

# マクロプログラムでマルチ周波数処理を体験する

Wataru Ogawa

feb.11.2007

## 1. はじめに

プログラミングができるようになるには、小さなプチプログラムを完成させて動作させ、自信をつけることが一番です。そこで今回はいきなりマルチ周波数処理のプログラムを書かないで、昨年おこなったボケマスクのプログラムを書くところから始めます。

## 2. プログラムの方針を立てる

とりあえず、3x3 の平滑化フィルタで強調係数が 1 のボケマスク処理を考えます。

処理手順

- 処理する画像はあらかじめ読み込んでおきます。読み込んだ画像の PidNumber を格納する変数を pid1 とします。
- 画像の複製をつくります。この画像の PidNumber を格納する変数を pid2 とします。
- 複製した画像に対し平滑化処理を行います。平滑化フィルターをあらかじめ用意しておく方法もありますが、今回は以下のようにプログラム上にフィルタを記述して使用します。

```
NewTextWindow('3x3 average',120,80);   新しいテキストウィンドウを開いて  
writeln(' 1 1 1 ');                    カーソル位置にテキストを書く  
writeln(' 1 1 1 ');                    //  
writeln(' 1 1 1 ');                    //
```

- 作成したフィルタカーネルを Convolve(""); 命令でたたみ込みます。
- 平滑化された画像を原画像から ImageMath('subtract', 命令を使って差し引き、ボケマスクを得ます。
- 原画像にボケマスクを ImageMath('add', 命令を使って加え、処理画像を得ます。

使用する変数

pid1、pid2      整数なので、変数を宣言する部分で integer とします。

使用する関数

Duplicate                      : 複製を作成します

NewTextWindow : 新しいテキストウィンドウをひらきます  
writeln : カーソル位置にテキストを書いて改行します  
Convolve : フィルタをコンボリューション(たたみ込み)します  
Dispose : ウィンドウを閉じます  
ImageMath : 画像間演算を行います

```
-----8<-----8<-----  
macro 'USM 3x3';  
var  
  pid1,pid2 : integer;  
  
begin  
  
  pid1 := PidNumber;  
  Duplicate('USM 3x3 強調係数=1');  
  pid2 := PidNumber;  
  
  NewTextWindow('3x3 average',120,80);  
  writeln(' 1 1 1');  
  writeln(' 1 1 1');  
  writeln(' 1 1 1');  
  Convolve('');  
  Dispose;  
  
  ImageMath('subtract', pid1, pid2, 1, 128, pid2);  
  
  ImageMath('add', pid1, pid3, 1, -128, pid2);  
  
end;
```

原画像からボケ画像を引くと負の値が発生してしまう。負の値はすべて0になるため、offsetとして128を加えている

実際にマクロを読み込んで処理結果を確認してみてください。いかがでしょうか？エラーが出たり、うまく動作しなかった方は上記のプログラムリストと良く見比べて間違いを捜してください。これをプログラムのデバッグといいます。実際にはプログラムリストと比べることだけをさすのではありませんが、似たような作業だと思ってください。本格的なプログラムを作成すると、実際のコーディングよりデバッグの方がずっと分量の多いやっかいな作業になります。

### 3. プログラムを改良する

作成したマクロ“USM 3x3”は強調係数を 1 として作成しました。一般的には小さなカーネルのボケマスクでは強調係数に大きな値を使用します。そこで、上記のマクロに強調係数をいろいろ変えて結果を試せるように改良します。ここからはマルチ周波数処理のプログラムでも使うテクニックが入ってきます。

#### プログラムの変更手順

- GetNumber を使って、ダイアログボックスを開いて強調係数 K を入力します。
- 強調係数は負の値のあるボケマスクに掛けるので次のような計算および処理が必要です。
  - GetPicSize を使って、画像のサイズを取得します(1 ピクセルずつ強調係数を計算させます)。変数 Width, Height を定義します。
  - for next ループを使って、x 方向と y 方向に 1 ピクセルずつ移動させます。変数 x, y を定義します。
  - GetPixel を使って、ピクセルの値を取得し変数に代入します。変数 UPv を定義します。
  - 変数 Pv を定義し、計算結果を代入します。Pv の値を PutPixel を使ってピクセルの値と置き換えます。

#### 新たに宣言する変数

Width, Height	: 画像の幅と高さを格納する変数(integer:整数)
K	: 強調係数(Real:実数)
str	: 文字列を格納する変数(string:文字列)
X, y	: 画像上のピクセルを移動するための変数(integer:整数)
Upv	: ボケマスクのピクセル値を格納する変数(real:実数)
Pv	: ピクセルごとの計算結果を格納するための変数(real:実数)

#### 新たに使用する関数

GetNumber	: ダイアログボックスを開いて数値の入力を求めます
concat	: 文字列をつなぎます
GetPicSize	: 画像のサイズを幅と高さで取得します
GetPixel	: 画像のピクセル値を取得します
PutPixel	: 画像上にピクセル値を与えます
for next do: for next	で繰り返し処理を行います
if then else:	もし(if)~ならば(then)~それ以外は(else)~という条件判断文

```

macro 'USM 3x3';
var
  K          : real;
  Width,Height : integer;
  x,y       : integer;
  Upv,Pv    : real;
  pid1,pid2 : integer;
  str       : string;

begin
  pid1 := PidNumber;
  GetPicSize(Width,Height);
  K := GetNumber('強調係数',3,0);
  str := concat('USM 3x3 K=',K);
  Duplicate(str);
  pid2 := PidNumber;

  NewTextWindow('3x3 average',120,80);
  writeln(' 1 1 1');
  writeln(' 1 1 1');
  writeln(' 1 1 1');
  Convolve('');
  Dispose;
  ImageMath('subtract', pid1, pid2, 1, 128, pid2);

  for y := 0 to Height - 1 do begin
    for x := 0 to Width - 1 do begin
      Upv := GetPixel(x,y) - 128;
      if Upv = 0 then Pv := 128 else
        Pv := Upv * K + 128;
      PutPixel(x,y,Pv);
    end;
  end;

  ImageMath('add', pid1, pid2, 1, -128, pid2);
end;

```

だんだんプログラムが大きくなってきましたが、最初にした小さなプログラムと比較して変更した箇所を注意して見てください。

処理前の画像



USM 3x3 K=3 の処理結果



#### 4. マクロメニューを増やす

できあがったプログラムをすべてコピーして、そのプログラムの下に貼り付けてください。今回は 9x9 サイズの平滑化フィルターを使うボケマスク処理を作ります。変更するのは以下の部分です。

```
-----8<-----8<-----  
macro 'USM 9x9';  
  .  
  .  
  str := concat('USM 9x9 K=',K);  
  .  
  .  
  NewTextWindow('9x9 average',120,80);
```

```
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
```

-----8<-----8<-----

マクロに USM 9x9 というメニューが増えているのを確認したら、早速試してみてください。3x3 のときと比べて結果はどうだったでしょうか？

USM 3x3 K=3



USM 9x9 K=3



## 5. マルチ周波数処理に挑戦

さていよいよマルチ周波数処理に挑戦です。マルチ周波数処理のポイントを簡単にまとめると以下ようになります。

- 平滑化に重み付け平均を用いる。(エッジがなめらかな平滑化画像を作成する) 実際にはカーネルサイズを変えた複数の平滑化フィルターを用います。
- コントラスト依存非線形関数変換を使用します。実際にはボケマスクにおいてピクセル値の大きい(または小さい)部分の値を抑制します。すなわち画像上でコントラストの高い部分の強調を抑制します。

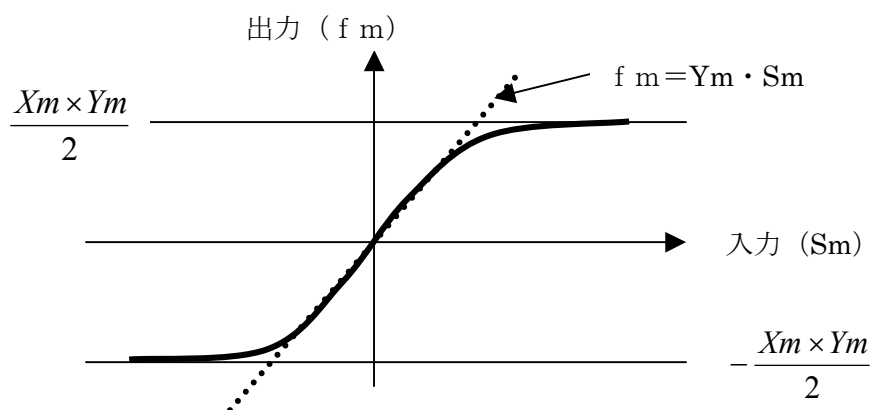
プログラムの方針

- 基本的な部分は同じなので、USM 3x3 の部分をコピーして貼り付ける。
- 平滑化カーネルを 3 種類使う。(5x5、9x9、13x13 の 3 つ)
- コントラスト依存非線形関数

$$f_m(S_m) = Y_m \times |S_m| \times \frac{\exp(X_m/S_m) - 1}{\exp(X_m/S_m) + 1}$$

$X_m$ : コントラスト依存を変化させるパラメーター  
 $Y_m$ : 加算比率を変化させるパラメーター  
 $S_m$ : 差分信号強度 (ボケマスクのピクセル値)

の計算を 1 ピクセルずつ実行する。



新たに使用する変数

Sm, Ym, Xm : コントラスト依存非線形関数変換に使用

新たに使用する関数

abs : 絶対値を返す

exp : 指数関数を返す

```
-----8<-----8<-----
macro 'MFP';
var
  Width, Height      : integer;
  Pv                 : real;
  Sm, Ym             : real;
  Xm                 : integer;
  x, y               : integer;
  pid1, pid2         : integer;
  str                 : string;

begin
  pid1 := PidNumber;
  GetPicSize(Width, Height);
  Xm := GetNumber('Xm:コントラスト依存パラメータ:', 30, 1);
  Ym := GetNumber('Ym:加算比率パラメータ:', 1, 0);
  str := concat('MFP Xm=', Xm, ' Ym=', Ym);
  Duplicate(str);
  pid2 := PidNumber;

  NewTextWindow('5x5 average', 120, 80);
  writeln(' 1 1 1 1 1');
  writeln(' 1 1 1 1 1');
  writeln(' 1 1 1 1 1');
  writeln(' 1 1 1 1 1');
  writeln(' 1 1 1 1 1');
  Convolve('');
  Dispose;
```

```

NewTextWindow('9x9 average',180,160);
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1');
Convolve('');
Dispose;

```

```

NewTextWindow('13x13 average',240,200);
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
writeln(' 1 1 1 1 1 1 1 1 1 1 1 1 1');
Convolve('');
Dispose;

```

```
ImageMath('subtract', pid1, pid2, 1, 128, pid2);
```

```

for y := 0 to Height - 1 do begin
  for x := 0 to Width - 1 do begin

```

```

    Sm := GetPixel(x,y) - 128;

```

```

    if Sm = 0 then Pv := 128 else
    Pv := Ym*abs(Sm) * ((exp(Xm/Sm) -1) /
        (exp(Xm/Sm) +1)) +128;
    PutPixel(x, y, Pv);

end;
end;

ImageMath('add', pid1, pid2, 1, -128, pid2);
end;

```

プログラムは大分大きくなってきました。しかし変更箇所は最初のプログラムと比較して、それほど多くないのではないのでしょうか。マクロメニューには USM 3×3 と USM 9×9、そして新たに MFP が加わり、マクロプログラムらしくなったのではないのでしょうか。今回のプログラムは 1 ピクセルずつ計算させるルーチンが入っているので計算に時間がかかりますが、これを機に本格的なプログラミングに取り組んでみてはいかがでしょうか。

USM 9×9 K=2



MFP

