画像処理 （4J）

第 8 回・補足資料 [2022-12-08]

1. ピクセルデータへのアクセスに関しての補足

IplImage 構造体の定義にあるように、imageData は 【char 型】 として宣言されています。一方で、格納するデータは IPL_DEPTH_8U で指定したように、符号無し 8 ビット整数 = 【unsigned char 型】として扱う必要があります。

単に char と宣言した場合、符号付き(signed char)となるか、符号なし(unsigned char)となるかは処理系に依存する[1]のですが、VC の場合は符号付きとして扱われるようです[2]。

従って、imageData からピクセル値を読み取る際には、型に気をつけなければなりません。

VC の場合、 (signed) char は -128～127 の値、unsigned char は 0～255 の値を表現できることになります。

Bit 表現:	0000 0000	0111 1111	1000 000	1111 1111
singed char:	0	127	-128	-1
unsigned char:	0	127	128	255

例えば、以下のコード

```
(適当な画像を IplImage* img に持っている状態で)
int x = 20;
int y = 10;
img->imageData[img->widthStep * y + x] = 200;
printf("%d¥n",img->imageData[img->widthStep * y + x]);
```

の実行結果は、(200 を代入したつもりなのに、)

```
-56
```

と表示されるはずです。ここで、最後の行を

```
printf("%d¥n",(unsigned char)img->imageData[img->widthStep * y + x]);
```

のように unsigned char 型にキャストすることで、200 が表示されるようになります。

---

[1] ここでは、char のサイズは 8bit として扱う(そして実際に殆どの場合は 8bit である)が、厳密には、char 型のサイズが 8bit とは限らない(処理系による)。

[2] デフォルトの場合。オプションで符号なしに切り替えることも可能らしい。

細かく見てみると、結構ややこしいことが行われています。まず、c 言語で単純に整数の数値を書いた場合(「200」など)、これは int 型 (符号付き) として扱われます。ですので、

```
img->imageData[img->widthStep * y + x] = 200;
```

の行では、左辺は unsigned char 型で扱える範囲が -128～127 なので、数値 200 は代入できない(オーバーフローする)ことになります。このような場合は、右辺の Bit 表現そのものを、左辺に入る分だけ代入することになります。

右辺の Bit 表現　　… 0000 0000 0000 0000 0000 0000 1100 1000 (32bit)

左辺に代入されたもの … 1100 1000 (8bit)

ということで、結果的には、左辺の変数に代入された値をそのまま(左辺の変数 imageData の型である)char 型として扱うと、「-56」という数値と解釈され、unsigned char 型として扱うと「200」という数値として解釈されることになります。char 型では扱えないはずの、「200」を代入していますが、“結果的には”unsigned char で解釈した場合の「200」が代入されていることになり、意図した通りにうまく動いているということになります[3]。(128～255 を代入した場合は、このような流れで“結果的に”見た目通りの動きとなる)

Bit 表現(内部表現)「1100 1000」は同じですが、数値としての意味が、char 型と解釈した場合と、unsigned char 型と解釈した場合とで異なる点に、注意が必要です。

このことは、ピクセル値を取得して、演算を行うような場合にも、問題になってきます。(※演算結果を 0～255 の数値にして、ピクセルに代入するような場合は、暗黙的に左辺の imageData の型である char 型に変換される過程で、前述と同様の流れとなるため、結果的にうまく動きます。

つまり、本来 imageData(char 型)で扱える -128～127 範囲を超える値を代入するような記述

```
img->imageData[img->widthStep * y + x] = 200;
```

等も、意図通り問題なく動きます。)

[3] なお、例えば下記のように 100 を代入した場合は、

```
img->imageData[img->widthStep * y + x] = 100;
```

左辺の char 型で扱える範囲の値であるため、数値の 100 として代入が行われる。char 型として数値の 100 が格納されるので、Bit 表現で [0110 0100] となるが、これは unsigned char 型で解釈しても、数値の 100 である。(0～127 を代入した場合は、このような理由でうまく動くことになる)

2. 型変換による影響の詳細（※詳細を理解したい人向け）

例えば、下記のようなコードを考えます。

```
char imageData0, imageData1, imageData2;
imageData0 = 200;

float f1 = imageData0:                          // 実数型に代入
float f2 = (unsigned char)imageData0;      // 実数型に代入
imageData1 = f1:     // 実数型から代入
imageData2 = f2;     // 実数型から代入

printf("%f, %f, %u, %u¥n", f1, f2, (unsigned char)f1, (unsigned char)f2);
printf("%d, %d, %u, %u¥n", (int)imageData1, (int)imageData2
                          ,(unsigned int)imageData1, (unsigned int)imageData2);
printf("%d, %u, %d, %u¥n", imageData1, imageData1
                          , (unsigned char)imageData1, (unsigned char)imageData1);
```

実行結果は、以下のようになります。

```
-56.000000. 200.000000. 200. 200          ・・・(1)
-56. -56. 4294967240. 4294967240          ・・・(2)
-56, 4294967240, 200, 200                 ・・・(3)
```

まず、imageData0 = 200; の時点で、先に述べたオーバーフローが起きるため、内部表現は[1100 1000]となっています。

(1) unsigned char 型にキャストしないと、内部表現[1100 1000]が数値「-56」と解釈された上で代入されるため、実際のピクセル値が得られていないことがわかります(f1=-56.0)。f2 は正しくキャストしているので、実際のピクセル値が得られています(f2=200.0)。unsigned char にキャストすると、数値「-56.0」と「200.0」が内部表現[1100 1000]と[1100 1000]に変換されることになります。これを unsigned int 型として表示(%u)すると、どちらも 200 と解釈されることになります。

(2) まず、f1=-56.0、f2=200.0 という値が char 型の変数 imageData1、imageData2 に代入される際に、整数-56 と 200 に解釈された上で、内部表現が[1100 1000]と[1100 1000]となります。これを char 型で解釈するので、どちらも整数 -56 を意味することになります。

これを int でキャストした場合、整数-56 を int 型で表現するように変換するため、内部表現がどちらも[1111 1111 1111 1111 1111 1111 1100 1000]と変換されます。これは、int 型として表示(%d)すると、-56 となります。unsigned int 型に変換する場合も、内部表現がどちらも[1111 1111 1111 1111 1111 1111 1100 1000]と変換されるため、これを unsigned int 型として表示(%u)すると、4294967240 となります。

(3) (1)について述べた通り、imageData1 の内部表現は[1100 1000]となっています。これは-56 を意味しているので、int 型に(暗黙的に)変換された結果、内部表現が[1111 1111 1111 1111 1111 1111 1100 1000]となります。これを int 型として表示(%d)した場合、-56 と解釈され、unsigned int 型として表示(%u)した場合には、4294967240 と解釈されることになります。

unsigned char にキャストした場合は、内部表現[1100 1000]が 200 と解釈されるので、int 型に(暗黙的に)変換された結果、内部表現が[0000 0000 0000 0000 0000 0000 1100 1000]となります。これは int 型として表示(%d)しても、unsigned int 型として表示(%u)しても、200 と解釈されることになります。

※このページの文章は、ちょっと適当です。整数拡張に関して等、次頁の記述と合わせて見てみて下さい。

3. 対処法

imageData から、ピクセル値を取得する際には、必ず unsigned char 型に変換(キャスト)する。

unsigned char に代入する場合だけでなく、

```
unsigned char p = (unsigned char)img->imageData[img->widthStep * y + x];
```

int に代入する場合や、

```
int i          = (unsigned char)img->imageData[img->widthStep * y + x];
```

float や double に代入する場合

```
float f        = (unsigned char)img->imageData[img->widthStep * y + x];
double x       = (unsigned char)img->imageData[img->widthStep * y + x];
```

なども、キャストが必要です。

●具体例で示します。(x,y)の**ピクセル値は 200** とします。

```
○ (1) int i1 = (unsigned char)img->imageData[img->widthStep * y + x];
× (2) int i2 = img->imageData[img->widthStep * y + x];
```

(1)の場合、内部表現[1100 1000]を unsigned char として解釈するため、数値「200」が右辺値となり、i1 には**「200」が代入**されます。

(2)の場合、内部表現[1100 1000]が imageData の変数型である char で解釈されるため、数値「-56」が右辺値となり、i2 には**「-56」が代入**されてしまいます。

ちなみに、ピクセル値が 127 以下の場合は、内部表現を char で解釈しても、unsigned char で解釈しても同じ値となるため、" 0～127 のピクセル値しか扱わない場合 " には、不具合は生じません。そのため、バグに気が付きにくい場合があります。

●printf()で、ピクセル値を表示したい場合なども同様で、(x,y)の**ピクセル値を 200 とする**と、

```
○ (1) printf("%u", (unsigned char)img->imageData[img->widthStep * y + x]);
× (2) printf("%u", img->imageData[img->widthStep * y + x]);
× (3) printf("%d", img->imageData[img->widthStep * y + x]);
```

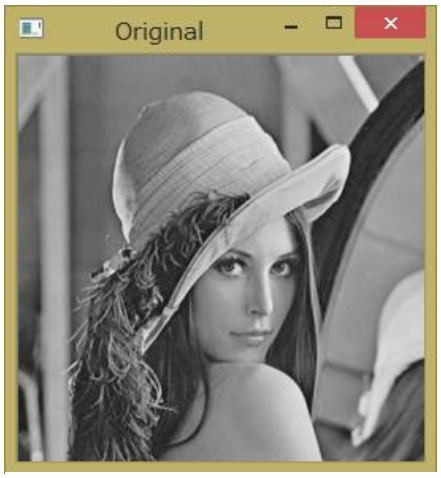
(1)の場合、内部表現[1100 1000]を unsigned char 型として解釈するため「200」を意味することになり、これは int 型で[0000 0000 0000 0000 0000 0000 1100 1000]と内部部表現される(整数拡張)。この内部表現を unsigned int 型として解釈して表示(%u)するため正しく**「200」が出力される。**

(2)の場合、内部表現[1100 1000]が imageData の変数型である char で解釈されるため、「-56」を意味することになり、これが int 型で[1111 1111 1111 1111 1111 1111 1100 1000]として内部表現される(整数拡張)。この内部表現を unsigned int 型として解釈して表示(%u)するため、**「4294967240」が出力される。**

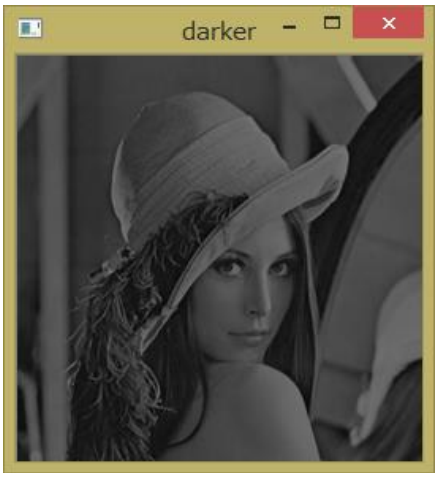
(3)(整数拡張された)int 型の内部表現は(2)と同じだが、これを int 型として解釈して表示(%d)するため、**「-56」が出力される。**

【ピクセルデータへのアクセスと演算(※グレイスケール画像の場合)】

```
// Sample Program [p01-d.cpp]グレイスケール画像を暗くして表示(ピクセル値を 1/2 にする)

#include <stdio.h>
#include <opencv/highgui.h>

void main(int argc, char* argv[])
{
   if (argc > 1) {   // 起動オプションでファイル名が指定されている場合のみ
      IplImage* img;
      if ((img = cvLoadImage(argv[1], CV_LOAD_IMAGE_UNCHANGED)) != NULL) { // 画像が読
み込めない(NULL)場合は終了
         cvNamedWindow("Original");
         cvShowImage("Original", img);

         float f;

         for (int y = 0; y < img->height; y++) {
            for (int x = 0; x < img->width; x++) {
               f = (unsigned char)img->imageData[img->widthStep * y + x];
               // f = img->imageData[img->widthStep * y + x]; // キャストを忘れた場合

               img->imageData[img->widthStep * y + x] = f / 2;
            }
         }
         cvNamedWindow("darker");
         cvShowImage("darker", img);

         cvWaitKey(0);  // 何かキーが押されるまで待つ

         cvDestroyAllWindows();
         cvReleaseImage(&img);
      }
   }
}
```



(図 1) 元画像



(図 2) キャストした場合の処理結果

(図 3) キャストを忘れた場合の処理結果