画像処理2019—3

画像のディジタル化

パターン認識過程

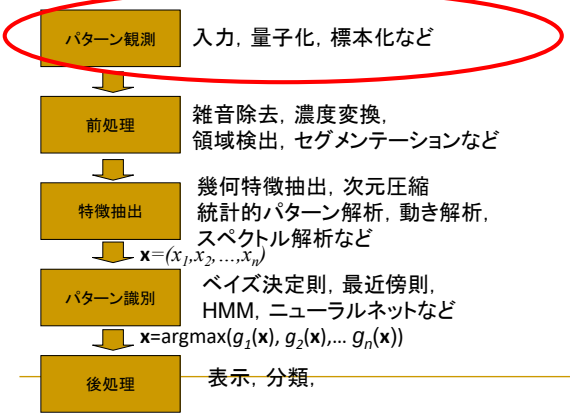


図1.10　デジタル画像の表現

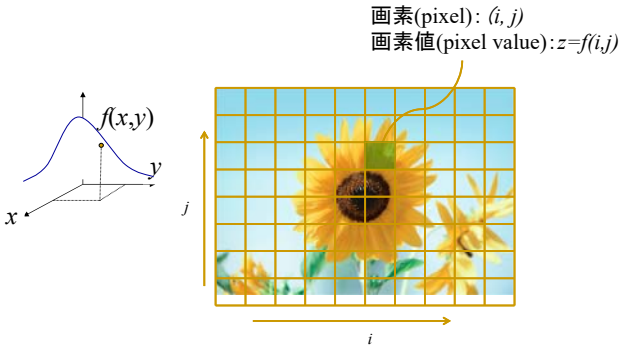
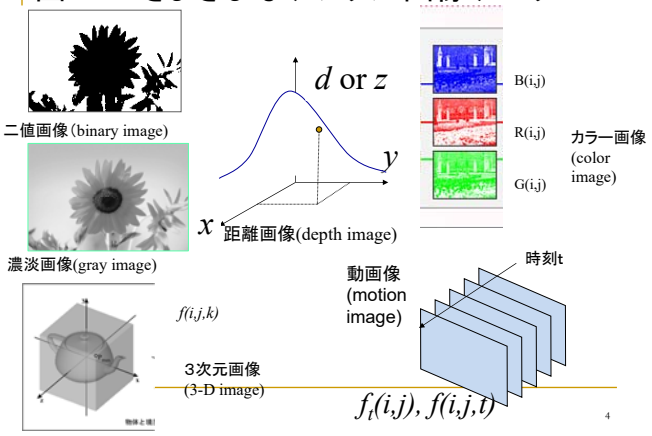
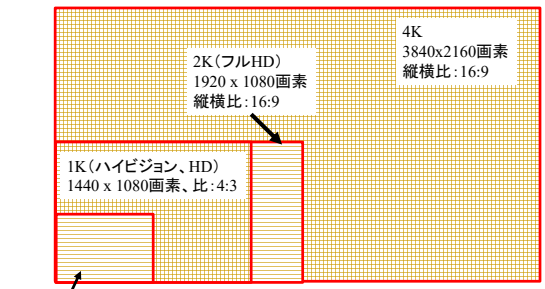


図1.11　さまざまなデジタル画像データ



画像の大きさと呼び名（解像度、画素数）



SD: standard density
HD: high density
HDTV：high definition TV(NHK用語)

図1.12　画像のプロファイル（断面）

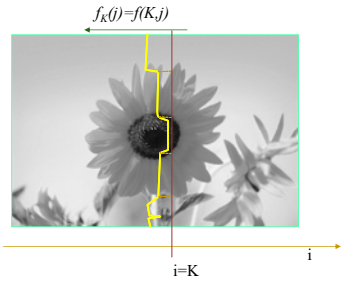
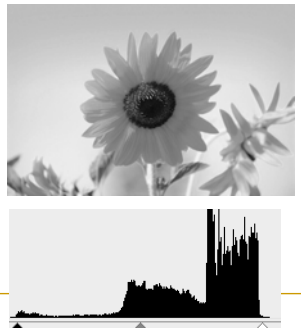
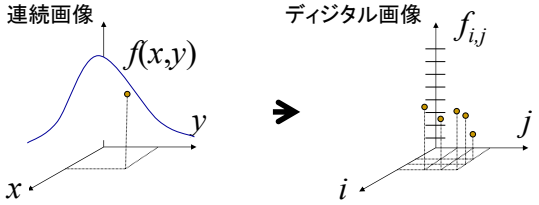


図1.13　画像のヒストグラム



1．画像のディジタル化

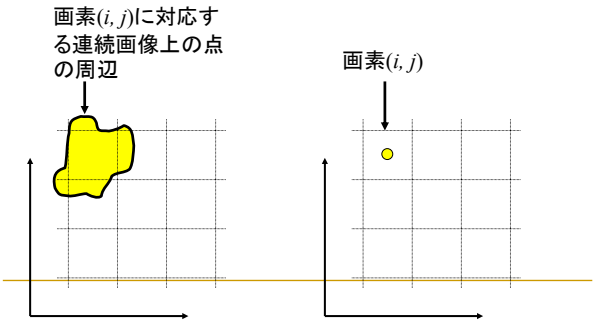
・標本化（Sampling）
　画像全体に離散的な標本点（Sampling Point）を配置する
・量子化（Quantization）
　各標本点（画素）の濃度値を離散的な有限個のレベルに割り当てる



標本点i,jにおける画素の濃度値
（明るさ、輝度値ともいう:f_{ij}でも表す）

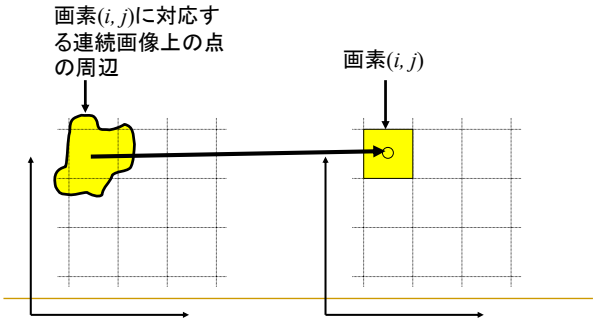
標本化

連続画像上の点(x, y)周辺にひろがっている濃淡から，サンプル（標本）を得る操作
→格子上の標本点(x, y)（＝(i, j)；i,jは離散値）の濃度値f_{ij}を定める手順（センサモデル2011-2参照）



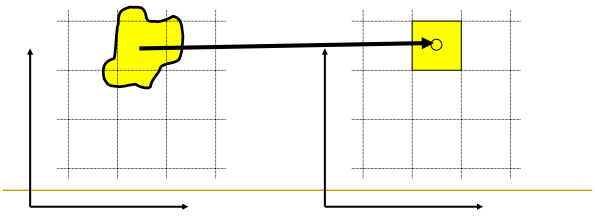
標本化

連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める



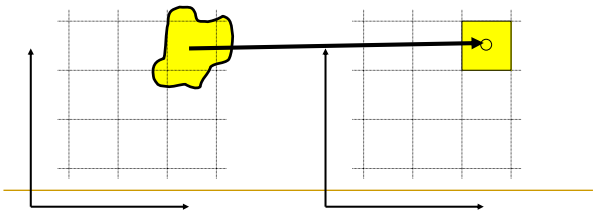
標本化

連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める



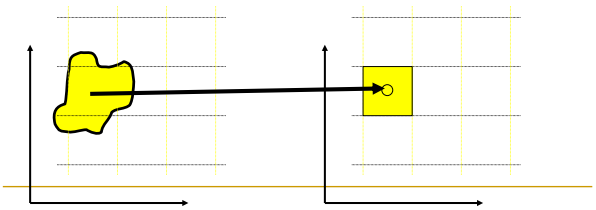
標本化

連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める



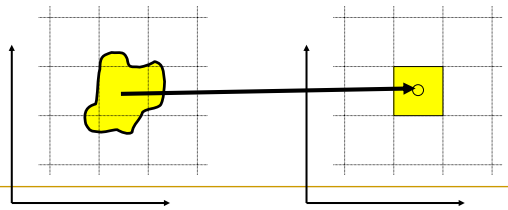
標本化

連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める



標本化

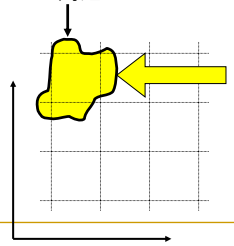
連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める



標本化

連続画像上の点(x, y)周辺
→ディジタル画像の画素(i, j)の濃度値f_{ij}を定める

画素(i, j)に対応する連続画像上の点の周辺

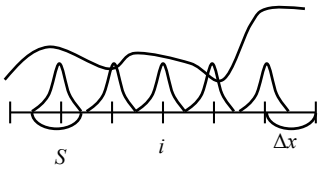


周辺はどう定義される？

画素と画素の間の距離は？

f_{ij} の決め方

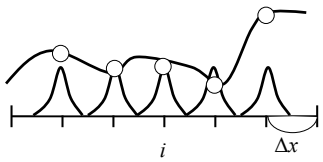
$\Delta x, \Delta y$: x,y軸方向の標本点間隔
S : 画面上の適当な領域,
$h(u,v)$: 適当な重み関数



f_{ij} の決め方

$$f_{ij} = \iint_S f\left(i\times\Delta x+u, j\times\Delta y+v\right) h(u,v)\,dudv$$

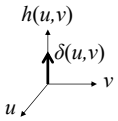
$\Delta x, \Delta y$: x,y軸方向の標本点間隔
S : 画面上の適当な領域,
$h(u,v)$: 適当な重み関数



$h(u,v)$の例と$f_{i,j}$

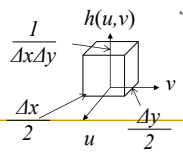
$f(x,y)$ 連続画像
$f'_{i,j}$ ディジタル画像

1) 2次元デルタ関数



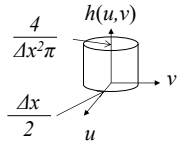
$$f'_{i,j} = f\left(i\times\Delta x, j\times\Delta y\right)$$

2) 2次元方形波



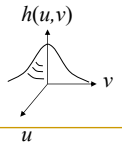
$$f_{i,j} = \frac{1}{\Delta x\Delta y} \times \iint_s f\left(i\times\Delta x+u, j\times\Delta y+v\right) dudv$$

3) 2次元円柱形



$$f'_{i,j}=\frac{4}{\Delta x^2\pi}\times\iint_s f\left(i\times\Delta x+u,\ j\times\Delta y+v\right)dudv$$

4) 2次元ガウス型関数



$$f'_{i,j}=\iint_s f\left(i\times\Delta x+u,\ j\times\Delta y+v\right)\times\frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{2\sigma^2}}dudv$$

標本点間隔の決め方

標本化定理　（染谷，Shannonが独立に証明）

$$f(x,y)=\sum_{i=-\infty}^{\infty}\sum_{j=-\infty}^{\infty}f\left(\frac{i\pi}{U},\frac{j\pi}{V}\right)\frac{\sin U\left(x-\frac{i\pi}{U}\right)}{U\left(x-\frac{i\pi}{U}\right)}\frac{\sin V\left(y-\frac{j\pi}{V}\right)}{V\left(y-\frac{j\pi}{V}\right)}$$

$$\left(x,y\in R,\quad i,j\in Z\right)$$

$$\boxed{F(u,v)=0}\quad\left(|u|\geq U,\ |v|\geq V\right)$$

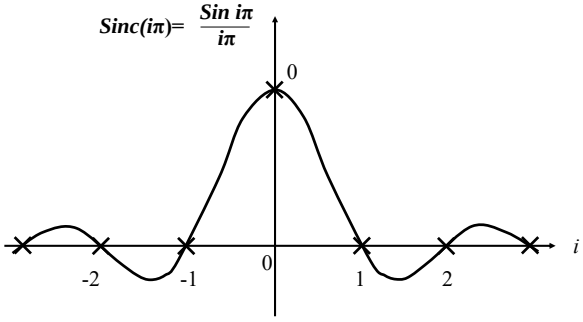
$f(x,y)$のフーリエ変換がU,V以上で0　すなわち　$f(x,y)$の最大空間周波数がU,Vのとき

画像をナイキスト間隔（π/U，π/V）で標本化すれば情報は失われない

標本化周波数(2U,2V)の1／2
＝ナイキスト周波数(U,V)
＝最大空間周波数

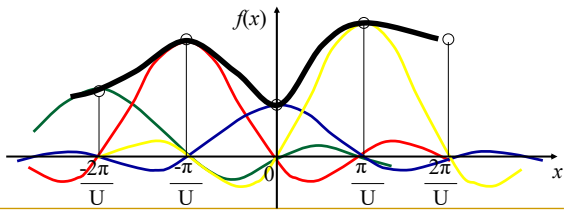
標本化関数

$$Sinc(i\pi)=\frac{Sin\ i\pi}{i\pi}$$



$$f(x)=\sum_{i=-\infty}^{\infty}f\left(\frac{i\pi}{U}\right)\frac{\sin U\left(x-\frac{i\pi}{U}\right)}{U\left(x-\frac{i\pi}{U}\right)}$$

$$=\cdots+f\left(-\frac{\pi}{U}\right)\frac{\sin U\left(x+\frac{\pi}{U}\right)}{U\left(x+\frac{\pi}{U}\right)}+f(0)\frac{\sin Ux}{Ux}+f\left(\frac{\pi}{U}\right)\frac{\sin U\left(x-\frac{\pi}{U}\right)}{U\left(x-\frac{\pi}{U}\right)}+\cdots$$



標本化定理による標本化の問題点

1) 最大空間周波数（U,V）が標本化の前に正確に分かる事はほとんど無い

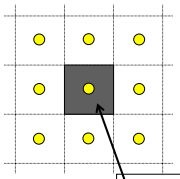
2) 画像パターン認識では，原画像の忠実な復元よりも認識に重要な情報を含むように ディジタル化する事の方が重要

3) ディジタル化では濃度値「$f(i\pi/U, j\pi/V)$」も量子化されて正確な値は分からない．また，標本化関数についても同様．

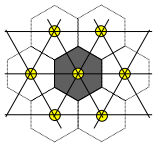
標本化の方法

平面を均等に充填分割する方法として、正方格子または三角格子がある。距離の概念など幾何学には、三角格子が優れているが、工学的に製造が楽、計算機の行列表現との親和性のため、正方格子の標本化がよくもちいられる。

正方格子標本化



三角格子標本化
（正三角形または正六角形）



画素(pixel：picture cell)

特徴

計算機による扱いが容易

隣接画素までの画素間距離が同じ

標本化：アナログ信号の離散化

再標本化：デジタル信号の標本化周波数の変更
サブサンプリング：密な信号のダウンサンプリング
スーパーサンプリング：素な信号のアップサンプリング

ミニ演習

1. 画像を横640画素x縦480画素（VGA)でサンプリングすると，画面全体の画素数はいくらか？
2. 1画素あたりRGB（赤，緑，青）各8ビット＝計24ビットで量子化すると，上記の画像のデータ量は何バイトになるか？1バイト＝8ビットとする
3. 上記画像をJPEG符号化すると46.08Kバイト（1Kバイト＝1000バイト）となった．情報圧縮率（＝圧縮後／圧縮前の比）を計算せよ．
4. 上記圧縮画像を携帯電話で友人に送る．何秒で送れるか？（伝送速度384Kbpsがフルに使えると仮定する）

宿題

- 手持ちのデジカメ・携帯カメラ・パソコンを使って，画像を撮影し，JPEG保存したときの画像の圧縮率を調べよ．最も高能率に圧縮できた例と最も低能率に圧縮できた例を説明せよ．（能率を変化させることができるときは，能率の違いと画像の品質の関係を見てみよ．）
- （例）写真を示すまたは写真の内容を説明して，圧縮比を記せ

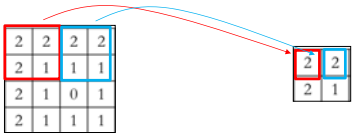
3648x2736画素→4．79MB　　4．08MB
29MB相当　　0．16　　0．14
単色の領域が大きい写真

標本化、サブサンプリング、ローパスフィルタ、エリアシングの関係

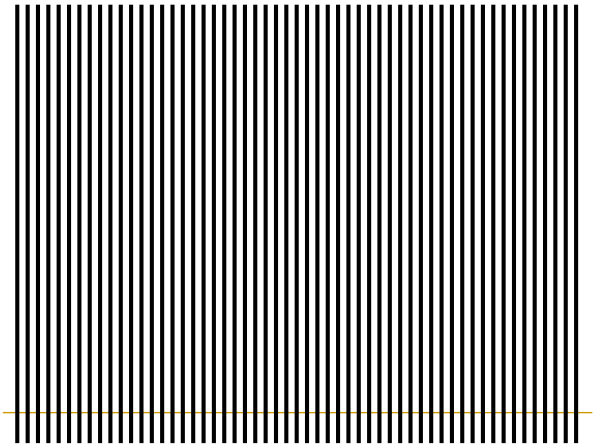
- NxM画素の画像、標本化周波数fの画像
- ↓ 1/nのサブサンプリング
- N/n x M/n画素の画像
  - ナイキスト周波数がf/nになる
  - 画像にf/n以上の周波数信号があればエリアシングが生じる
- 内在周波数をf/nに抑える必要があるので、ローパスフィルターをかける。σ＝n/π～3n/πのGaussian フィルターをかける。
- エリアシングは生じない（撮影時のエリアシングを消すことはできない）

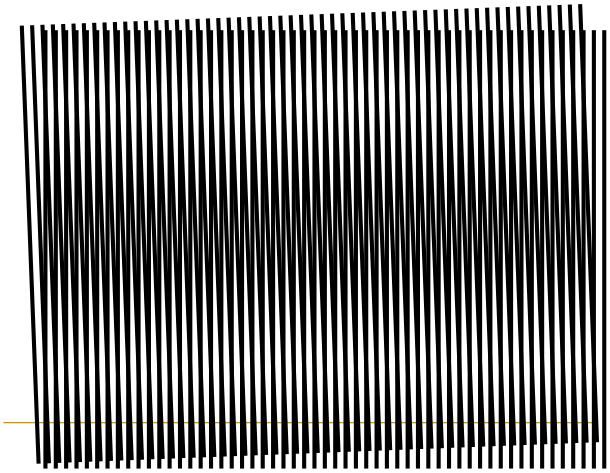
サブサンプリング（sub-sampling）

- 標本点の間引き

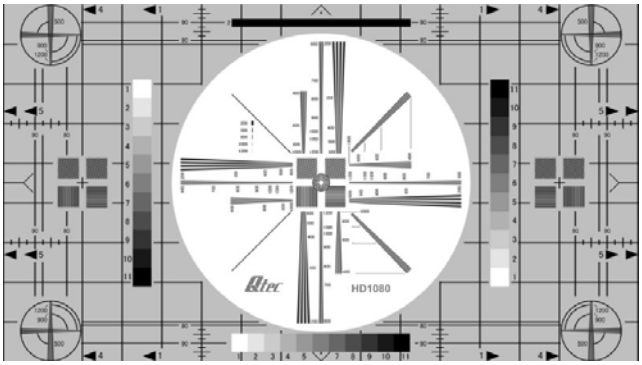


- 図の例：2x2の局所処理（最大値選択）をして解像度の低い画像を作成
- 局所処理としては，max，mix，平均など





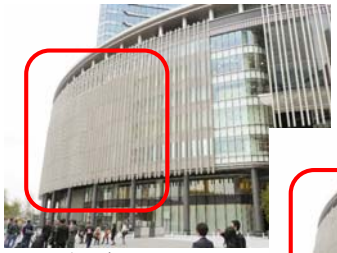
モノスコ(monoscope)テストパターン



原画像
(1200x900pixel)

Sub-samplingによるaliasingの影響

Sub-sampled画像
1/2 (600x450pixel)



フィルターなし

Gaussian フィルター適用後サンプリング

Sub-samplingによるaliasingの影響

Sub-sampled画像
1/2 (600x450pixel)

フィルターなし

Gaussian フィルター適用後サンプリング

画像は波の重ね合わせからなる

- 周波数の概念
- 2次元の波
- 例:フーリエ変換・逆変換
  - わかりやすい例
  - http://opencv.jp/opencv2-x-samples/2d_dft_idft

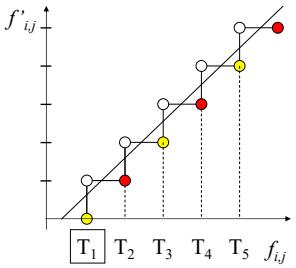
テレビ画像

- NTSCとPAL

$f_{i,j} \rightarrow f'_{i,j}$ の量子化の例

（アナログ→デジタル変換、A/D変換時の量子化）

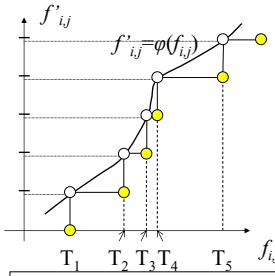
直線量子化



しきい値

階調数 小（黄のみ）
大（赤と黄）

非直線量子化



まず$f'_{i,j}=\varphi(f_{i,j})$によって変換し，その後$f'_{i,j}$を直線量子化したと考える事も出来る．

256階調

8階調

$\varphi(f'_{i,j})$の例

$$f_{i,j}= \begin{cases} k\, f'_{i,j} & (k:\text{定数}) \\ k\, \log f'_{i,j} & \text{人の視覚との整合性が良い（対数変換）} \end{cases}$$

∵ 大まかには…

[人の目が感じる明るさ] = $\alpha \log I_{i,j}^{(out)} + \beta$ （α, β :定数）

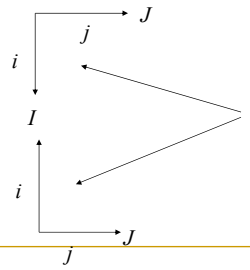
画像フォーマット（C言語の場合）

I行(row)J列(column)の画像

a[I][J]

J画素

I画素



a[i][j]=*((*(a+i))+j)
（jから回る）
(a[i*J+j])

```
for(i=0,i<I,i++){
 for(j=0,j<J,j++){
  …}}
```

OpenCV

Intelがスタートしたcvのオープンソースフォーラム
（以下，数頁はちょっと古い話）

- 入手： http://opencv.jp/
- 最新バージョン：4.1.2（3. 4. 3）
- 処理系： Windows、Linux
- 言語： c++, phython
- テキスト：v1.1の日本語など
- サンプルコード：
  http://opencv.jp/sample/index.html

OpenCVにおける配列表現（構造体）

- CvArr 仮想的な配列構造
- CvMat： 多重行列
- IplImage： IPL画像ヘッダ
- ndarray：pythonにおける画像表現

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

ファイルから *IplImage* として画像を読み込みます．

•関数 cvLoadImage は，指定したファイルから画像を読み込み，その画像へのポインタを返します．現在のところ，以下のファイルフォーマットがサポートされています．

•Windows bitmaps - BMP, DIB
•JPEG files - JPEG, JPG, JPE
•Portable Network Graphics - PNG
•Portable image format - PBM, PGM, PPM
• rasters - SR, RAS
•TIFF files - TIFF, TIF

現在の実装では，アルファチャンネルがもしあったとしても，出力画像からは取り除かれることに注意してください．例えば，4チャンネルRGBA画像は，RGB画像として読み込まれます．

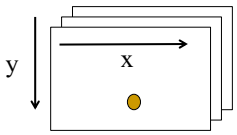
パラメタ:
- *filename* – 読み込むファイル名
- *iscolor* – 読み込む画像のカラーの種類を指定します：
  - **CV_LOAD_IMAGE_COLOR** 画像は，強制的に3チャンネルカラー画像として読み込まれます
  - **CV_LOAD_IMAGE_GRAYSCALE** 画像は，強制的にグレースケール画像として読み込まれます
  - **CV_LOAD_IMAGE_UNCHANGED** 画像は，そのままの画像として読み込まれます

- OpenCVでは，IPL（Intel Image Processing Library）で使われていた構造体 IplImage フォーマットの一部をサポートしている．OpenCVの多くの関数が，構造体 IplImage を含む CvArr を，その引数に取る．構造体 IplImageのメンバである imageData に実際の画素値が格納されており，画像の幅と高さはそれぞれ，width，heightで示される．また，メンバ変数 widthStep は，画像の水平方向1ライン分をバイト単位で表す値であり，これは，imageData のデータに直接アクセスする際に利用される．widthStep はメモリのアライメント（alignment）により，width と同じか，それよりも大きい値となる．
これらの知識は，画素値を直接操作する際に必要になるが，簡単なプログラムを書く場合は，このようなメモリ上の画素値の配置を考慮することはないかもしれない．しかし，OpenCVには実装されていない画像処理手法を用いたい場合や，任意の描画を行いたい（画像上に二次曲線を描きたいなど）場合，IplImageの画像データを別の構造体（他のライブラリの構造体など）で扱いたい場合などは，どうしても画素値を直接操作する（各画素へのポインタを取得する）方法が必要になる．しかし，ここでは単純なサンプルを使っていくつかの方法を紹介する．

画素値アクセスの方法（3種類）

- （1）画素値へのポインタを最初に計算する
- （2）CV_IMAGE_ELEM マクロを利用する
- （3）OpenCVの配列アクセス関数を利用する
  - 遅い

マクロ




```
#define IMAGE(i, x, y, n)
 *((unsigned char *)
   ( (i)->imageData  + (i) ->widthStep * (y)
   + sizeof(unsigned char) * (x)  *3 ) + (n))
```

```
*((unsigned char *)
  ( (i)->imageData  +                  ⇐画像構造体の画像部ポインタ+
    (i) ->widthStep * (y) +            ⇐ 1行データ幅＊y行目　＋
sizeof(unsigned char) * (x)  *3 ) +    ⇐ 1画素サイズ＊x画素＊3色+
    (n))                               ⇐ 色要素
```

http://opencv.jp/sample/basic_structures.html

8ビット3チャンネルカラー画像を読み込み，画素値を変更する

```
#include <cv.h>
#include <highgui.h>

int
main (int argc, char **argv)
{
  int x, y;
  uchar p[3];
  IplImage *img;

  if (argc != 2 || (img = cvLoadImage (argv[1],
CV_LOAD_IMAGE_COLOR)) == 0)
    return -1;

  // (1)画素値(R,G,B)を順次取得し，変更する
  for (y = 0; y < img->height; y++) {
```

http://opencv.jp/sample/basic_structures.html（続き）

```
 // (1)画素値(R,G,B)を順次取得し，変更する
 for (y = 0; y < img->height; y++) {
   for (x = 0; x < img->width; x++) {
     /* 画素値を直接操作する一例 */
     p[0] = img->imageData[img->widthStep * y + x * 3];       // B
     p[1] = img->imageData[img->widthStep * y + x * 3 + 1];   // G
     p[2] = img->imageData[img->widthStep * y + x * 3 + 2];   // R
     img->imageData[img->widthStep * y + x * 3] = cvRound (p[0] *
0.6 + 10);
     img->imageData[img->widthStep * y + x * 3 + 1] = cvRound (p[1]
* 1.0);
     img->imageData[img->widthStep * y + x * 3 + 2] = cvRound (p[2]
* 0.0);
   }
 }
```

画像ファイルの話

- ファイルフォーマット
  - ヘッダー部＋データ部（＋EOF部）
- データ部の違い
  - 符号化圧縮なしビットマップ
    - PNM(Portable aNyMap, PBM, PGM, PPM)
  - 符号化されたビットマップ（符号化テーブル＋信号）
    - JPEG，MPEG
  - カラーマップ＋ビットマップ
    - GIF, PNG
- 統合された形式
  - TIFF，BMP，PICT（ベクター＋ビットマップ）

簡単な例－PNM

- ヘッダー
  - マジックナンバー（P1-P6) 2 Bytes
    - P1-PBM+ascii, P3-PBM-binary, P4-PPM-ascii,,,
    - 例：P1
  - コメント
    - 例：# this is a comment
  - 画像サイズ　横幅(column)　高さ(row)
    - 例：6 10
  - ピクセル値の最大値
    - 例：15 （15階調）
- データ

ピクセル配列（PPMは，各画素のRGB値，次の画素，，，の順）

PBM（バイナリビットマップファイルの例）

```
P1
# feep.pbm
24 7
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 1 0
0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0
0 1 0 0 0 0 0 1 1 1 1 0 0 1 1 1 1 0 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Gray Image ファイルの例

```
P2
# feep.pgm
24 7
15
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 3 3 3 3 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 15 0
0 3 3 3 0 0 0 7 7 7 7 0 0 0 11 11 11 0 0 0 15 15 15 15 0
0 3 0 0 0 0 0 7 0 0 0 0 0 11 0 0 0 0 0 15 0 0 0 0
0 3 0 0 0 0 0 7 7 7 7 0 0 11 11 11 11 0 0 15 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Color Imageファイルの例

P3
feep.ppm
4 2　　画像サイズ
15　　画素の最大値
15 15 15　15 0 0　　0 15 0　0 0 15
15 15　0　0 15 15　15 0 15　0 0 0

JPEG変換

1. 原画像
2. 8x8画素ごとのブロック化
   1. DCT（離散コサイン変換）
   2. 符号長の決定（ベクトル量子化，ハフマン符号化）
3. ヘッダー＋
   （量子化・ハフマンテーブル＋8x8のDCTパターン）xn

今日のまとめ

- 画像のディジタル化
- 標本化定理
- コンピュータ内部の表現
- 画像処理ツール（ライブラリ）OpenCV