合，ディスクの割当量を超えて容量を消費していることが原因と思われる．不要なファイルを削除すること．各自のディスク使用量は，以下のようにして確認できる．

> cd (ホームディレクトリへ)

> du –sh

また，サイズの大きなファイルは以下のようにして確認できる．

> du -sk .??* * | sort -n

課題０ サンプルプログラムの実行と解析

（課題１の授業までに必ず実行しておくこと）

課題 0 では，サンプルプログラムを実行，解析する．課題 1 以降はサンプルプログラムを書き換えることで実装できるので, まずはサンプルプログラムをしっかりと読解すること．

サンプルプログラムの main()関数（ファイル main.c に記述されている）は以下のようになっている．

```
int main(void) {
    // 画像処理を行う関数の呼び出し
    // 例１）PPM 形式の入力データから反転画像を生成
    NegativeImage();
    // 例 2）PGM 形式のデータに直線群を描く
    //DrawLines();
    // 正常終了
    return 0;
}
```

デフォルト状態では反転画像生成用の関数 NegativeImage()がアクティブになっている．関数を入れ替えると，直線群描画の関数が呼び出される．

課題０ 反転画像の作成

0.1 実行プログラムの生成

サンプルプログラムには Makefile（付録３参照）が付属している．以下のように make コマンドを使って実行プログラム imgfilter を生成せよ．

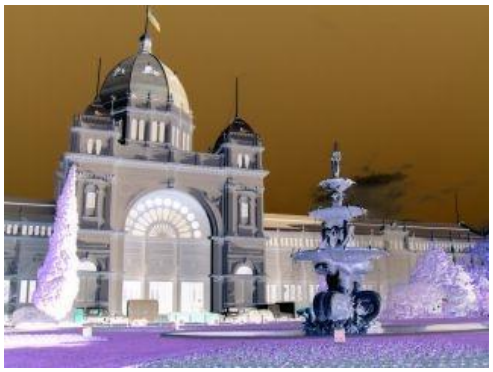
例）

```
> make clean
> make
gcc -g –Wall -c -o ImageFileIO.o ImageFileIO.c
gcc -g -Wall -c -o ImageFilter.o ImageFilter.c
gcc -g -Wall -c -o ImageTools.o ImageTools.c
gcc -g -Wall -c -o main.o main.c
gcc -o imgfilter ImageFileIO.o ImageFilter.o ImageTools.o main.o -lm
```

付録３を参照し，make の機能を確認せよ．また，emacs か less（付録３参照）で Makefile の中身を表示し，記述を確認せよ．

0.2 プログラムの実行

前節で生成した imgfilter を実行し，Castle.ppm の反転画像（右）を作成，表示せよ．



カラー反転画像

```
> ./imgfilter
**** Negative Image *****
Input INPUT PPM file name[*.ppm]? > Castle.ppm
Input OUTPUT PPM file name[*.ppm]? > xxx.ppm
> display xxx.ppm
```

その他の PPM ファイルでも反転画像の作成，表示を試してみよ．

0.3 プログラムの読解

サンプルプログラムを解読し，画像反転の仕組みを理解せよ．

ヒント

画像はピクセル（画素）の集合である．各ピクセルは，輝度や色などの情報を表すひとつ以上の整数データを持っている．濃淡画像の場合，各ピクセルは輝度を表す 0〜255 の数値を持っている．0 は黒，255 は白，その中間の数値はグレーを表す．反転操作により，輝度 0 は 255 に，1 は 254 に，2 は 253 に変換され，以降同様である．カラー画像では各ピクセルは３つの整数データを持つ．それぞれ R（赤），G（緑），B（青）の色成分である．(R,G,B) = (255,0,0) は赤，(0,255,0)は緑，(0,0,255)は青であり，これらを反転させた(0,255,255)はシアン，(255,0,255)はマゼンタ，(255,255,0)は黄となる．

画像処理プログラミング（1 週目）

課題１　濃淡画像の生成

様々な画像処理において，カラー画像を濃淡画像（右）に変換する前処理が行われる（濃淡画像をさらに２値画像に変換することも多い）．色彩情報の記述の方式には RGB 表色系や YIQ 表色系があり，後者において Y は明度を表す（I と Q は色成分を示す）．従って，RGB 表色系で記述されたカラー画像を YIQ 表色系に変換し，Y の値を輝度として用いることで，濃淡画像を生成できる．

濃淡画像

プログラミング課題：

RGB→YIQ の変換式を用いてカラー画像を濃淡画像に変換し，その結果を出力せよ．ファイル ImageFilter.c 内に GrayscaleImage(void)という関数を実装し，main()関数内で呼び出すこと．

※2 つ以上のカラー画像を濃淡画像に変換すること．

ヒント

PPM ファイルを読み込んで Y を計算し，これを輝度値として濃淡画像データを生成して，結果を PGM ファイルに出力する．

記述課題：

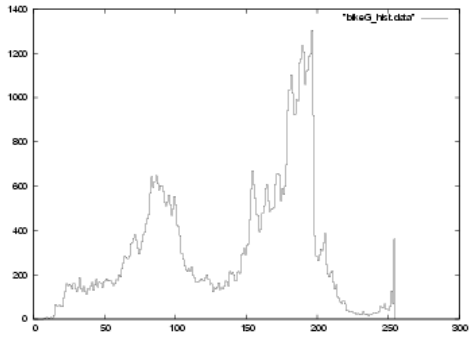
(1a) RGB→YIQ の変換式を示し，各係数がなぜそのような配分になるのか考えて理由を書け．特に，なぜ Y を求めるのに RGB が同じ比率ではないのか，自分の考えを記せ．

(1b) Y は明度だが，他方 I と Q が何を意味するか調べて記せ．

画像処理プログラミング（2 週目）

課題２ 濃淡画像の輝度値のヒストグラムと２値化

ヒストグラムも画像データの統計的な解析のための有効な手段である．濃淡画像の輝度値のヒストグラム（右）は，様々な画像処理のパラメータを決めるために使われる．例えば，濃淡画像の輝度値を黒(0)と白(255) に 2 値化するための閾値決定に用いられる．



輝度値のヒストグラム

プログラミング課題：

濃淡画像の輝度値ヒストグラムを作成せよ. ImageFilter.c 内にヒストグラム作成用の関数 : BrightnessHistogram(void)を作成し，main() 関数から呼び出してヒストグラム用の数値データファイルを生成する．次に，このファイルを GNUPLOT の plot コマンド（with steps オプション付き，付録２参照）でグラフ化する．

白黒 2 値画像

また，作成したヒストグラムを見て，目視で輝度の 2 値化の閾値を決定せよ．また，決定した閾値を用いて 2 値画像（右）を生成せよ．ImageFilter.c 内に PGM ファイル生成用関数：BlackWhiteImage(void)を実装し，main()関数内で呼び出すようにすること．

※2 つ以上の濃淡画像に対してヒストグラムを生成し，2 値化すること．

記述課題：

(2a) 選んだ 2 値化の閾値を示せ.

(2b) なぜ閾値にその数値を選んだのか，客観的な理由を述べよ.

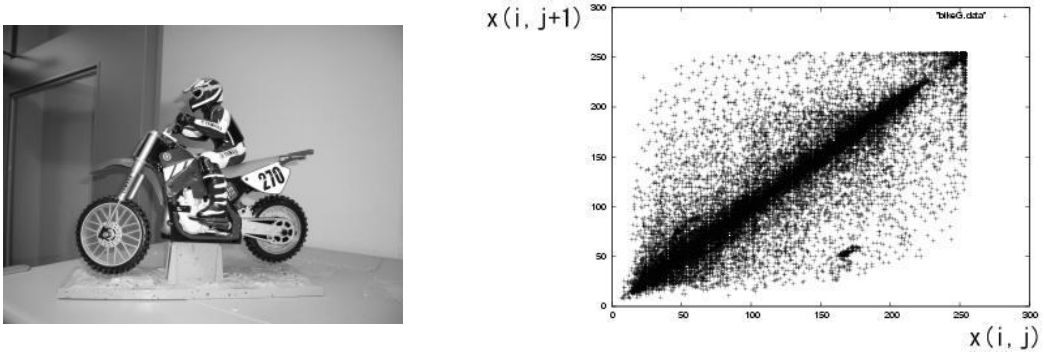
(2c) いつも適切な閾値が選べるかどうか，いくつかの画像について試して検討せよ．また検討結果の理由も述べよ.

(2d) P-タイル法について調べて説明せよ.

画像処理プログラミング（3 週目）

課題３ 濃淡画像の統計的性質（近接画素間の相関度）抽出

一般に，（空間周波数成分に変換した）画像は低周波成分が多く，高周波成分が少ない．つまり画像の多くの部分では画素値が滑らかに変化し，物体間の境界等の画素値が急激に変化している部分は比較的少ないという統計的性質を持つことが知られている．



画像の近接画素間の相関度

課題：

濃淡画像の各画素値における近接する画素間の相関について考察せよ．

(3a) 隣り合う 2 つの画素値をそれぞれ x(i, j)，x(i, j+1)とする．まず，x 軸を x(i, j)，y 軸を x(i, j+1)とする 2 次元信号空間にこれらの画素対をマップするプログラムを作成し，ImageFilter.c 内に追加する．続いて，これを main()関数内で呼び出してそのマップデータファイルを生成すること．GNUPLOT の plot コマンド（with steps オプションなし，付録２参照）でグラフ化せよ．

また，このマップデータの分布状態から，隣り合う 2 つの画素空間の相関度について考察せよ．

(3b) 近接する 3 つの画素値をそれぞれ x(i, j)，x(i, j+1)，x(i, j+2)とする．これらの 3 つの画素値の組を，x 軸を x(i, j)，y 軸を x(i, j+1)，z 軸を x(i, j+2)とする 3 次元信号空間にマップするプログラムを作成し，main()関数内で呼び出してそのマップデータファイルを生成し，グラフ化せよ．

また，このマップデータの分布状態から，隣り合う 2 つの画素空間の相関度について考察せよ．

(3c) 上記の結果を比較し，近接する 2 つの画素間と 3 つの画素間の相関度の違いについて考察せよ．

※2 つ以上の濃淡画像に対して近接する画素間の相関について考察すること．

画像処理プログラミング（4〜5 週目）

課題４ 微分フィルタによるエッジ検出

画像とは各ピクセルが持つ数値データの分布，つまり「場」であると解釈できる（注：「場」とは座標の関数のことである）．例えば濃淡画像は，数学的には 2 次元空間(i,j)における輝度の場$g=f(i,j)$であると解釈できる．fの値が急激に変化する部分を「エッジ」と呼ぶ．エッジは視覚的には境界線として認識される．エッジを画像処理の計算で見つけ出すことを「エッジ検出」と呼び，エッジを可視化した画像を「エッジ画像」と呼ぶ．

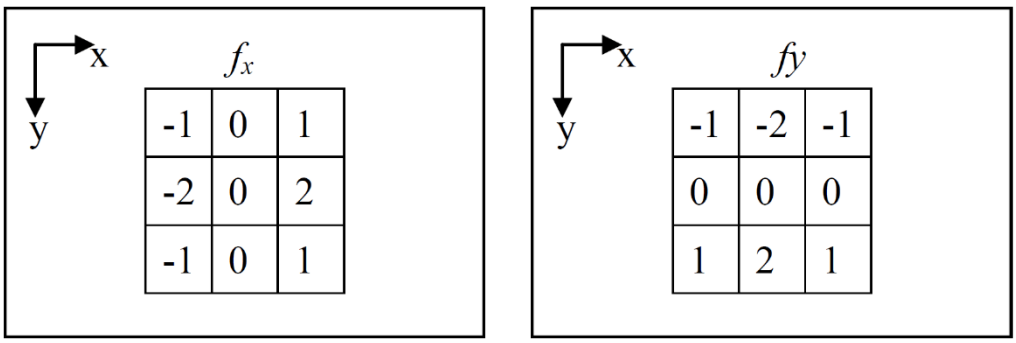
fの値が急激に変化する部分を調べるには，fの勾配∇の大きさ$(|\nabla f|)$を数値計算し，そのピークを探せばよい．これらの微分計算には，幾つかの「微分フィルタ」が提案されており，それらが与える規則どおりに機械的に計算を実行すればよい．微分フィルタの詳細は付録４を参照すること．

なお，通常のエッジ画像は閾値以上の$|\nabla f|$を最大輝度値（255）で可視化し，それ以外を最低輝度値（0）で可視化した 2 値画像であるが，本実験では簡単のため，$|\nabla f|$の大きさをそのまま反映させた濃淡画像を便宜的にエッジ画像と呼ぶ（右）．ただし，|∇f|の最大値が最大輝度に対応するように数値をスケール変換しておくものとする．

$|\nabla f|$の可視化（Sobel フィルタ）

プログラミング課題：

カラー画像を濃淡画像に変換し，Sobel の微分フィルタを用いてエッジ画像を生成し，結果を出力せよ（ImageFilter.c 内に EdgeDetection(void)という関数を実装し，main()関数内で呼び出すようにせよ）．



Sobel の微分フィルタ

画像処理プログラミング（4〜5 週目）

考察課題：

(4a) Sobel フィルタはなぜ上下 3 列の画素の値を使ってエッジコントラストを計算しているのか，その必要性，有用性について自分の考えを述べよ．

(4b) Sobel フィルタを拡張して，5x5,7x7 のようにウィンドウを大きくするとどのような効果があるのか，考えて述べよ（必要に応じて実際に実験しても良い）．

-2	-1	0	1	2
-4	-2	0	2	4
-8	-4	0	4	8
-4	-2	0	2	4
-2	-1	0	1	2

f_x

-2	-4	-8	-4	-2
-1	-2	-4	-2	-1
0	0	0	0	0
1	2	4	2	1
2	4	8	4	2

f_y

Sobel フィルタの拡張（5x5）

画像処理プログラミング（6～7 週目）

課題５ パターンの検出（テンプレートマッチング）

画像中に特定の人物が撮影されているかどうかを調べるためには, 特定の人物の顔をあらかじめ撮影した画像を用意して，画像中にも存在するかを探せばよい.

2 つの画像が同じかどうかを判断するために，画像を重ね合わせて違いを調べるような考え方に基づく処理を，一般にマッチング(matching)とよぶ.

画像の視覚的特徴や画素値そのものをパターン(pattern)といい，パターンの存在や位置を検出することをパターンマッチングと呼ぶ．さらに，あらかじめ標準パターンをテンプレート(template)として用意しておき, テンプレートを用いて入力画像とのマッチングを行うことをテンプレートマッチング(template matching)という. テンプレートマッチングを行うことで, 用意したテンプレートと同じパターンが画像中に存在するかどうか, また存在するとしたらどの位置にあるのかを知ることができる.

下図に示すように，テンプレートを画像全体に対して移動し，それぞれの位置で類似度を調べる．このとき，テンプレートを移動する範囲を探索範囲という．同図のように左端から水平方向に, それを順次下の行に向かって探索することをラスタスキャンという．探索範囲（この場合は画像全体）に対して類似度を調べたら，その中から最も類似する位置を検出する.

テンプレートマッチングにおけるテンプレートの移動

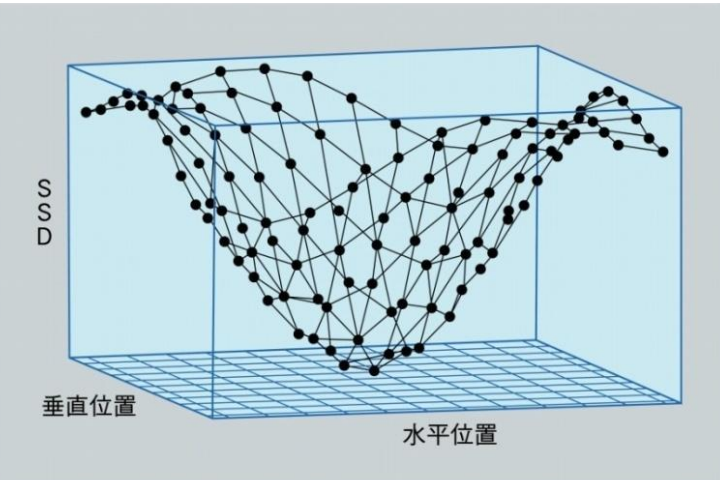
画像処理プログラミング（6〜7 週目）

濃淡画像のテンプレートマッチングでは, 2 つの画像間の類似度(similarity measure)または相違度(dissimilarity measure)を調べるために，次の SSD（Sum of Squared Difference の略．差の 2 乗和．SSD はユークリッド距離の 2 乗）を利用することが多い．

$$SSD(x,y) = \sum_{v=0}^{N-1} \sum_{u=0}^{M-1} |I(x+u, y+v) - T(u,v)|^2 \qquad (1)$$

ただし，テンプレートの大きさを M×N，テンプレートの位置(x, y)における画素値を T(u, v)，テンプレートと重ね合わせた対象画像の画素値を I(x+u, y+v)とする．

SSD は，画像がテンプレートと完全に一致したときに値が 0 になり，相違が大きいほど大きな値になるので，相違度を表している．下図に，SSD 相違度の例を示す．グラフの高さが SSD の値を示す．画像に対してテンプレートを 2 次元的に移動するので，SSD は 2 次元位置に対応して求められる．SSD が最も小さくなるような 2 次元位置を探索することで，対象画像中でのテンプレートの位置を得ることができる．



SSD 相違度の例

課題：

顔とペットボトルのテンプレートを用いて入力画像からテンプレートを検出せよ．また，入力画像から以下の 3 種類のマッチング結果を比較考察せよ．

2 つの画像間の類似度を(1)式を用いて調べることにより，

① ノイズなしの入力画像から，テンプレートを検出せよ．

② ノイズありの入力画像から，テンプレートを検出せよ．

③ ①と②の類似度を比較し，ノイズの影響を考察せよ．

付録１．PGM，PPM 形式の画像ファイルについて

PGM 形式

PGM（portable graymap）形式はグレースケール画像のファイル形式である．端末エミュレータを起動し，サンプルとして提供している Female.pgm を less で表示せよ．

```
> less Female.pgm
```

以下のように表示されるはずである．

```
P2
256 240
255
162 161 159 161 162 158 158 157 155 159 154 155 155 153 154 154 154 156
161 163 165 167 173 172 169 170 167 159 149 146 124 109 96 92 92 99
…
```

１行目の「P2」はこのファイルが PGM 形式であることを表している．２行目は画像の幅と高さを示す．Female.pgm は，幅 256 ピクセル×高さ 240 ピクセルの画像であることがわかる．３行目は輝度値の最大が 255 であることを表している．255 が白，0 が黒に相当する．４行目以降が各ピクセルの輝度値である．輝度値が画像の左上(0,0)から(255,0)，改行して(0,1)から(255,1), . . . というように，右下の(255,239) まで順に記述されている．

PPM 形式

PPM（portable pixmap）形式はカラー画像のファイル形式である．サンプルとして提供している Castle.ppm を less で表示せよ．

```
> less Castle.ppm
```

以下のように表示されるはずである．

```
P3
320 240
255
103 137 198
103 137 198 104 138 199 104 138 199 105 139 200
105 139 200 104 138 199 104 138 199 104 138 199
. . .
```

１行目の「P3」はこのファイルが PPM 形式であることを表している．２行目，３行目は PGM 形式と同様である．4 行目以降が各ピクセルの RGB 値である．数値が３つずつ組となって RGB 値を表すこと以外は, PGM 形式と同様である. つまり, (0,0)の R 値, (0,0)の G 値，(0,0)の B 値，(1,0)の R 値，(1,0)の G 値，(1,0)の B 値 ... のように RGB 値が順に記述されている．

画像の表示

画像の表示は ImageMagic で行うことができる．起動コマンドは display を使用する．例えば PGM ファイル test.pgm を表示するには，端末エミュレータで以下のように入力する．

```
> display test.pgm &
```

ImageMagic はファイル形式の変換機能も持っている．例えば PostScript ファイルへの変換は以下の手順で行えるので，LaTeX でのレポート提出の際に利用するとよい．

(1) 表示された画像を右クリックして Save を選択する．
(2) 出てきたウィンドウ内の Format ボタンをクリックする．
(3) EPS を選択して Select ボタンをクリックする．
(4) 必要であれば保存するファイル名を変更し，Save ボタン，Select ボタンを順にクリックする．

これで（名前を変更していない場合）test.eps というファイル名で PostScript ファイルが生成される．

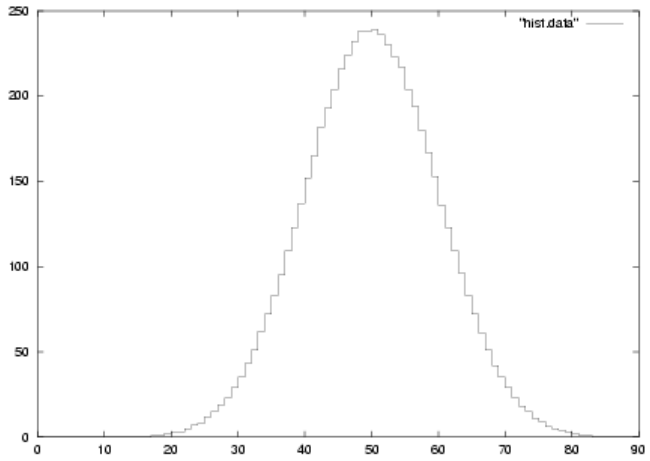
付録 2．GNUPLOT によるグラフ作成

GNUPLOT は多くのプラットフォームで使えるフリーのグラフ作成ソフトウェアである．非常に多機能なソフトウェアであるが，以下では本実験に関係のある機能についてのみ簡単に紹介する．

２次元ヒストグラムの作成

２次元ヒストグラムを作成するには，plot コマンドで with steps オプションを指定すれば良い．

```
> gnuplot
gnuplot> plot "hist.data" with steps
gnuplot> set terminal postscript eps
gnuplot> set output "hist.eps"
gnuplot> replot
gnuplot> exit
> display hist.eps
```



plot コマンド実行例（with steps オプションあり）

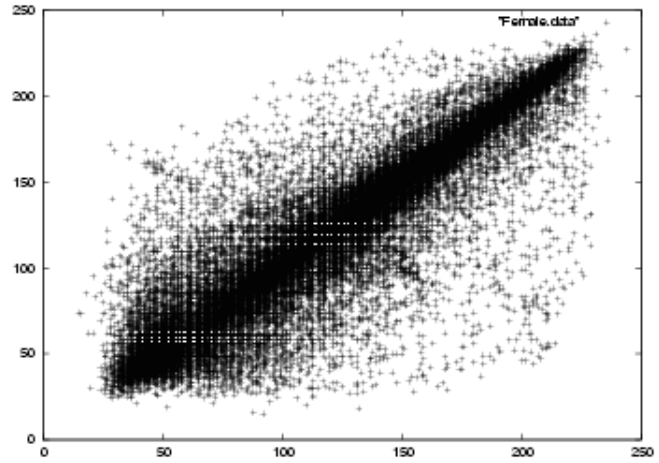
```
x0   y0
x1   y1
x2   y2
x3   y3
. . . . .
```

hist.data のフォーマット

２次元，３次元の相関グラフの作成

２次元の相関グラフを作成する場合，plot コマンドを with steps オプションをつけないで実行すれば良い．3 次元相関グラフの作成には splot コマンドを用いる．

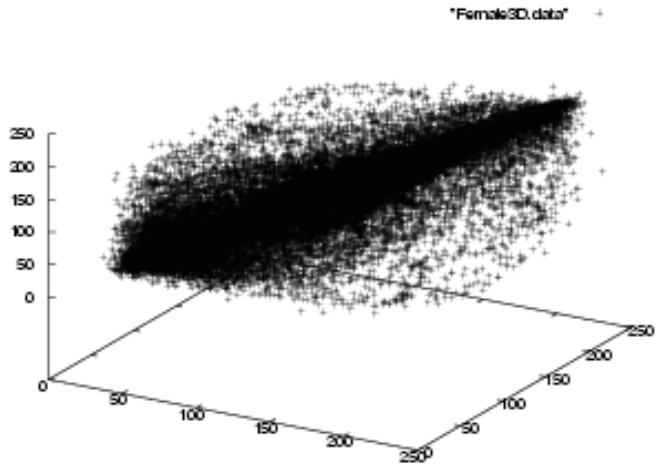
```
> gnuplot
gnuplot> plot "soukan.data"
gnuplot> set terminal postscript eps
gnuplot> set output "soukan.eps"
gnuplot> replot
gnuplot> exit
> display soukan.eps
```



plot コマンド実行例（with steps オプションなし）

```
x0   y0
x1   y1
x2   y2
x3   y3
. . . . .
```

soukan.data のフォーマット



plot コマンド実行例（with steps オプションなし）

```
x0   y0   z0
x1   y1   z1
x2   y2   z2
x3   y3   z3
. . . . .
```

soukan3D.data のフォーマット

付録３．Unix ツール

課題を行うのに必要／便利な Unix ツールについて紹介する．

make

make はコンパイル，リンクの自動実行のためのツール・プログラムである．Makefile という名前のファイルにコンパイルの手順を書いてカレントディレクトリに置き，端末エミュレータから make と入力することでコンパイルとリンクが自動的に行える．サンプルプログラムとともに提供する Makefile は，以下の操作を自動実行するものである．

```
gcc -g -Wall -c -o ImageFileIO.o ImageFileIO.c
gcc -g -Wall -c -o ImageFilter.o ImageFilter.c
gcc -g -Wall -c -o ImageTools.o ImageTools.c
gcc -g -Wall -c -o main.o main.c
gcc -o imgfilter ImageFileIO.o ImageFilter.o ImageTools.o main.o -lm
```

すなわち，ImgeFileIO.c（ファイル入出力関連の関数），ImageFilter.c（画像処理アルゴリズムを実装した関数群），ImageTools.c（複数の課題で共通して使える関数群），そして main.c（メイン関数）をそれぞれコンパイルしてオブジェクトファイルを作成し，次にこれらをリンクして，imgfiler という実行ファイルを生成している．

課題を実行するにあたっては，上記のファイル及びそれらに付随するヘッダファイルに修正を加えると良い．新たにプログラムやファイルを作成する場合は Makefile の修正が必要となる．

grep

文字列検索ツールである．例えば，DrawLines()という関数がどのファイルに定義されているかを知りたい場合，以下の様にすると ImageFilter.c，ImageFilter.h で定義されていることが分かる．

```
> grep DrawLines *.c *.h
. . .
ImageFilter.c: void DrawLines(void) {
. . .
ImageFilter.h: void DrawLines(void);
```

emacs

テキストエディタである．このツールは良く知っているはずなので詳細は述べないが，ひとつだけ便利な機能を紹介する．以下のようにすると，ファイルを開くと同時に指定した行番号にジャンプすることができる．

```
> emacs +行番号  ファイル名
```

less

テキストファイルの表示ツールである．処理の重い emacs を立ち上げなくても less で表示できる．また，ファイルを書き換えてしまう心配もない（注 : emacs にも Read Only モードがある）．使い方は以下の通り．

```
> less  ファイル名
```

j キーで下スクロール, k キーで上スクロール, /キーで文字列検索, などの機能がある．詳しくは man less などで確認せよ．

gdb

デバッガである．gcc によるコンパイル，リンクの際に–g オプションをつけて生成した実行ファイルに対して利用できる．例えば以下のようにすると，プログラムがエラーで異常終了した場所を知ることができる．

```
> gdb imgfilter
gdb> run
gdb> where
```

詳細は man gdb などで確認せよ．

付録４．画像処理のための微分計算

以下で述べる微分計算は，デジタル画像処理への適用を前提としている．このため，以下の点に注意すること．

(a) 微分の定義は差分のままであり，連続極限はとらない．

(b) 差分のステップ幅は（適当な単位を取って）１としてある．

１．偏微分

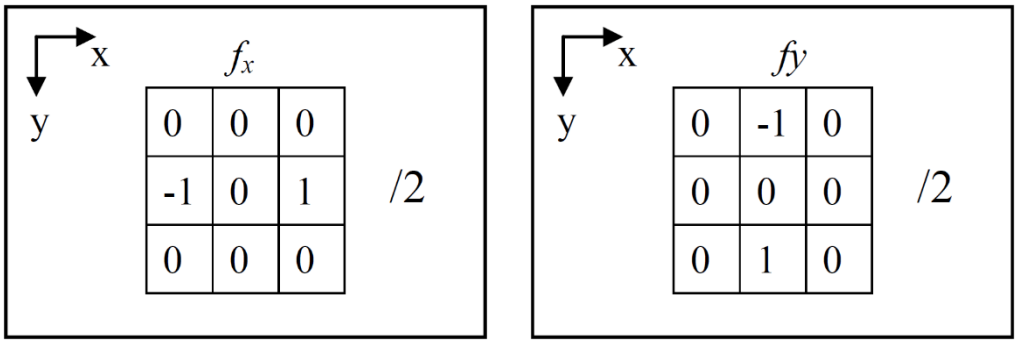
関数$f(x,y)$のx方向とy方向の偏微分をそれぞれ次式で定義する．

$$f_x(x,y) \equiv \frac{f(x+\Delta x,y)-f(x-\Delta x,y)}{2\Delta x} = \frac{f(x+1,y)-f(x-1,y)}{2}$$

$$f_y(x,y) \equiv \frac{f(x,y+\Delta y)-f(x,y-\Delta y)}{2\Delta y} = \frac{f(x,y+1)-f(x,y-1)}{2}$$

ここで，差分化による誤差を小さくするために「中心差分」を用いている．

上式を右辺の分子の各項の係数のみを書き出して，下図のようにxy平面上に並べて略記する．以後，図中の９個の小さい四角形を「セル」と呼ぶ．また，中央のセルを「注目セル」と呼び，このような図示の仕方を微分の「セル表示」と呼ぶ．注目セルの座標は$f(x,y)$の微分を計算する座標点を示す．



微分のセル表示

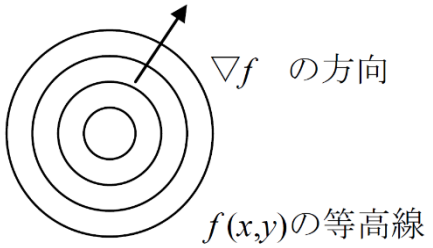
注目セルでの微分を数値計算するには，９個の各セルで係数とfの値の積を求め，それらを単純に足しあわせばよい．全体を２で割る計算は，微分の大きさのピーク値を見つけるエッジ検出などの計算においては省略可能である．

２．勾配

偏微分f_xとf_yを成分とするベクトルを「fの勾配」と呼び，以下のように表す．

$$\nabla f \equiv (f_x, f_y)$$

∇fはfが最も急速に変化する方向
$|\nabla f|$は∇fの方向でのfの変化率
ここで，∇は「ナブラ」と読む．



３．Sobelの微分フィルタ

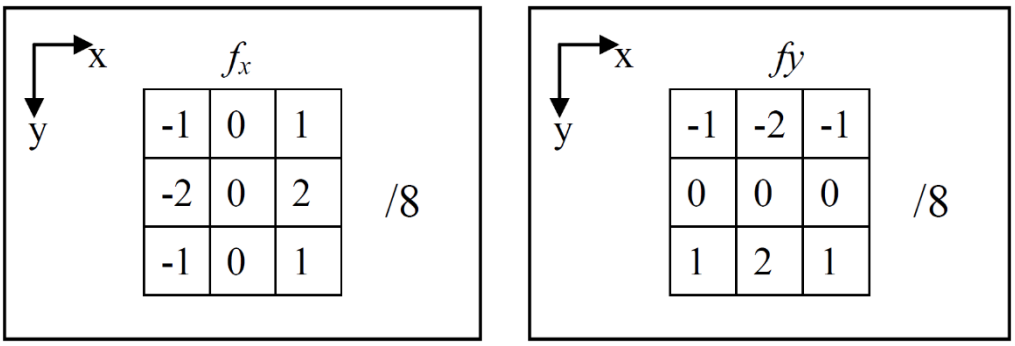
画像データにおいては$f(x,y)$の関数形は与えられず，各(x,y)でのfの数値データが与えられているだけである．この数値データは滑らかに変化するとは限らなので，微分の計算結果にノイズが発生する．また，もともと滑らかな関数から生成された画像データであっても，デジタル化に際した情報欠損等によってノイズが生じることがある．Sobelの微分フィルタは，これらのノイズを補正した微分操作を定義するものである．このアイデアは“平均操作によってノイズの影響を小さくする”というものである．Sobel の微分フィルタに対応する偏微分f_xの定義は以下の様になる．

$$f_x(x,y) \equiv \frac{\overline{f(x+1)^y} - \overline{f(x-1)^y}}{2}$$

添え字yを付けられたオーバーラインは，次式で定義されるy方向の平均操作を表す．

$$\overline{f(x)^y} \equiv \frac{1}{4}\{f(x,y+1) + 2f(x,y) + f(x,y-1)\}$$

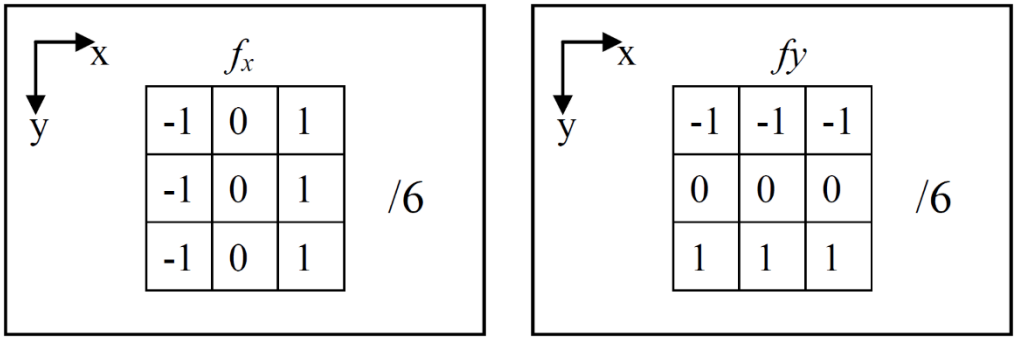
微分の結果への影響の大きさを考慮し，$f(x,y)$の係数（重み）を他の２倍大きくしている（この平均の定義を変えると別の微分フィルタが定義できる）．セル表示でのf_x, f_yはそれぞれ以下のようになる．これを Sobel の微分フィルタと呼ぶ．



Sobel の微分フィルタ

４．Prewittの微分フィルタ

上記の平均操作で，$f(x,y)$の係数（重み）を他の項と同一にしたものである．セル表示は以下の通り．



Prewitt の微分フィルタ

５．ラプラシアンフィルタ

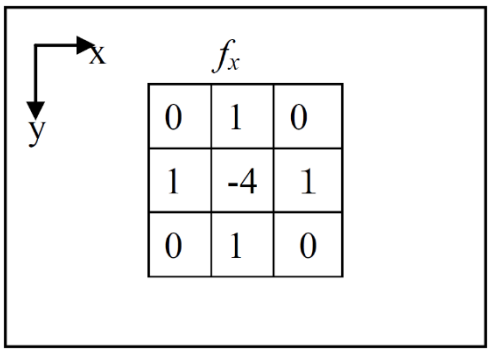
fのラプラシアンの数学的な定義式は以下の通りである．

$$\nabla^2 f \equiv \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

これを差分化すると，中心差分を繰り返し行えば，次式が得られる．

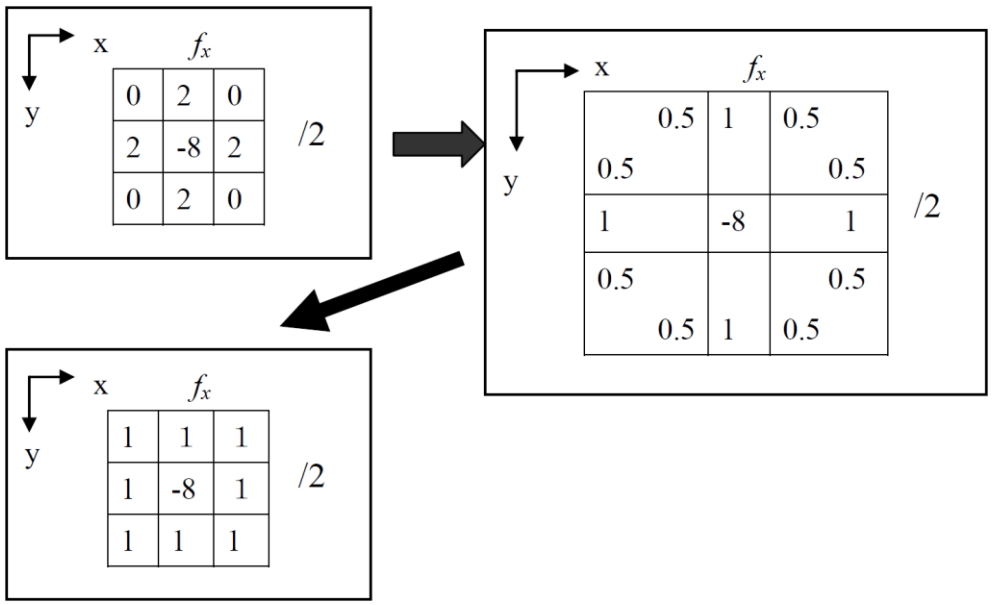
$$\nabla^2 f = f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

セル表示は以下のようになり，４方向ラプラシアンフィルタと呼ばれる．



４方向ラプラシアンフィルタ

４方向ラプラシアンフィルタの全体に２を掛けて２で割ってから Sobel フィルタと同様の考え方で平均化を行えば，８方向ラプラシアンフィルタが得られる．



８方向ラプラシアンフィルタの導出

６．エッジ検出への応用

輝度値$f(x,y)$が急激に変化する箇所をエッジと呼ぶ．従って，$|\nabla f|$または $|\nabla^2 f|$のピークを検出するプログラムを作成すれば良い．エッジを画像として表示する際，処理の最後に値が 0-255 となるようにスケール変換するので，全体を定数で除算する操作は省略できる．

付録 5.　サンプルソースコード

```
//=============================================================================//
//  ファイル名 : ImageFileIO.h                                                 //
//  内      容 : ppm (Portable PixMap), pgm (Portable GrayMap)ファイル          //
//               から入力データの読み込みおよび出力データを書き出しを行う.      //
//  更  新  日 : 2004. 2. 11 木村 彰徳 (Computer Graphics Lab.)                 //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)                     //
//=============================================================================//
#ifndef _IMAGEFILEIO_H_
#define _IMAGEFILEIO_H_ 1

// 構造体 RGB 型
struct RGB {
    int iRed;
    int iGreen;
    int iBlue;
};

// RGB パラメータ
#define RED   0
#define GREEN 1
#define BLUE  2

//-----------------------------------------------------------------------//
// void GetAxisFromIndex(int iWidth, int iIndex, int * iX, int * iY)   //
// 内      容 : １次元のインデックスから２次元座標を取得                 //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)         //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
// [引数]                                                              //
// iWidth       : (入力) 画像の幅 (画素数)                             //
// iX           : (入力) 横方向の座標 (画素数)                         //
// iY           : (入力) 縦方向の座標 (画素数)                         //
// iIndex       : (出力) インデックス                                  //
//-----------------------------------------------------------------------//
void GetAxisFromIndex(int iWidth, int iIndex, int * iX, int * iY);
```

```
//-------------------------------------------------------------------//
// void GetIndexFromAxis(int iWidth, int iX, int iY, int * iIndex)   //
// 内　　容 ： ２次元座標から１次元のインデックスを取得              //
//  更　新　日 ： 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [引数]                                                           //
// iWidth       ： (入力) 画像の幅 (画素数)                         //
// iIndex       ： (入力) インデックス                              //
// iX           ： (出力) 横方向の座標 (画素数)                     //
// iY           ： (出力) 縦方向の座標 (画素数)                     //
//-------------------------------------------------------------------//
void GetIndexFromAxis(int iWidth, int iX, int iY, int * iIndex);

//-------------------------------------------------------------------//
// struct RGB * ReadPpm(int * iWidth, int * iHeight, int * iMaxValue)  //
// 内　　容 ： PPM ファイルの読み込み                                  //
//  更　新　日 ： 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [戻り値]                                                         //
//              ： ２次元([画素数][RGB])の PPM 形式のデータ          //
//                画素数=iWidth * iHeight                           //
// [引数]                                                           //
// iWidth       ： 画像の幅 (画素数)                                //
// iHeight      ： 画像の高さ (画素数)                              //
// iMaxValue    ： 画像の各画素の RGB の最大値                      //
//-------------------------------------------------------------------//
struct RGB * ReadPpm(int * iWidth, int * iHeight, int * iMaxValue);

//-------------------------------------------------------------------//
// void FreePpm(struct RGB * pRGBOutputData)                        //
// 内　　容 ： PPM 形式の画像データを保存しているメモリ領域の開放     //
//  更　新　日 ： 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [引数]                                                           //
// pRGBOutputData ： ２次元([画素数][RGB])の PPM 形式のデータ       //
//-------------------------------------------------------------------//
void FreePpm(struct RGB * pRGBOutputData);
```

```
//-------------------------------------------------------------------//
// void WritePpm(struct RGB * pRGBOutputData, int iWidth,            //
//               int iHeight, int iMaxValue)                         //
// 内      容 : PPM ファイルへ書き出し                                //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [引数]                                                            //
// piRGBOutputData : ２次元([画素数][RGB])の PPM 形式のデータ        //
//                   画素数=iWidth * iHeight                         //
// iWidth       : 画像の幅 (画素数)                                  //
// iHeight      : 画像の高さ (画素数)                                //
// iMaxValue    : 画像の各画素の RGB の最大値                        //
//-------------------------------------------------------------------//
void WritePpm(struct RGB * pRGBOutputData, int iWidth, int iHeight, int iMaxValue);

//-------------------------------------------------------------------//
// int * ReadPgm(int * iWidth, int * iHeight, int * iMaxValue)       //
// 内      容 : PGM ファイルの読み込み                                //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [戻り値]                                                          //
//              : １次元([画素数])の PGM 形式のデータ                //
//                画素数=iWidth * iHeight                           //
// [引数]                                                            //
// iWidth       : 画像の幅 (画素数)                                  //
// iHeight      : 画像の高さ (画素数)                                //
// iMaxValue    : 画像の画素値の最大値                               //
//-------------------------------------------------------------------//
int * ReadPgm(int * iWidth, int * iHeight, int * iMaxValue);
```

```
//-------------------------------------------------------------------//
// void FreePgm(int * piOutputData)                                  //
// 内　　容 : PGM形式の画像データを保存しているメモリ領域の開放　　//
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)           //
// [引数]                                                            //
// piOutputData : ２次元([画素数][RGB])のPGM形式のデータ             //
//-------------------------------------------------------------------//
void FreePgm(int * piOutputData);

//-------------------------------------------------------------------//
// void WritePgm(int * piOutputData, int iWidth,                     //
//               int iHeight, int iMaxValue)                         //
// 内　　容 : PGMファイルへ書き出し                                 //
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)           //
// [引数]                                                            //
// piInputData  : １次元([画素数])のPGM形式のデータ                  //
//                画素数=iWidth * iHeight                            //
// iWidth       : 画像の幅 (画素数)                                  //
// iHeight      : 画像の高さ (画素数)                                //
// iMaxValue    : 画像の画素値の最大値                               //
//-------------------------------------------------------------------//
void WritePgm(int * piOutputData, int iWidth, int iHeight, int iMaxValue);
```

```
//-------------------------------------------------------------------//
// void WritePgm(int * piOutputData, int iWidth,                     //
//               int iHeight, int iResolution)                       //
// 内      容 : PGM ファイルへ書き出し                                //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)            //
// [引数]                                                            //
// piInputData  : 1 次元([画素数])の PGM 形式のデータ                 //
//                画素数=iWidth * iHeight                            //
// iWidth       : 画像の幅 (画素数)                                  //
// iHeight      : 画像の高さ (画素数)                                //
// iResolution  : 画像の解像度                                       //
//-------------------------------------------------------------------//
void WritePgm(int * piOutputData, int iWidth, int iHeight, int iResolution);

#endif
//===================================================================//
```

```
//============================================================================//
//  ファイル名 : ImageFileIO.c                                                //
//  内      容 : ppm (Portable PixMap), pgm (Portable GrayMap)ファイル         //
//               から入力データの読み込みおよび出力データを書き出しを行う.     //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)               //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)                    //
//============================================================================//

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "ImageFileIO.h"

void SkipCommentLine(FILE * fp);

//----------------------------------------------------------------------------//
// void GetAxisFromIndex(int iWidth, int iIndex, int * iX, int * iY)   //
// 内      容 : １次元のインデックスから２次元座標を取得                     //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)              //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)                  //
// [引数]                                                                   //
// iWidth      :  (入力) 画像の幅 (画素数)                                  //
// iX          :  (入力) 横方向の座標 (画素数)                              //
// iY          :  (入力) 縦方向の座標 (画素数)                              //
// iIndex      :  (出力) インデックス                                       //
//----------------------------------------------------------------------------//
void GetAxisFromIndex(int iWidth, int iIndex, int * iX, int * iY) {
    *iX = iIndex%iWidth;
    *iY = iIndex/iWidth;
}
```

```
//-------------------------------------------------------------------//
// void GetIndexFromAxis(int iWidth, int iX, int iY, int * iIndex)   //
// 内　　容 ： ２次元座標から１次元のインデックスを取得              //
//  更　新　日 ：2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [引数]                                                           //
// iWidth       ： (入力) 画像の幅 (画素数)                         //
// iIndex       ： (入力) インデックス                              //
// iX           ： (出力) 横方向の座標 (画素数)                     //
// iY           ： (出力) 縦方向の座標 (画素数)                     //
//-------------------------------------------------------------------//
void GetIndexFromAxis(int iWidth, int iX, int iY, int * iIndex) {
    *iIndex = iWidth*iY + iX;
}

//-------------------------------------------------------------------//
// struct RGB * ReadPpm(int * iWidth, int * iHeight, int * iMaxValue)  //
// 内　　容 ： PPM ファイルの読み込み                                  //
//  更　新　日 ：2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [戻り値]                                                         //
//              ： ２次元([画素数][RGB])の PPM 形式のデータ          //
//                画素数=iWidth * iHeight                           //
// [引数]                                                           //
// iWidth       ： 画像の幅 (画素数)                                //
// iHeight      ： 画像の高さ (画素数)                              //
// iMaxValue    ： 画像の各画素の RGB の最大値                      //
//-------------------------------------------------------------------//
struct RGB * ReadPpm(int * iWidth, int * iHeight, int * iMaxValue) {

    FILE *fp;                   // ファイルポインタ
    enum PPM {P3, P6};          // テキスト形式 or バイナリ形式
    int iPPMType;               // P3 or P6
    char chFileName[80];        // ファイルネーム
    const int MAXCHARS = 100;   // １行の最大文字数
    char chLine[MAXCHARS];      // ファイル形式の認識用変数
    struct RGB * pRGBInputData; // 入力データを格納する変数
    unsigned char * pucData;    // 入力データを格納する変数
```

```
struct RGB sRGBcolor;        // 輝度値
int iNumPixels;              // 画素数
int iIndex;                  // インデックス
int w, h;                    // ループカウンタ

// ファイル名の入力
printf("Input INPUT PPM file name[*.ppm]? > ");
scanf("%s", chFileName);

// 入力されたファイルのオープン
// オープンに失敗の場合は、エラー文を出力し、返り値として１を返す.
if((fp = fopen(chFileName, "r")) == NULL){
    fprintf(stderr, "Cannot open file [%s].¥n", chFileName);
    exit(1);
}

// ヘッダの読み込み
fgets(chLine, MAXCHARS, fp);

// PPM形式(P3)ではない場合、エラー文を出力し、返り値として１を返す
if (strncmp(chLine, "P3", 2) == 0){
    iPPMType = P3;
} else if (strncmp(chLine, "P6", 2) == 0){
    iPPMType = P6;
    fclose(fp);
    fp = fopen(chFileName, "rb");
    fgets(chLine, MAXCHARS, fp);
} else {
    fprintf(stderr, "[%s] is not PPM file.¥n", chFileName);
    exit(1);
}

// コメント行を読み飛ばす
SkipCommentLine(fp);

// 画像データから、画像の幅・高さを読み込む
fgets(chLine, MAXCHARS, fp);
sscanf(chLine, "%d %d", iWidth, iHeight);
```

```
// コメント行を読み飛ばす
SkipCommentLine(fp);

// 画像データから、解像度を読み込む
fgets(chLine, MAXCHARS, fp);
sscanf(chLine, "%d", iMaxValue);
iNumPixels = (*iWidth)*(*iHeight);

// 入力画像データのメモリ領域の確保
// psInputData[iWidth*iHeight]
pRGBInputData = (struct RGB *)malloc(iNumPixels*sizeof(struct RGB));

// 画素値の読み込み
if(iPPMType == P6) {
    pucData = (unsigned char *)malloc(iNumPixels*sizeof(unsigned char[3]));
    fread(pucData, sizeof(unsigned char), iNumPixels*3, fp);
}
for(h = 0; h < *iHeight; h++) {
    for(w = 0; w < *iWidth; w++) {
        if(iPPMType == P3) {
            if(feof(fp)) {
                fprintf(stderr, "Error: [%s] is not a correct PPM file.¥n",
                        chFileName);
                exit(-1);
            }
            // １画素の RGB データの読み込み
            fscanf(fp, "%d %d %d ",
                   &sRGBcolor.iRed, &sRGBcolor.iGreen, &sRGBcolor.iBlue);
            // RGB データの格納
            GetIndexFromAxis(*iWidth, w, h, &iIndex);
            pRGBInputData[iIndex].iRed   = sRGBcolor.iRed;
            pRGBInputData[iIndex].iGreen = sRGBcolor.iGreen;
            pRGBInputData[iIndex].iBlue  = sRGBcolor.iBlue;
        } else { // P6
            GetIndexFromAxis(*iWidth, w, h, &iIndex);
            pRGBInputData[iIndex].iRed   = pucData[3*iIndex];
            pRGBInputData[iIndex].iGreen = pucData[3*iIndex+1];
            pRGBInputData[iIndex].iBlue  = pucData[3*iIndex+2];
```

```
            }
        }
    }

    // ファイルのクローズ
    fclose(fp);

    return pRGBInputData;
}

//---------------------------------------------------------------------//
// void FreePpm(struct RGB * pRGBOutputData)                           //
// 内　　容 : PPM形式の画像データを保存しているメモリ領域の開放        //
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)        //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)              //
// [引数]                                                              //
// pRGBOutputData : ２次元([画素数][RGB])のPPM形式のデータ             //
//---------------------------------------------------------------------//
void FreePpm(struct RGB * pRGBOutputData) {
    free(pRGBOutputData);
}

//---------------------------------------------------------------------//
// void WritePpm(struct RGB * pRGBOutputData, int iWidth,              //
//               int iHeight, int iMaxValue)                           //
// 内　　容 : PPMファイルへ書き出し                                   //
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)        //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)              //
// [引数]                                                              //
// piRGBOutputData : ２次元([画素数][RGB])のPPM形式のデータ            //
//                画素数=iWidth * iHeight                              //
// iWidth       : 画像の幅 (画素数)                                    //
// iHeight      : 画像の高さ (画素数)                                  //
// iMaxValue    : 画像の各画素のRGBの最大値                            //
//---------------------------------------------------------------------//
void WritePpm(struct RGB * pRGBOutputData, int iWidth, int iHeight, int iMaxValue) {
```

```
    FILE *fp;               // ファイルポインタ
    char chFileName[80];    // ファイルネーム
    int i;                  // ループカウンタ

    // 出力ファイル名の入力
    printf("Input OUTPUT PPM file name[*.ppm]? > ");
    scanf("%s", chFileName);

    // 出力ファイルのオープン
    // オープンに失敗の場合は、エラー文を出力し、返り値として１を返す.
    if((fp = fopen(chFileName, "w")) == NULL){
        fprintf(stderr, "Cannot open file [%s].¥n", chFileName);
        exit(1);
    }

    // ヘッダ領域書き込み(画像サイズ等)
    fprintf(fp, "P3¥n");
    fprintf(fp, "%d %d¥n", iWidth, iHeight);
    fprintf(fp, "%d¥n", iMaxValue); // 最大値

    // 各画素データ書き込み
    for(i = 0; i < iWidth*iHeight; i++){
        // 画素値をファイルの書き込み
        fprintf(fp, "%5d%5d%5d  ",
                  pRGBOutputData[i].iRed,
                  pRGBOutputData[i].iGreen,
                  pRGBOutputData[i].iBlue);

        // ５画素値（RGB）ごとに改行
        if((i + 1)%5 == 0){
           fprintf(fp, "¥n");
        }
    }

    // ファイルのクローズ
    fclose(fp);
}
```

```
//-------------------------------------------------------------------//
// int * ReadPgm(int * iWidth, int * iHeight, int * iMaxValue)       //
// 内　　容 : PGM ファイルの読み込み                                  //
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [戻り値]                                                          //
//              : １次元([画素数])の PGM 形式のデータ                 //
//                画素数=iWidth * iHeight                           //
// [引数]                                                            //
// iWidth       : 画像の幅 (画素数)                                  //
// iHeight      : 画像の高さ (画素数)                                //
// iMaxValue    : 画像の画素値の最大値                               //
//-------------------------------------------------------------------//
int * ReadPgm(int * iWidth, int * iHeight, int * iMaxValue) {

    FILE *fp;                   // ファイルポインタ
    char chFileName[80];        // ファイルネーム
    enum PGM {P2, P5};          // テキスト形式 or バイナリ形式
    int iPGMType;               // P2 or P5
    const int MAXCHARS = 100;   // １行の最大文字数
    char chLine[MAXCHARS];      // ファイル形式の認識用変数
    int * piInputData;          // 入力データを格納する変数
    unsigned char * pucData;    // 入力データを格納する変数
    int iNumPixels;             // 画素数
    int iIndex;                 // インデックス
    int w, h;                   // ループカウンタ

    // ファイル名の入力
    printf("Input INPUT PGM file name[*.pgm]? > ");
    scanf("%s", chFileName);

    // 入力されたファイルのオープン
    // オープンに失敗の場合は、エラー文を出力し、返り値として１を返す.
    if((fp = fopen(chFileName, "r")) == NULL){
        fprintf(stderr, "Cannot open file [%s].¥n", chFileName);
        exit(1);
    }
```

```
// ヘッダ領域読み込み
fgets(chLine, MAXCHARS, fp);

// PPM形式(P3)ではない場合、エラー文を出力し、返り値として１を返す
if (strncmp(chLine, "P2", 2) == 0){
    iPGMType = P2;
} else if (strncmp(chLine, "P5", 2) == 0){
    iPGMType = P5;
    fclose(fp);
    fp = fopen(chFileName, "rb");
    fgets(chLine, MAXCHARS, fp);
} else {
    fprintf(stderr, "[%s] is not PGM file.¥n", chFileName);
    exit(1);
}

// コメント行を読み飛ばす
SkipCommentLine(fp);

// 画像データから、画像の幅・高さを読み込む
fgets(chLine, MAXCHARS, fp);
sscanf(chLine, "%d %d", iWidth, iHeight);

// コメント行を読み飛ばす
SkipCommentLine(fp);

// 画像データから、解像度を読み込む
fgets(chLine, MAXCHARS, fp);
sscanf(chLine, "%d", iMaxValue);
iNumPixels = (*iWidth)*(*iHeight);

// 入力画像データのメモリ領域の確保
// piInputData[iWidth*iHeight]
piInputData = (int *)(malloc(iNumPixels*sizeof(int)));

// コメント行の読み飛ばし
SkipCommentLine(fp);
```

```
    // 画素値の読み込み
    if(iPGMType == P5) {
        pucData = (unsigned char *)malloc(iNumPixels*sizeof(unsigned char));
        fread(pucData, sizeof(unsigned char), iNumPixels, fp);
    }
    for(h = 0; h < *iHeight; h++) {
        for(w = 0; w < *iWidth; w++) {
            if(iPGMType == P2) {
                  if(feof(fp)) {
                      fprintf(stderr, "Error: [%s] is not a correct PGM file.¥n",
                                chFileName);
                      exit(-1);
                  }

                  // 1画素のデータの読み込み, 格納
                  GetIndexFromAxis(*iWidth, w, h, &iIndex);
                  fscanf(fp, "%d", &piInputData[iIndex]);
            } else { // P5
                  GetIndexFromAxis(*iWidth, w, h, &iIndex);
                  piInputData[iIndex] = pucData[iIndex];
            }
        }
    }

    // ファイルのクローズ
    fclose(fp);

    return piInputData;
}
```

```
//---------------------------------------------------------------------//
// void FreePgm(int * piOutputData)                                    //
// 内　　容 : PGM形式の画像データを保存しているメモリ領域の開放　　//
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)           //
// [引数]                                                              //
// piOutputData : ２次元([画素数][RGB])のPGM形式のデータ               //
//---------------------------------------------------------------------//
void FreePgm(int * piOutputData) {
    free(piOutputData);
}

//---------------------------------------------------------------------//
// void WritePgm(int * piOutputData, int iWidth,                       //
//               int iHeight, int iMaxValue)                           //
// 内　　容 : PGMファイルへ書き出し                                   //
//  更　新　日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)       //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)           //
// [引数]                                                              //
// piInputData  : １次元([画素数])のPGM形式のデータ                    //
//                画素数=iWidth * iHeight                              //
// iWidth       : 画像の幅 (画素数)                                    //
// iHeight      : 画像の高さ (画素数)                                  //
// iMaxValue    : 画像の画素値の最大値                                 //
//---------------------------------------------------------------------//
void WritePgm(int * piOutputData, int iWidth, int iHeight, int iMaxValue) {

    FILE *fp;              // ファイルポインタ
    char chFileName[80];   // ファイルネーム
    int i;                 // ループカウンタ

    // 出力ファイル名の入力
    printf("Input OUTPUT PGM file name[*.pgm]? > ");
    scanf("%s", chFileName);

    // 出力ファイルのオープン
    // オープンに失敗の場合は、エラー文を出力し、返り値として１を返す.
    if((fp = fopen(chFileName, "w")) == NULL){
```

```
        fprintf(stderr, "Cannot open file [%s].¥n", chFileName);
        exit(1);
    }

    // ヘッダ領域書き込み(画像サイズ等)
    fprintf(fp, "P2¥n");
    fprintf(fp, "%d %d¥n", iWidth, iHeight);
    fprintf(fp, "%d¥n", iMaxValue);  // 最大値

    // 各画素データ書き込み
    for(i = 0; i < iWidth*iHeight; i++){

        // 画素値をファイルの書き込み
        fprintf(fp, "%5d", piOutputData[i]);

        // 10 画素値ごとに改行
        if((i + 1)%15 == 0){
           fprintf(fp, "¥n");
        }
    }

    // ファイルのクローズ
    fclose(fp);
}

//-------------------------------------------------------------------//
// void SkipCommentLine(FILE * fp)                                   //
// 内      容 : コメント行を読み飛ばす                               //
//  更  新  日 : 2004. 2. 15 木村 彰徳 (Computer Graphics Lab.)      //
                2013. 9. 5 脇田 航 (Computer VIsion Lab.)           //
// [引数]                                                           //
// fp     : ファイルストリームポインタ                               //
//-------------------------------------------------------------------//
void SkipCommentLine(FILE * fp) {

    const int MAXCHARS = 100;
    char chLine[MAXCHARS];
    fpos_t current;
```

```
    while(!feof(fp)) {
        // 現在のストリーム位置の保存
        fgetpos(fp, &current);

        // １行取得
        fgets(chLine, MAXCHARS, fp);

        // コメント行でなければ，ストリーム位置を戻してリターン
        if(chLine[0] != '#') {
            fsetpos(fp, &current);
            return;
        }
    }
}
//=====================================================================//
```

```
//=====================================================================//
//  ファイル名 : ImageTools.h                                          //
//  内      容 : PGM (Portable GrayMap)形式のデータにプリミティブな    //
//               図形を描くため関数の定義                              //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//=====================================================================//
#ifndef _IMAGETOOLS_H_
#define _IMAGETOOLS_H_

//---------------------------------------------------------------------//
// void DrawStraightLine(int * piImageData,                            //
//                       int iWidth, int iHeight, int iMaxValue,       //
//                       int iStartX, int iStartY, int iEndX, int iEndY, //
//                       int STYLE, int iBW)                           //
//  内      容 : PGM形式のデータに直線を引く                           //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
// [引数]                                                              //
// piImageData : (入力) 直線を描くのPGM形式のデータ                    //
// iWidth      : (入力) 直線を描くのPGM形式のデータの幅 (画素数)       //
// iHeight     : (入力) 直線を描くのPGM形式のデータの高さ (画素数)     //
// iMaxValue   : (入力) 直線を描くのPGM形式のデータの画素の最大値      //
// iStartX     : (入力) 直線の始点の X 座標                            //
// iStartY     : (入力) 直線の始点の Y 座標                            //
// iEndX       : (入力) 直線の終点の X 座標                            //
// iEndY       : (入力) 直線の終点の Y 座標                            //
// iSTYLE      : (入力) 直線の色, SOLID, DASHED または DOTTED          //
// iBW         : (入力) 直線の色, BLACKLINE または WHITELINE           //
//---------------------------------------------------------------------//
enum lineStype {SOLID, DASHED, DOTTED};
enum lineBW {BLACKLINE, WHITELINE};
void DrawStraightLine(int * piImageData,
                      int iWidth, int iHeight, int iMaxValue,
                      int iStartX, int iStartY, int iEndX, int iEndY,
                      int iSTYLE, int iBW);
```

```
//-------------------------------------------------------------------//
// void Histogram1D(int * piHistData, int iNumHistData,              //
//                  int iNumBin, int iMinBin, int iMaxBin, int iGRID)  //
//                                                                   //
//  内　容 : 1次元ヒストグラム                                       //
//  更　新　日 : 2004. 2. 13 木村 彰徳 (Computer Graphics Lab.)      //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)          //
// [引数]                                                           //
// piHistData   : (入力) ヒストグラミングするデータ                  //
// iNumHistData : (入力) ヒストグラミングするデータ数                //
// iNumBin      : (入力) ビン数                                     //
// iMinBin      : (入力) ビンの最小値                               //
// iMaxBin      : (入力) ビンの最大値                               //
// iGRID        : (入力) グリッドの有無 (GRID, NOGRID, XGRID, YGRID)  //
//-------------------------------------------------------------------//
#define NOGRID 0
#define GRID   1
#define XGRID  2
#define YGRID  3
void Histogram1D(int * piHistData, int iNumHistData,
                 int iNumBin, int iMinBin, int iMaxBin, int iGRID);


#endif
//===================================================================//
```

```
//=====================================================================//
//  ファイル名 : ImageTools.c                                          //
//  内      容 : PGM (Portable GrayMap)形式のデータにプリミティブな    //
//               図形を描くため関数の定義                              //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//=====================================================================//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "ImageTools.h"
#include "ImageFileIO.h"

//---------------------------------------------------------------------//
// void DrawStraightLine(int * piImageData,                            //
//                       int iWidth, int iHeight, int iMaxValue,       //
//                       int iStartX, int iStartY, int iEndX, int iEndY, //
//                       int iSTYLE, int iBW)                          //
//  内      容 : PGM形式のデータに直線を引く                           //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
// [引数]                                                              //
// piImageData : (入力) 直線を描くのPGM形式のデータ                    //
// iWidth      : (入力) 直線を描くのPGM形式のデータの幅 (画素数)       //
// iHeight     : (入力) 直線を描くのPGM形式のデータの高さ (画素数)     //
// iMaxValue   : (入力) 直線を描くのPGM形式のデータの画素の最大値      //
// iStartX     : (入力) 直線の始点の X 座標                            //
// iStartY     : (入力) 直線の始点の Y 座標                            //
// iEndX       : (入力) 直線の終点の X 座標                            //
// iEndY       : (入力) 直線の終点の Y 座標                            //
// iSTYLE      : (入力) 直線の色. SOLID, DASHED または DOTTED          //
//               (ただし未実装)                                        //
// iBW         : (入力) 直線の色, BLACKLINE または WHITELINE           //
//---------------------------------------------------------------------//
void DrawStraightLine(int * piImageData,
                      int iWidth, int iHeight, int iMaxValue,
                      int iStartX, int iStartY, int iEndX, int iEndY,
                      int iSTYLE, int iBW) {
```

```
 int iIndex;                          // インデックス
 float w, h;                          // 座標ループカウンタ
 const float fStep = 0.5;             // 直線を描くステップ
 const int iBLACK = 0;                // 黒
 const int iWHITE = iMaxValue;        // 白
 int iXmin, iXmax, iYmin, iYmax;      // 座標
float fx(float x) {
    float y = (iEndY - iStartY)*(x - iStartX)/(iEndX - iStartX) + iStartY;
    return y;
 }
 float fy(float y) {
    float x = (iEndX - iStartX)*(y - iStartY)/(iEndY - iStartY) + iStartX;
    return x;
 }
 // 直線の描画
 // |tanθ| < 1 または iStartY == iEndY
 if(abs(iEndY - iStartY) < abs(iEndX - iStartX) || iStartY == iEndY) {
    if(iStartX > iEndX) {
        iXmin = iEndX;
        iXmax = iStartX;
    } else {
        iXmin = iStartX;
        iXmax = iEndX;
    }
    for(w = iXmin; w < iXmax; w+=fStep) {
        h = fx(w);
        if(h >= 0 && h < iHeight) {
            GetIndexFromAxis(iWidth, (int)w, (int)h, &iIndex);
              if(iBW == BLACKLINE) {  // 黒
                  piImageData[iIndex] = iBLACK;
              } else {                // 白
                  piImageData[iIndex] = iWHITE;
              }
        }
    }
```

```
    // |tanθ| >= 1 または iStartX == iEndX
    }else {
        if(iStartY > iEndY) {
            iYmin = iEndY;
            iYmax = iStartY;
        } else {
            iYmin = iStartY;
            iYmax = iEndY;
        }

        for(h = iYmin; h < iYmax; h+=fStep) {
            w = fy(h);
            if(w >= 0 && w < iWidth) {
                  GetIndexFromAxis(iWidth, (int)w, (int)h, &iIndex);
                  if(iBW == BLACKLINE) {  // 黒
                      piImageData[iIndex] = iBLACK;
                  } else {                // 白
                      piImageData[iIndex] = iWHITE;
                  }
            }
        }
    }
 }
//=========================================================================//
```

```
//=====================================================================//
//  ファイル名 : ImageFilter.h                                         //
//  内      容 : PPM(Portable PixMap) または PGM(Portable GrayMap)形式 //
//               のデータを画像処理する関数の定義                      //
//  更  新  日 : 2004. 2. 14 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//=====================================================================//
#ifndef _IMAGEFILTER_H_
#define _IMAGEFILTER_H_

//---------------------------------------------------------------------//
// 例１)                                                               //
// void  NegativeImage(void)                                           //
//  内      容 : PPM形式の入力データをネガ画像に変換                    //
//  更  新  日 : 2004. 2. 11 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//---------------------------------------------------------------------//
void NegativeImage(void);

//---------------------------------------------------------------------//
// 例２)                                                               //
// void DrawLines(void)                                                //
//  内      容 : PGM形式の入力データに線を描く                          //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//---------------------------------------------------------------------//
void DrawLines(void);

#endif
//=====================================================================//
```

```
//=====================================================================//
//  ファイル名 : ImageFilter.c                                         //
//  内      容 : PPM(Portable PixMap) または PGM(Portable GrayMap)形式 //
//               のデータを画像処理する関数の定義                      //
//  更  新  日 : 2004. 2. 14 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//=====================================================================//
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <ctype.h>

#include "ImageFilter.h"

#include "ImageFileIO.h"
#include "ImageTools.h"

//---------------------------------------------------------------------//
// 例１)                                                               //
// void  NegativeImage(void)                                           //
//  内      容 : PPM形式の入力データをネガ画像に変換                    //
//  更  新  日 : 2004. 2. 11 木村 彰徳 (Computer Graphics Lab.)        //
//               2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//---------------------------------------------------------------------//
void NegativeImage(void) {

    int iWidth, iHeight, iMaxValue;   // 画像の幅, 高さ, 解像度
    struct RGB * pRGBInputData;       // 入力データを格納
    struct RGB * pRGBOutputData;      // 画像処理したデータを格納
    int i;                            // ループカウンタ

    // ネガ画像
    printf("¥n**** Negative Image *****¥n");

    // PPM形式の入力データの読み込み
    pRGBInputData = ReadPpm(&iWidth, &iHeight, &iMaxValue);
```

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

    // 入力データ(pRGBInputmData)の RGB の値を反転して,
    // 出力データ(pRGBOutputData)に保存
    for(i = 0; i < iWidth*iHeight; i++){
        pRGBOutputData[i].iRed   = iMaxValue - pRGBInputData[i].iRed;
        pRGBOutputData[i].iGreen = iMaxValue - pRGBInputData[i].iGreen;
        pRGBOutputData[i].iBlue  = iMaxValue - pRGBInputData[i].iBlue;
    }

    // PPM 形式のファイルに出力データを出力
    WritePpm(pRGBOutputData, iWidth, iHeight, iMaxValue);

    // 入力データを保存しているメモリ領域を解放
    FreePpm(pRGBInputData);

    // 出力データを保存しているメモリ領域を解放
    FreePpm(pRGBOutputData);
}
```

```
//---------------------------------------------------------------------//
// 例2)                                                                //
// void DrawLines(void)                                                //
//  内       容 : PGM形式のデータに線を描く                             //
//  更  新  日 : 2004. 2. 12 木村 彰徳 (Computer Graphics Lab.)        //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//---------------------------------------------------------------------//
void DrawLines(void) {
    int iWidth, iHeight, iMaxValue;     // 画像の幅, 高さ, 解像度
    int * piOutputData;                 // 画像処理したデータを格納
    int iIndex;                         // インデックス
    //int w, h, x, y;                   // 座標ループカウンタ
    int iStartX, iStartY, iEndX, iEndY; // 直線の始点と終点座標
    int iWHITE;                         // 白の値

    // 線を描く
    printf("¥n**** Draw Lines *****¥n");

    // 画像の幅, 高さ, 解像度の指定と白の設定
    iWidth  = 400;
    iHeight = 400;
    iMaxValue = 255;
    iWHITE = iMaxValue;

    // 出力データ（1次元）のメモリ領域の確保
    // piOutputData[iWidth*iHeight]
    piOutputData = (int *)malloc(iWidth*iHeight*sizeof(int));

    // 画素を白で初期化
    for(iIndex = 0; iIndex < iWidth*iHeight; iIndex++)
        piOutputData[iIndex] = iWHITE;
```

```
    // 出力データ(piOutputData)に線を描く
    iStartX = 0; iStartY = 0;
    iEndY = iHeight;
    for(iEndX = iWidth - 1; iEndX >= 0; iEndX-=5) {
        DrawStraightLine(piOutputData, iWidth, iHeight, iMaxValue,
                         iStartX, iStartY, iEndX, iEndY, SOLID, BLACKLINE);
    }
    iEndX = iWidth;
    for(iEndY = iHeight - 1; iEndY >= 0; iEndY-=5) {
        DrawStraightLine(piOutputData, iWidth, iHeight, iMaxValue,
                         iStartX, iStartY, iEndX, iEndY, SOLID, BLACKLINE);
    }

    // PGM形式のファイルに出力データを出力
    WritePgm(piOutputData, iWidth, iHeight, iMaxValue);

    // 出力データを保存しているメモリ領域を解放
    FreePgm(piOutputData);
}
//=========================================================================//
```

```
//=====================================================================//
//  ファイル名 : main.c                                                //
//  内      容 : 画像処理プログラムのメインプログラム                  //
//  更  新  日 : 2004. 2. 17 木村 彰徳 (Computer Graphics Lab.)        //
                 2013. 9. 5 脇田 航 (Computer VIsion Lab.)             //
//=====================================================================//
#include <stdio.h>

#include "ImageFilter.h"

int main(void) {

    // 画像処理を行う関数の呼び出し
    // 例１）PPM形式の入力データをネガ画像に変換
    NegativeImage();

     // 例２）PGM形式のデータに線を描く
    // DrawLines();

    // 正常終了
    return 0;
}
//=====================================================================//
```

--------- 以上 ---------