

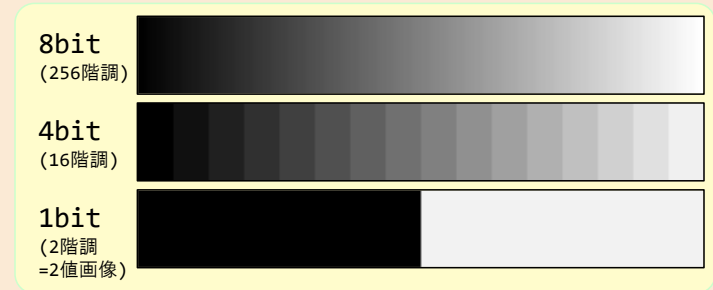
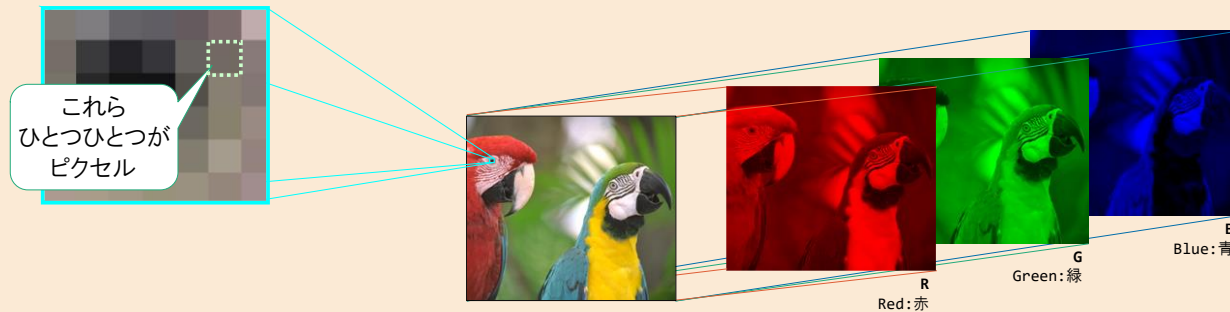
Moodleの
出席確認を
提出しておいて
下さい。

VisualStudio2019(等)で、
C言語+OpenCV のコーディングができる状態に
準備してください。

画像処理 (4J)

第13回

- ラスタ画像とベクタ画像 ... この授業では、ピクセル情報の集合であるラスタ画像を扱う
- 解像度 ... 画像の大きさ(細かさ)
- ピクセル(画素) ... ラスタ画像を構成する1つの点
- チャンネル ... 1ピクセルをいくつかの値で表現するか (例:RGBの3ch)
- 階調数 ... 濃度を何段階で表現するか (例:8bit(=256段階))



デジタル写真 = 有限の解像度で空間的にサンプリング(標本化)し、
有限の階調値で明るさを表現(量子化)したもの ...と捉えることができる。

※音声信号のデジタル化と対応させると、サンプリング周波数が解像度に、量子化bit数が階調数に、チャンネル数はそのまま対応する

第7回のまとめ

12

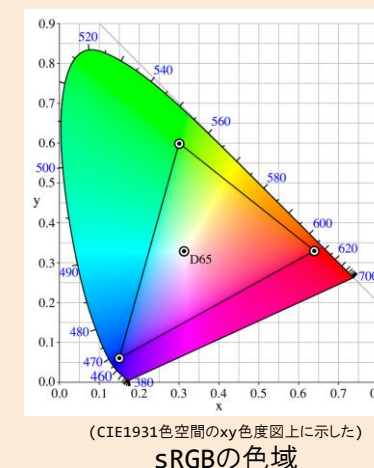
●グレースケール画像とカラー画像

- グレースケール画像は1つの (x, y) 座標点に1つの濃度値 $g(x, y)$
- RGBカラー画像は、1つの座標点に、3つの濃度値



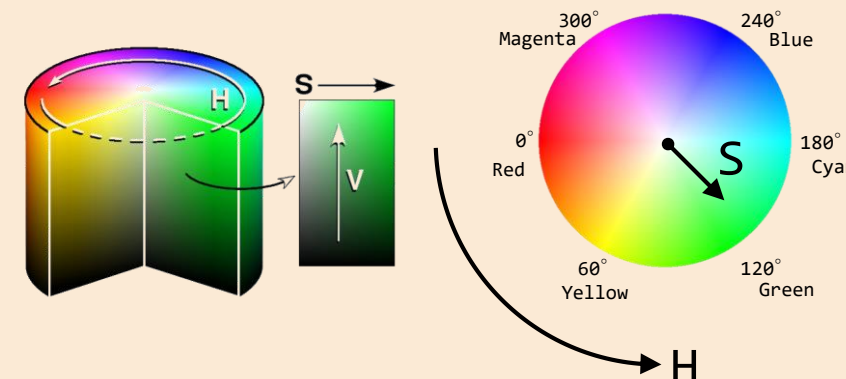
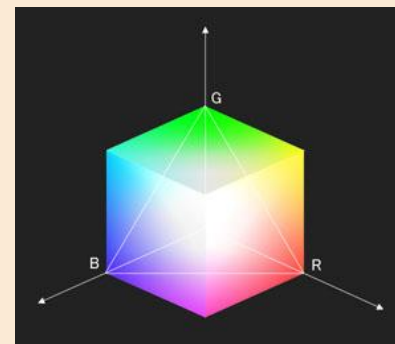
●RGBカラー画像

- RGB値が同じでも、同じ色が表示されるとは限らない
- sRGBに準拠させれば、一貫した色表現が可能。
(ただし表現できる色域が狭い)



●色空間: RGBとHSV

- 相互に変換可能
- 他にも様々な表色系がある

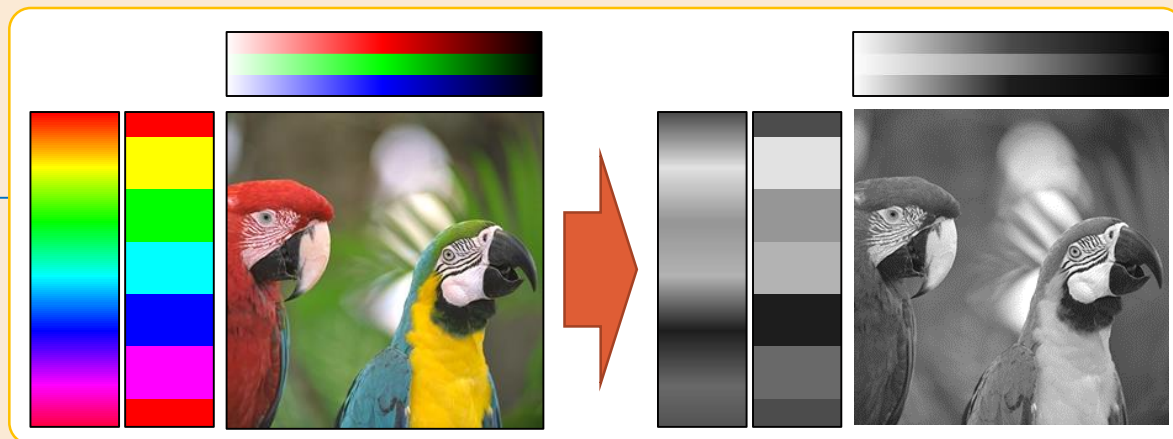


第8回まとめ

●グレイスケール化

- NTSC加重平均法がよく使われる

$$Y = (0.298912 \times R + 0.586611 \times G + 0.114478 \times B)$$



●二値化

- 閾値を堺に、 $\{0,1\}$ の二値の画像に変換
- 閾値は任意に決められるが、画像統計量から閾値を自動決定する方法として大津の方法(判別分析法)が有名。



第9回まとめ

原画像



階調反転 14

●濃度変換 …… 濃度値を一定の方法で新しい濃度値に変換

➤線形変換 (Linear Stretch)

$$output = input \times a + b$$

➤ガンマ変換 (Gamma Stretch)

$$output = 255 \times \left(\frac{input}{255}\right)^{1/\gamma}$$

➤輝度調整、コントラスト調整、階調反転などに利用可能



輝度



コントラスト



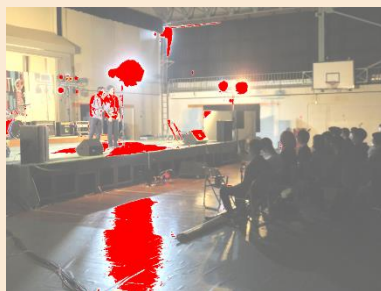
ガンマ

●濃度変換に伴う画像の劣化

- 白飛び …… 変換後に最大値以上になった場合に、最大値にクリップされる
- 黒つぶれ …… 変換後に最小値以下になった場合に、最小値にクリップされる
- 階調飛び(トーンジャンプ) …… 中間値の階調が失われ、濃度値が不連続に変化



白飛び



黒つぶれ



階調飛び

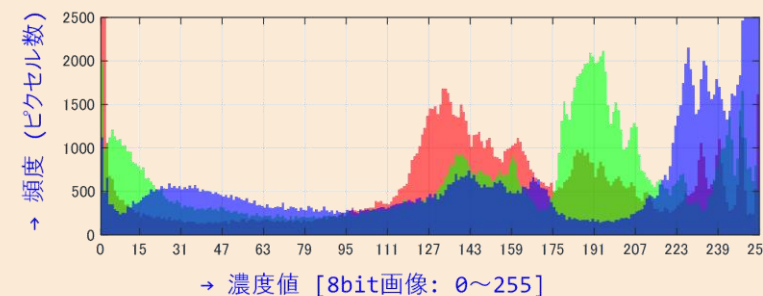


第10回まとめ

15

●ヒストグラム

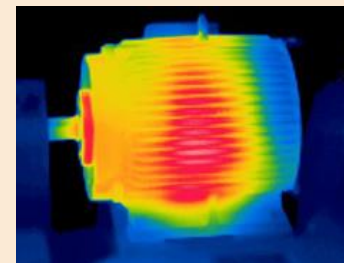
- 濃度値の頻度 (各濃度値が画像中にいくつあるか) を示したもの
- ヒストグラムの形状から、画像の性質がある程度わかる



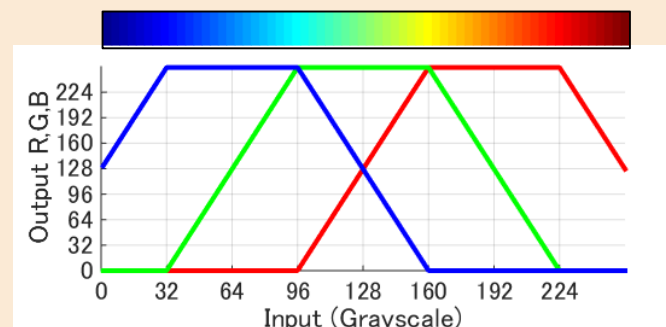
●濃度変換(2)

① 疑似カラー

- グレースケール値に色 (RGB値) を対応付けて表すもの
- 対応関係を示したもの: カラーマップ

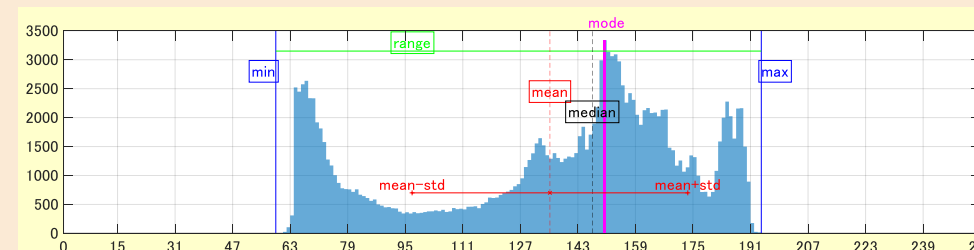


② ヒストグラム平坦化



●画像統計量

- 最大/最小/最頻
- 平均/中央
- 範囲/分散/標準偏差

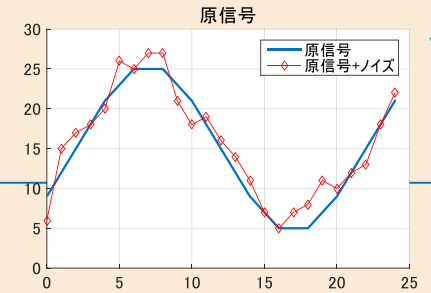
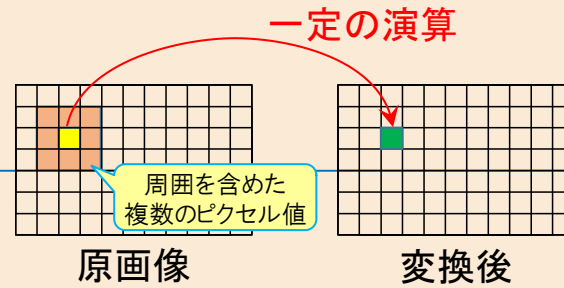


第11回まとめ

16

●近傍演算とは？

- 注目ピクセルの近傍(周囲)を含めた複数のピクセル値を用いて、新たなピクセル値を計算

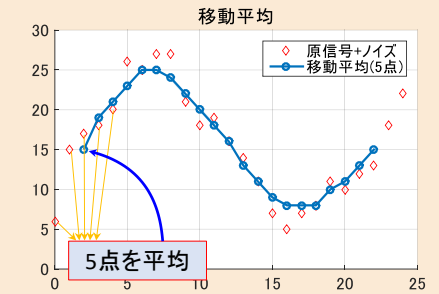
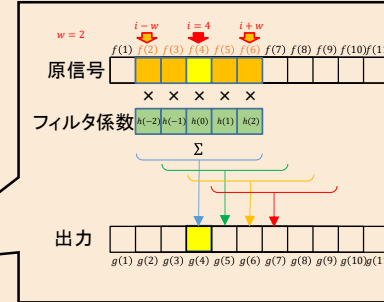


●畳み込み積分

- 1次元信号の畳み込み積分
$$g(i) = \sum_{n=-w}^w f(i+n)h(n)$$

- 単純移動平均 $\dots h(n) = \frac{1}{2w+1}$

- ガウシアンフィルタ(加重平均の一種) $\dots h(n) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(n-\mu)^2}{2\sigma^2}}$

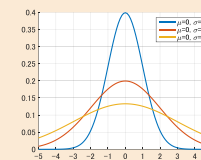


- 2次元信号(=画像)の畳み込み積分
$$g(x,y) = \sum_{n=-w}^w \sum_{m=-w}^w f(x+m,y+n)h(m,n)$$

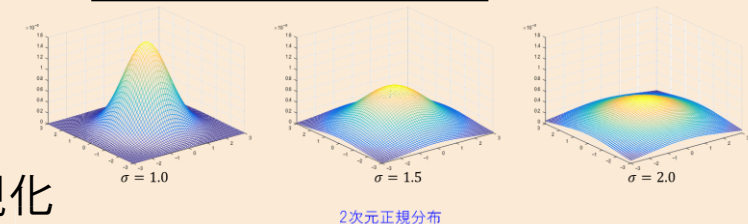
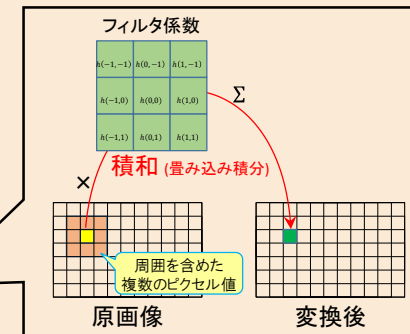
- 単純移動平均 $\dots h(n) = \frac{1}{(2w+1)^2}$

- ガウシアンフィルタ(加重平均の一種) $\dots h_1(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{m^2+n^2}{2\sigma^2}\right)}$

の二次元正規分布に比例し、フィルタ係数の総和が1.0になるように正規化



ガウス分布



2次元正規分布

(加重)移動平均することで、平滑化された(=高周波成分が低減された)信号になる。
すなわち、一種のLPF(Low Pass Filter: 低域透過フィルタ)として働く。

第12回のまとめ

17

●輪郭抽出

- 輪郭とは、「ピクセル値が急激に変化しているところ」
- 微分により、輪郭抽出ができる
- 離散信号の微分は、差分を取るだけ → $f'(i) = f(i+1) - f(i)$
- 画像処理としては、近傍演算(畳み込み積分)で実装可能
- 一次微分のPrewittフィルタ/Sobelフィルタ、二次微分のLaplacianフィルタなどがよく使われる



●鮮鋭化

- 鮮明(シャープ)な画像とは、「輪郭部分でピクセル値が急激に変化している」画像
- 元画像から、二次微分(ラプラシアンフィルタ)した画像を減算することで、変化を強調することができる
⇒ フィルタ係数同士の演算で得られたフィルタ係数で、上記の処理を同時に行うことができる。



$i-1$	i	$i+1$		$i-1$	i	$i+1$	
-1	0	1	$j-1$	-1	-1	-1	$j-1$
-1	0	1	j	0	0	0	j
-1	0	1	$j+1$	1	1	1	$j+1$
横方向に微分 縦方向に平滑化				横方向に平滑化 縦方向に微分			

Prewittフィルタ
[※一次微分]

$i-1$	i	$i+1$	
0	1	0	$j-1$
1	-4	1	j
0	1	0	$j+1$

4近傍のLaplacianフィルタ
[※二次微分]

$i-1$	i	$i+1$		$i-1$	i	$i+1$	
-1	0	1	$j-1$	-1	-2	-1	$j-1$
-2	0	2	j	0	0	0	j
-1	0	1	$j+1$	1	2	1	$j+1$
横方向に微分 縦方向に平滑化				横方向に平滑化 縦方向に微分			

Sobelフィルタ
[※一次微分]

原画像		ラプラシアンフィルタ		鮮鋭化フィルタ						
0	0	0		0	-1	0				
0	1	0	-	1	-4	1	=	-1	5	-1
0	0	0		0	1	0		0	-1	0

鮮鋭化フィルタは、フィルタ係数の演算で得ることができる

近傍演算(3)

メディアンフィルタ

メディアアンフィルタ

畳み込み積分ではない近傍演算

- 前回、前々回で、畳み込み積分で表現できるさまざまな画像処理を学びました。

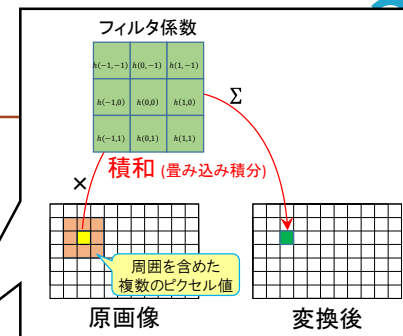
- ぼかしフィルタ（ガウシアンフィルタ等）
- エッジ検出（ラプラシアンフィルタ等）
- 鮮鋭化フィルタ

- 近傍演算でできる画像処理は、畳み込み積分だけか？？

畳み込み積分

$$g(x, y) = \sum_{n=-w}^w \sum_{m=-w}^w f(x + m, y + n) h(m, n)$$

畳み込み積分の場合、
固定のフィルタ係数との積を求めて和を取るだけ



畳み込み積分ではない近傍演算

- 畳み込み積分で表現できない近傍演算？
- 例えば、第10回で扱った画像統計量を近傍のピクセル内で求めることを考える

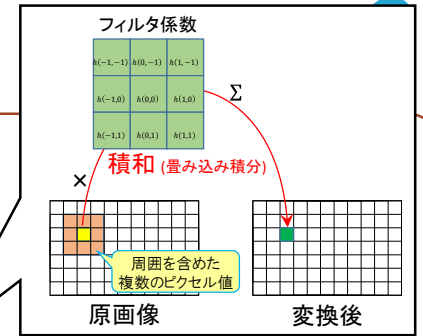
畳み込み積分で表現が...

- 最大値 (できる / できない)
- 最小値 (できる / できない)
- 最頻値 (できる / できない)
- 平均値 (できる / できない)
- 中央値 (できる / できない)
- 範囲 (できる / できない)
- 分散 (できる / できない)
- 標準偏差 (できる / できない)

畳み込み積分


$$g(x, y) = \sum_{n=-w}^w \sum_{m=-w}^w f(x+m, y+n)h(m, n)$$

畳み込み積分の場合、
固定のフィルタ係数との積を求めて和を取るだけ




- 近傍の中央値を、変換後のピクセル値とするもの
 - ※ 中央値 \neq 平均値 なので注意

110	120	130
180	190	140
170	160	150



150


10	20	30
220	230	40
210	200	150



150


3×3の場合、
ピクセル値の順に並べた際に
5番目に来たピクセルの値にする

0	100	100
200	255	100
200	200	150




150

0	0	0
0	255	0
0	0	0



0

0	0	10
255	255	10
10	10	10



10

- 結果的に、極端に大きな値や、小さな値は無視される

●ごま塩ノイズ (Salt & Pepper Noise) (インパルスノイズ(impulse noise)とも)

- ランダムな位置のピクセルが 0 または 255 に置き換わってしまうノイズ
- 第11回の授業の Tool-11 でノイズ付加が可能(**ノイズ種類=1**)

• 例:



```
>Tool-11.exe LENNA.bmp 1  
argc = 3  
argv[0] = tool-11  
argv[1] = LENNA.bmp  
argv[2] = 1  
  
Noise Type = 1 (Salt&PepperNoise)  
Noise Lv = 0.050000
```

Noise Level = 0.05
(デフォルト)



```
>tool-11 LENNA.bmp 1 0.1  
argc = 4  
argv[0] = tool-11  
argv[1] = LENNA.bmp  
argv[2] = 1  
argv[3] = 0.1  
  
Noise Type = 1 (Salt&PepperNoise)  
Noise Lv = 0.100000
```

Noise Level = 0.1



演習： メディアアンフィルタの作成

OpenCVを使った画像生成の流れ



30

【画像生成時の流れ】

```
IplImage* img = cvCreateImage(CvSize size, IPL_DEPTH_8U, int channels);
```

↓

… 画像を扱うための構造体 `img` を生成する

```
cvSetZero(img);
```

… 画像データ `img` を 0(=黒) で初期化

↓

↓

【画像読み込み時の流れ】

↓

```
IplImage* img = cvLoadImage(const char* filename, CV_LOAD_IMAGE_UNCHANGED);
```

↓

↓

… 画像ファイルを読み取り、画像データ `img` を生成

↓

↓

<`img` に対する何らかの処理>

↓

↓

```
cvSaveImage(img);
```

… 画像データ `img` を画像ファイルとして保存

↓

```
cvReleaseImage(&img);
```

… 画像を扱うための構造体 `img` に割り当てたメモリの開放

各種関数のリファレンス(1)



31

```
IplImage* img  
= cvCreateImage(CvSize size, int depth, int channels);
```

- size: 画像のサイズ。
- depth: ピクセルのデータ形式。
 - ※本授業では常に IPL_DEPTH_8U (符号無し8ビット整数 = unsigned char)とする。
- channels: ピクセル毎のチャンネル数。[グレイスケール = 1 , カラー = 3]

※ロードに失敗した場合は NULL が返る。
内部でmalloc()されているので、cvReleaseImage()で開放する必要がある。

```
typedef struct CvSize {  
    int width;    /* 横幅 */  
    int height;   /* 高さ */  
} CvSize;
```

各種関数のリファレンス(2)



32

```
void cvSetZero(IplImage *img);
```

- `img`: `cvCreateImage()` が返した `IplImage*` のアドレス。
全ピクセルデータを 0(黒)で初期化する

```
IplImage* img  
= cvLoadImage(const char* filename, int iscolor);
```

- `filename`: ファイル名。対応ファイル形式は(表1)を参照。
- `iscolor`: 読み込む画像のカラーの種類。
※本授業では常に `CV_LOAD_IMAGE_UNCHANGED` とする。

指定した画像ファイルを `IplImage` 形式に読み込む

※内部で `malloc()` されているので、`cvReleaseImage()` で開放する必要がある。

各種関数のリファレンス(3)



33

```
int cvSaveImage(const char* filename, IplImage* image);
```

- filename: ファイル名。拡張子で保存形式が決まる。→ (表1)を参照。
- image: 保存する画像データ
IplImage を、画像ファイルとして保存する。

※保存に成功した場合は 1 、失敗した場合は 0 が返る(らしい)。

```
void cvReleaseImage(IplImage** img);
```

- img: cvCreateImage() が返した IplImage* のアドレス。
cvCreateImage()やcvLoacImage()で確保された領域を開放する。

(表 1) cvLoacImage()、cvSaveImage() の対応形式と、指定する拡張子

形式	Windows bitmaps	Jpeg	Portable Network Graphics	Portable image format	Sun rasters	TIFF files	OpenEXR HDR images	JPEG 2000 images
拡張子	BMP,DIB	JPEG, JPG,JPE	PNG	PGM,PGM PPM	SR,RAS	TIFF, TIF	EXR	Jp2

IplImage 構造体 (types_c.h 内で定義) 再

34

```
typedef struct _IplImage
{
    int    nSize;           /* sizeof(IplImage) */
    int    ID;              /* version (=0) */
    int    nChannels;        /* Most of OpenCV functions support 1,2,3 or 4 channels */
    int    alphaChannel;     /* Ignored by OpenCV */
    int    depth;           /* Pixel depth in bits: IPL_DEPTH_8U, IPL_DEPTH_8S, IPL_DEPTH_16S,
                             IPL_DEPTH_32S, IPL_DEPTH_32F and IPL_DEPTH_64F are supported. */
    char    colorModel[4];   /* Ignored by OpenCV */
    char    channelSeq[4];   /* ditto */
    int    dataOrder;        /* 0 - interleaved color channels, 1 - separate color channels.
                             cvCreateImage can only create interleaved images */
    int    origin;          /* 0 - top-left origin,
                             1 - bottom-left origin (Windows bitmaps style). */
    int    align;           /* Alignment of image rows (4 or 8).
                             OpenCV ignores it and uses widthStep instead. */
    int    width;           /* Image width in pixels. */
    int    height;          /* Image height in pixels. */
    struct _IplROI *roi;     /* Image ROI. If NULL, the whole image is selected. */
    struct _IplImage *maskROI; /* Must be NULL. */
    void    *imageId;        /* " */
    struct _IplTileInfo *tileInfo; /* " */
    int    imageSize;        /* Image data size in bytes
                             (==image->height*image->widthStep
                             in case of interleaved data) */
    char    *imageData;      /* Pointer to aligned image data. */
    int    widthStep;        /* Size of aligned image row in bytes. */
    int    BorderMode[4];    /* Ignored by OpenCV. */
    int    BorderConst[4];   /* Ditto. */
    char    *imageDataOrigin; /* Pointer to very origin of image data
                             (not necessarily aligned) -
                             needed for correct deallocation */
}
IplImage;
```


- `IplImage` のメンバ変数の `nChannels`

- 3の場合カラー画像

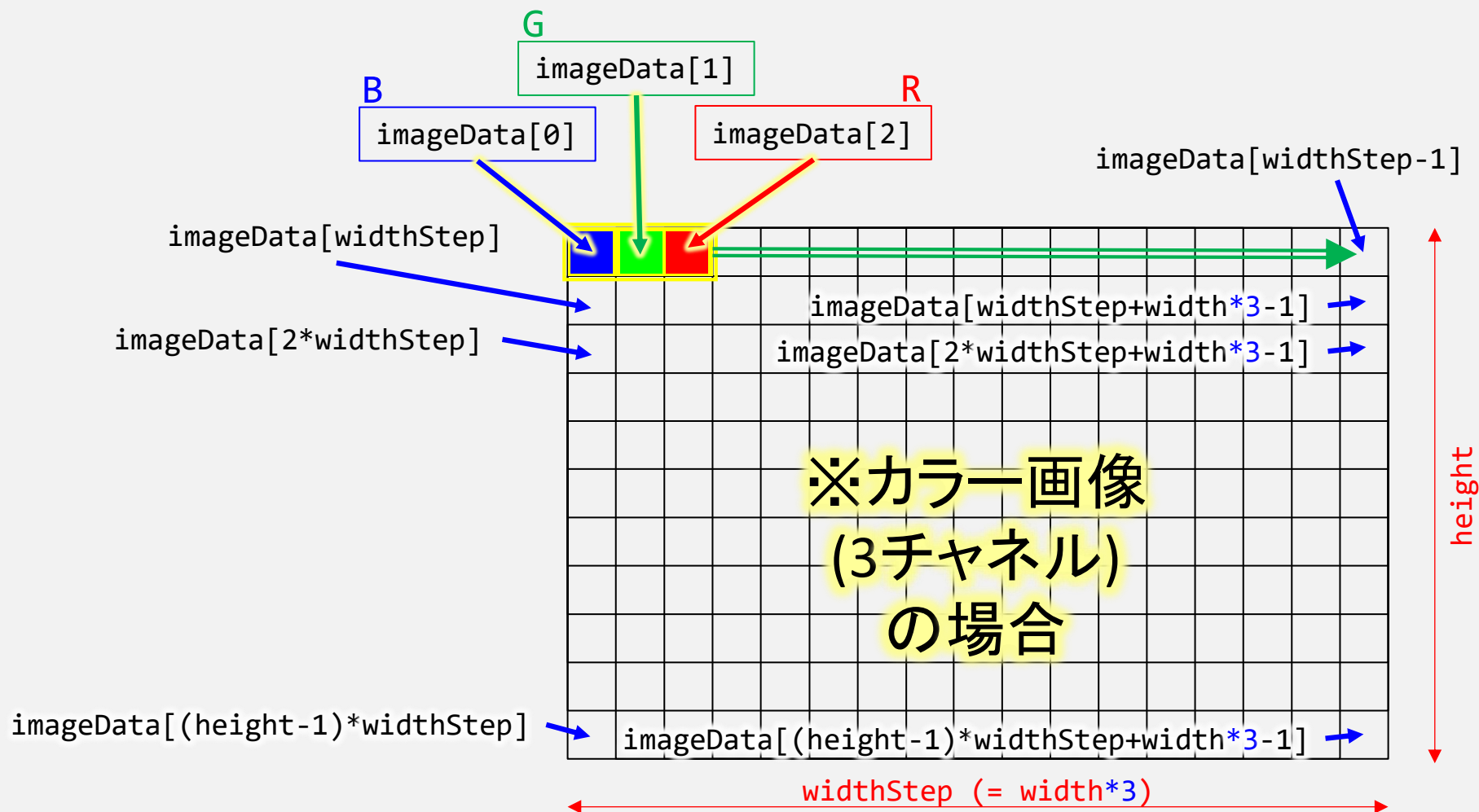
- 1の場合グレースケール画像

- (※本授業では、`nChannels`が1か3の場合のみ、取り扱うものとする)

Ip1ImageのRGB値へのアクセス 再

36

RGBカラー画像の個々のRGB値は、
下図のような順に一次元配列 `imageData[]` に格納される。



IplImageのRGB値へのアクセス

再 (変更後) 37

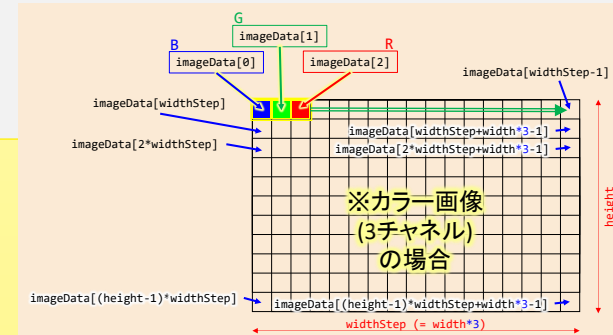
- IplImage のメンバ変数を用いて、個々のピクセルへアクセスする。
- imageDataには、
BGRBGRBGRBGR.....の順で格納されていることに注意 (**RGBの順ではない!**)。
 - char* imageData ... 画像データへのポインタ
 - int widthStep ... 画像データ1ライン分のバイト数(= char で数えた数)

【例】

IplImage *img の画像(RGBカラー画像)に対して、
座標点 (x, y) のカラーチャンネルごとのピクセル値(RGB値)へは、

```
b = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 0];  
g = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 1];  
r = (unsigned char)img->imageData[img->widthStep * y + x * 3 + 2];
```

としてアクセスすることが出来る。



●注意(ヒント)

- カラー画像(特にpng画像)を読み込んだのに、 `nChannels == 3` でない場合、インデックスカラーや、透明チャンネルあり(つまり `nChannels == 4`)の場合があります。
- そのような場合は、ペイント等で開いて、BMP形式で保存したものを用いて下さい。

演習 & 課題No.13

課題 No.13 : メディアンフィルタの作成

```
// No.13 : メディアンフィルタ (OPE_SIZE = 1 => 3x3 サイズ, OPE_SIZE = 2 => 5x5 サイズ)
#include <stdio.h>
#include <opencv/highgui.h>

#define OPE_SIZE 2

// プロトタイプ宣言
IplImage* subImage(IplImage* subimg, IplImage* img, int x, int y, int w, int h);
unsigned char medianFilter(IplImage* subimg, int ch);
void filterImage(IplImage* img1, IplImage* img2);

// 画像の一部(画像ブロック)を切り出す
IplImage* subImage(IplImage* subimg, IplImage* img, int x, int y, int w, int h) {
    // いったん点で初期化(imgの幅や高さを超えた場合の対策)
    cvSetZero(subimg);

    for (int i = y; i < y + h; i++) {
        for (int j = x; j < x + w; j++) {
            subimg->imageData[i * img->widthStep + j] = img->imageData[i * img->widthStep + j] * img->nChannels + ch;
        }
    }

    return subimg;
}

// メディアン値(中央値)を求める (カラー→グレースケール変換のため、ch を指定、0 指定の場合は各チャンネル毎に処理する)
// 各チャンネル毎に処理 (channels) の場合は、ch は各チャンネル
unsigned char medianFilter(IplImage* subimg, int ch) {
    unsigned char median = 0;
    unsigned char neighbor[OPE_SIZE * 2 + 1];
    int i = 0;

    // 近傍のピクセル値を neighbor[] に格納
    for (int x = 0; x < subimg->width; x++) {
        for (int y = 0; y < subimg->height; y++) {
            neighbor[i++] = (unsigned char)subimg->imageData[y * subimg->widthStep + subimg->nChannels * x + ch];
        }
    }

    // =====
    // 【ここを作成!!】
    // =====

    return median;
}

void filterImage(IplImage* img1, IplImage* img2) {
    IplImage* buff = cvCreateImage(cvSize(OPE_SIZE * 2 + 1, OPE_SIZE * 2 + 1), img1->depth, img1->nChannels);

    for (int y = 0; y < img1->height - OPE_SIZE - 1; y++) {
        for (int x = 0; x < img1->width - OPE_SIZE - 1; x++) {
            for (int ch = 0; ch < img1->nChannels; ch++) { // グレースケールカラー変換のためループ
                buff->imageData[y * buff->widthStep + x * buff->nChannels + ch] = medianFilter(subImage(buff, img1, x - OPE_SIZE, y - OPE_SIZE, OPE_SIZE * 2 + 1, OPE_SIZE * 2 + 1), ch);
            }
        }
    }

    // フィルタ結果の画像書き出し(ファイル名も自動生成)
    void writeIplImage(IplImage* img, const char* fileName) {
        char ext_p;

        strcpy_s(fileName, 256, img->filename); // 読み込んだ画像のファイル名をコピー
        ext_p = strrchr(fileName, '.');
        next_s = ".tif"; // ファイル名から拡張子を決めず
        sprintf(fileName, 256, "%s%s", fileName, ext_p); // ファイル名を付加したファイル名を生成
    }

    cvSaveImage(fileName, img);
}

void main(int argc, char* argv[]) {
    IplImage* img1;
    IplImage* img2;

    // ファイルを Drag&Drop で取得できるようにするには、①この下の一連のコメントアウトを消し、
    // char fn[256];

    // 数値オプションのチェック
    printf("Usage : %s\n", argv[0]);
    for (int k = 0; k < argc; k++) {
        printf("Arg[%d] = %s\n", k, argv[k]);
    }
    printf("Welcome");

    if (argc < 2) {
        printf("ファイル名を指定してください。No.");
        return;
    }
    strcpy_s(fn, 256, argv[1]);

    char fn1[] = "LennaSaltPepperNoise.bmp"; // Drag&Drop で取得する場合は、②ここをコメントアウトする

    // 画像データの読み込み
    if ((img1 = cvLoadImage(fn, CV_LOAD_IMAGE_ANYDEPTH | CV_LOAD_IMAGE_ANYCOLOR)) == NULL) { // 読み込んだ画像はカラーの場合も、グレースケール画像の場合もある
        printf("画像ファイルの読み込みに失敗しました。No.");
        return;
    }

    // 読み込んだ画像の表示
    cvNamedWindow("Original");
    cvShowImage("Original", img1);
    printf("File Name = %s\n", fn);
    printf("OPE_SIZE = %d, Size = %d x %d\n", OPE_SIZE, OPE_SIZE, OPE_SIZE * 2 + 1, OPE_SIZE * 2 + 1);
    img2 = cvCreateImage(cvSize(img1->width, img1->height), img1->depth, img1->nChannels); // 読み込んだ画像と同じ大きさの画像を生成

    // =====
    cvSetZero(img2); // 0(黒)で初期化しておく
    filterImage(img1, img2);
    cvNamedWindow("Median Filter");
    cvShowImage("Median Filter", img2);

    writeIplImage(img2, fn, ".Median.tif"); // ファイル結果を画像ファイルとして出力

    // =====
    cvWaitKey(0);
    cvDestroyAllWindows();
}
```

(1) 以下のメディアン値(中央値)を求める関数を完成させる

unsigned char medianFilter(IplImage* subImg, int ch)

--- ソースコードの掲載は、この関数のみで良い

(2) 第11回の授業の Tool-11 を用いて、適当な画像に Solt&Pepper ノイズを付加し、それに対してメディアンフィルタを適用する。

① 結果について考察する

② ガウシアンフィルタによるノイズ除去と比較して考察する

- Solt&Pepperノイズ × ガウシアンフィルタ
- Solt&Pepperノイズ × メディアンフィルタ
- ホワイトノイズ × ガウシアンフィルタ
- ホワイトノイズ × メディアンフィルタ

のそれぞれで比較する等すること。

--- 指定したノイズレベル、ノイズ付加前の元画像、ノイズ付加後の画像、フィルタのパラメータ(フィルタサイズも)、フィルタ後の画像 を必ず掲載して説明する