

知能制御システム学

画像処理の基礎 (1)  
— 基礎概念と OpenCV の導入 —

東北大学 大学院情報科学研究科

鏡 慎吾

swk(at)ic.is.tohoku.ac.jp

2010.06.29

# 予定

前半で学んだビジュアルサーボシステムを実現するための**画像処理技術の基礎の基礎**について可能な限り**実践的に学ぶ**

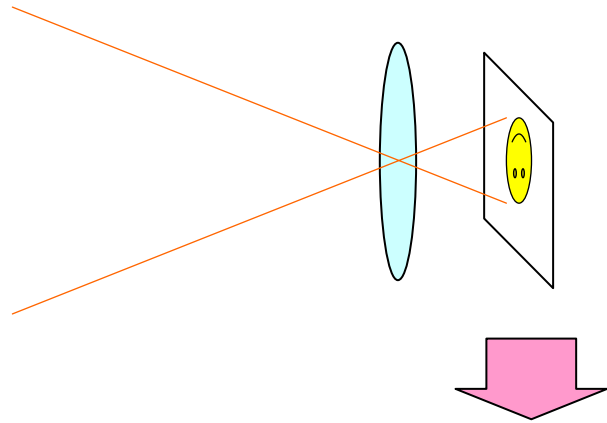
- 2010.06.29 画像処理 (1) 準備編
- 2009.07.06 画像処理 (2) 基礎編
- 2009.07.13 画像追跡 (1)
- 2009.07.20 画像追跡 (2)  
(7/27 は休講)
- 2009.08.03 画像追跡 (3)

講義資料(サンプルコード): <http://www.ic.is.tohoku.ac.jp/~swk/lecture/>

昨年度の内容は,

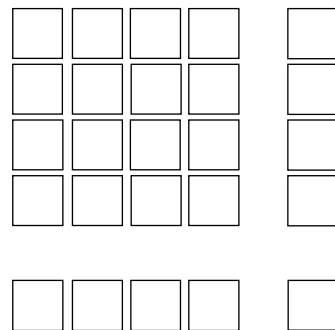
- カメラとイメージセンサの基礎 (今年は無し)
- OpenCVの概要と, 画像処理の基本的な考え方・実例
- 画像追跡の例として, Harrisオペレータ, KLTトラッカ, Mean Shiftトラッカ
- Cプログラミングを前提として, 高速化のための基本事項をいくつか

# デジタル画像



撮像面における入射光  
強度のアナログ分布

2次元離散化 (「画素」への分割)  
量子化 (A/D変換)

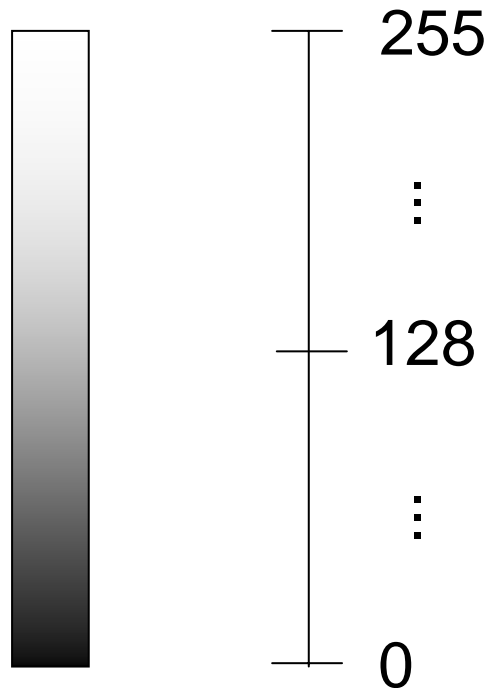


デジタル画像:

画素の明るさを表す値 (画素  
値) が2次元に並んだもの

# 画素値

「明るさ」なり「電圧」なりというアナログの量を，デジタルな数字に量子化する．このようにして得られた値を濃度値，濃淡値，グレイレベル，あるいは単に画素値などと呼ぶ



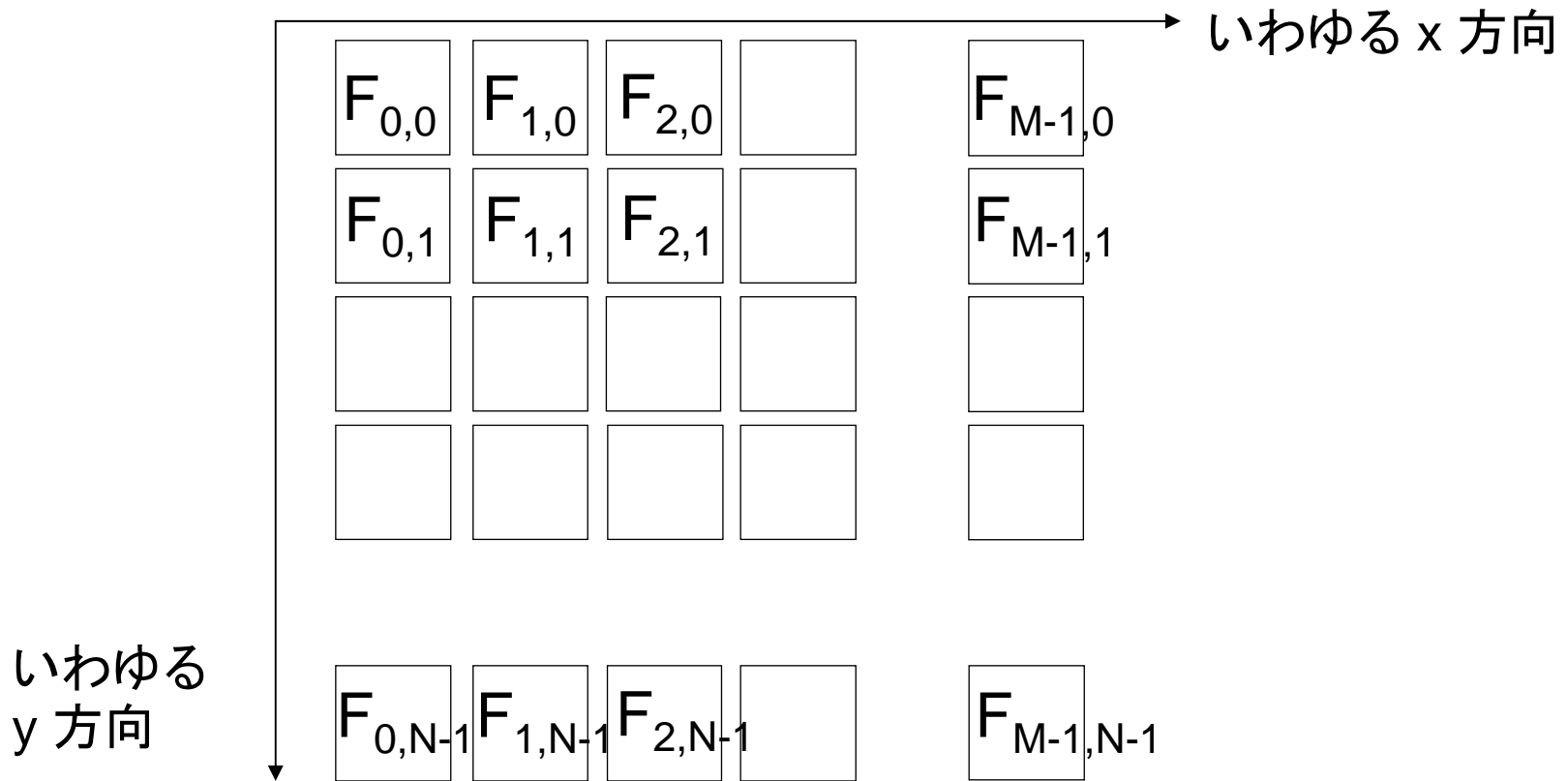
0~255 の 256 階調に一樣量子化する例  
→ 8ビット多値画像  
(8-bit grayscale image)

特に1ビット画像のことを2値画像  
(binary image) と呼ぶ

# デジタル画像の表現

$M \times N$  画素のデジタル画像: 画像上の位置  $(x,y)$  における画素値を  $F_{x,y}$  で表す

$\{ F_{x,y} \}, x = 0, 1, \dots, M-1, y = 0, 1, \dots, N-1$



# Cでの表現

- 8ビットグレイスケール画像であれば, unsigned char 型の配列で表せばよい
- 通常はx方向を優先して走査したいので, x方向に画素が連続して配置されるような構造が望ましい

```
#define M 640  
#define N 480
```

```
unsigned char image1[N][M];  
// (unsigned char の N 要素の配列) の N 要素の配列
```

```
unsigned char *image2[N];  
// (unsigned char の配列へのポインタ) の N 要素の配列
```

```
unsigned char image3[M * N];  
// unsigned char の (M * N) 要素の配列
```

- 2次元配列(またはポインタの配列)をそのまま使うと、添え字の順番が慣例と逆になる
- 「配列へのポインタ」の配列にする方が、ランダムアクセスの際のアドレス計算の回数は減らせる(その分、動的確保に手間はかかり、余分な記憶領域が必要)
- むしろ最初から1次元配列にしてしまうことも多い

```
#define M 640
#define N 480
unsigned char image3[M * N];
```

```
image3[M * y + x] = 30;
// F(x, y) := 30
```

image1, image3

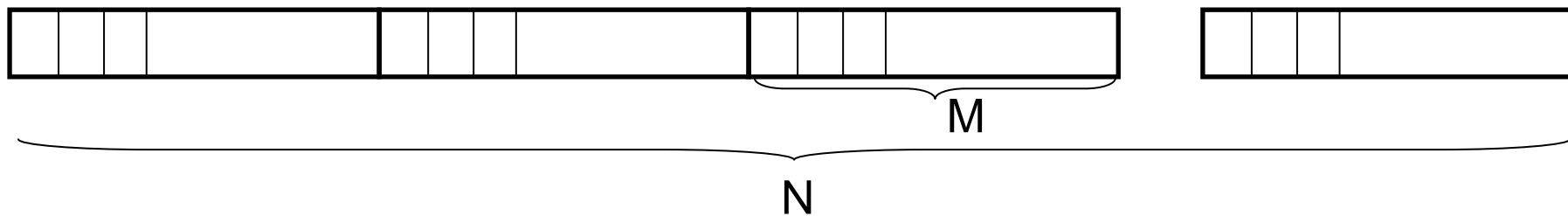
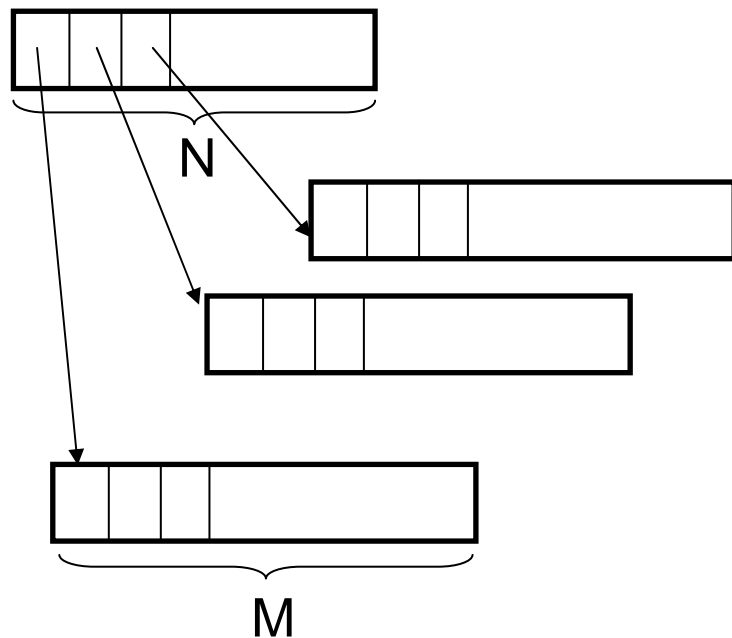


image2





# 典型的な画像処理

## 2値化 (binarization, thresholding)

```
#define M 640
#define N 480
unsigned char image[M * N];
int i, j;

for (j = 0; j < N; j++) {
    for (i = 0; i < M; i++) {
        if (image[M * j + i] >= 128) {
            image[M * j + i] = 255;
        } else {
            image[M * j + i] = 0;
        }
    }
}
```

# 画像処理ライブラリ

このように自前で画像処理を書くことはもちろん可能だが、標準的なライブラリを利用すると(あるいは少なくともデータ構造だけでも標準的なライブラリに合わせておくと), いろいろと便利

ここでは OpenCV を紹介する

- 以下で示すプログラム例は, 説明用に簡略化されたものであり, そのまま実用に耐えるものではない点に注意すること.
- OpenCV の使い方の講義ではない. 画像処理の概念を学ぶことが目的なので, OpenCVに標準関数として用意されているものを敢えて再実装している場合がある(していることが多い).

# この講義での実行環境

他の環境を使いたい人は、適当に脳内変換して下さい  
(私自身も決してこれらの環境での作業が得意なわけではないです。というかどちらかというと苦手です)

- OS: Microsoft Windows
- 開発環境.: Microsoft Visual Studio Professional 2005
- 言語: C (C++)
- ライブラリ: OpenCV (この講義ではver1.0を使用. 古い)

<http://sourceforge.net/projects/opencvlibrary/>

<http://opencv.willowgarage.com/wiki/>

# Visual Studio

- 講義では Microsoft Visual Studio Professional 2005 を使用. インストール法などの説明は省略
- 注意: フリーのバージョンの Visual Studio (Visual Studio Express など) には, コードの最適化に制限があるようである (従って, 性能の評価がしにくいかも知れない)

# OpenCV のインストール

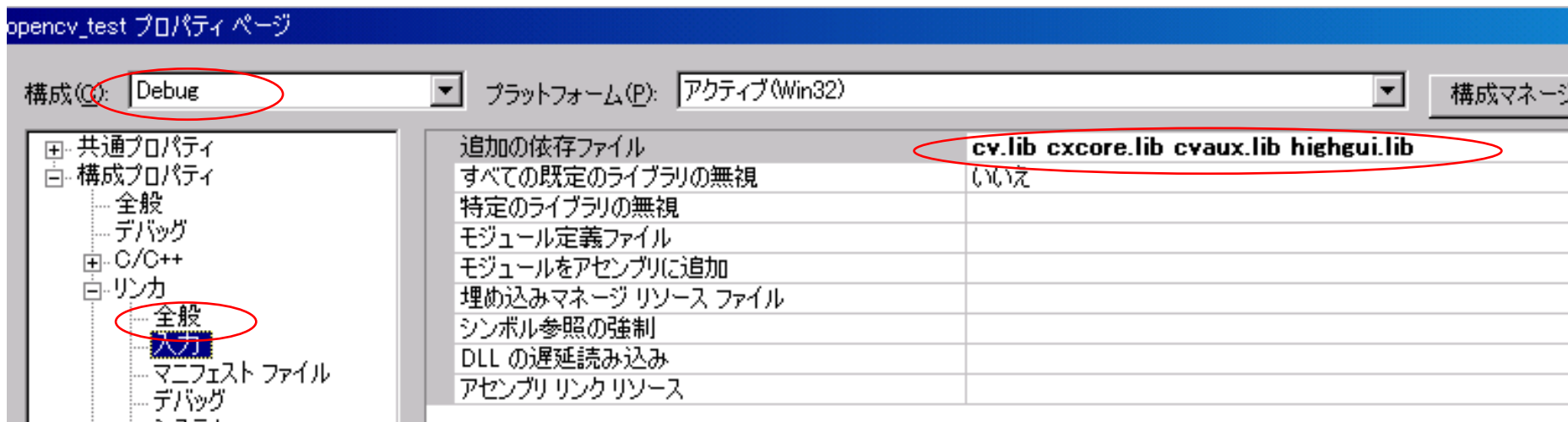
- opencv-win をダウンロードしてインストール
- Visual Studio の ツール-オプション にて,  
プロジェクトおよびソリューション – VC++ ディレクトリを選び,  
「インクルードファイル」に以下を追加
  - C:¥Program Files¥OpenCV¥cv¥include
  - C:¥Program Files¥OpenCV¥cxcvcore¥include
  - C:¥Program Files¥OpenCV¥otherlibs¥highgui
  - C:¥Program Files¥OpenCV¥cvaux¥include
  - C:¥Program Files¥OpenCV¥otherlibs¥cvcam¥include
- 「ライブラリファイル」に以下を追加
  - C:¥Program Files¥OpenCV¥lib
- 「ソースファイル」にも対応するディレクトリを追加しておくとも便利かもしれない
- 環境変数 PATH に以下を追加
  - ;C:¥Program Files¥OpenCV¥bin
- See e.g.  
<http://opencv.willowgarage.com/wiki/VisualC++>

# 新規プロジェクトの作成

- プロジェクトのプロパティで構成プロパティ-リンカ-入力の「追加の依存ファイル」に以下を追加

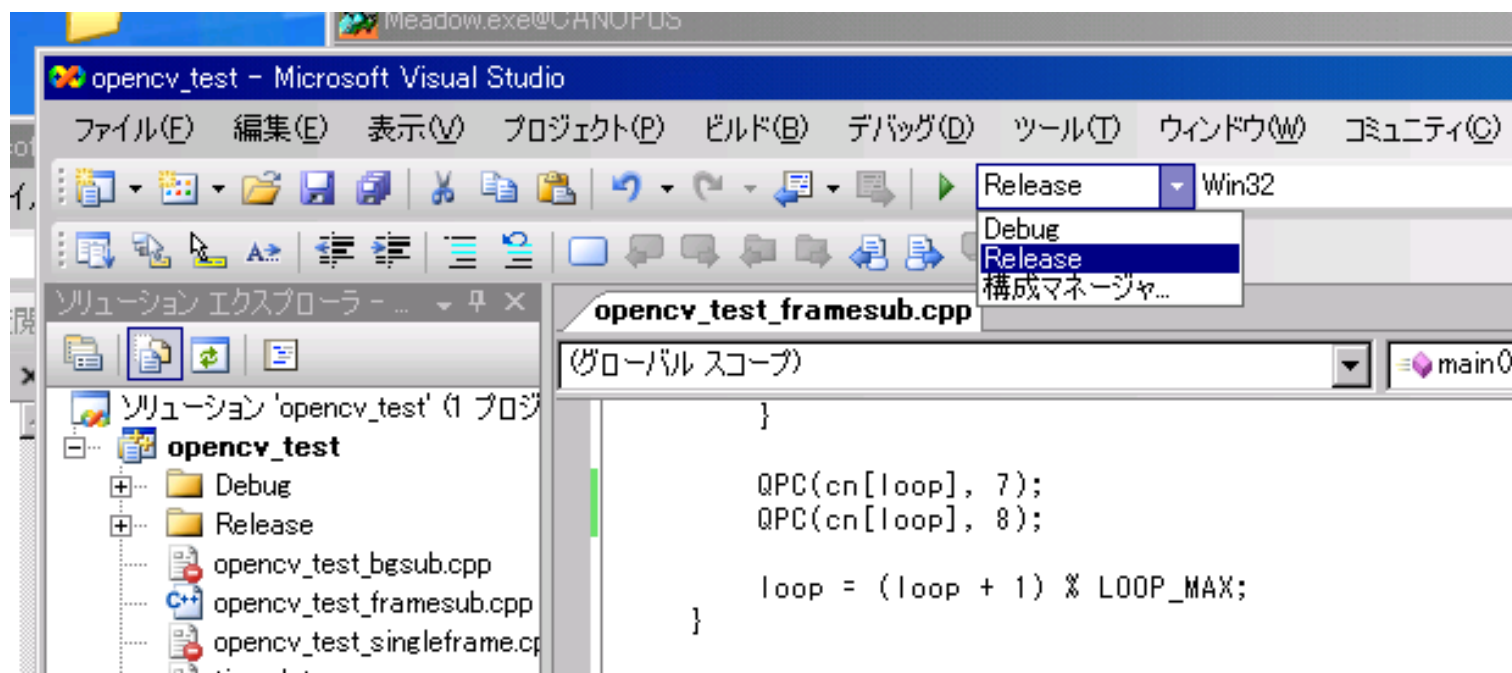
```
cv.lib cxcore.lib highgui.lib
```

- Release構成でも設定しておくのを忘れないこと



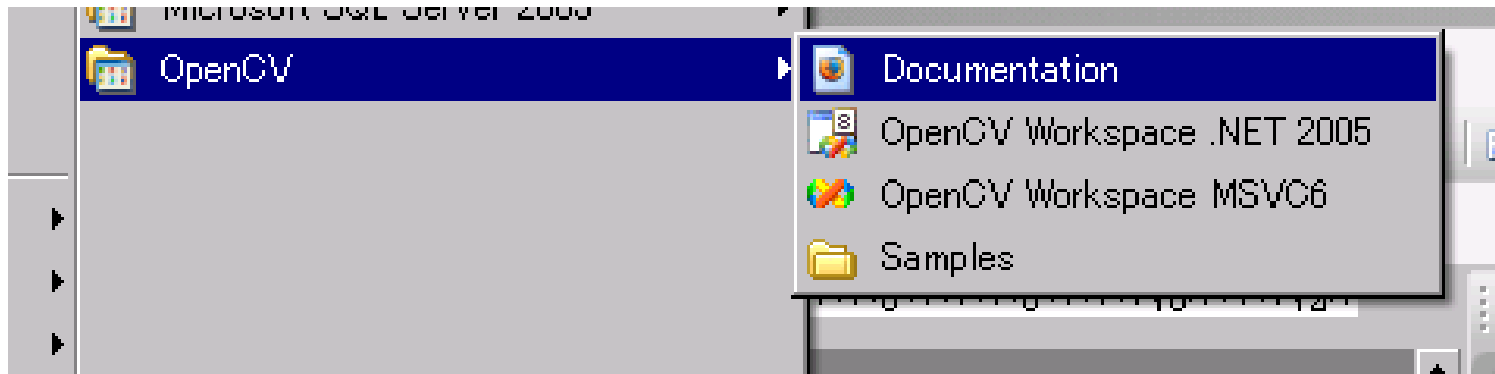
# Debug 構成と Release 構成

Debug 構成だと、あまり最適化が効いていない(かもしれない)ので注意. 性能を測定するときは Release 構成でビルド・実行すること (設定を忘れないように)



# OpenCV ドキュメント

スタートメニューから HTML のリファレンスマニュアルを開ける



最近はウェブ上で、あるいは書籍で日本語の情報も多い



# 典型的な流れ(単一の画像)

```
#include "cv.h"
#include "cxcore.h"
#include "highgui.h"

int
main()
{
    // Initialize

    // Load Image

    // Process the Image

    // (Display or output the result)

    // (Finalize)

    return 0;
}
```

sample programs:

- single\_image.cpp
- single\_image\_load.cpp

# IplImage Structure

```
typedef struct _IplImage {  
    ...  
    int nChannel;  
    int depth;  
    int origin;  
    int width;  
    int height;  
    char *imageData;  
    int widthStep;  
    ...  
}
```

# IplImage Structure

nChannel

1: grayscale images

3: color images

depth

IPL\_DEPTH\_8U: 8-bit natural number (i.e. uchar)

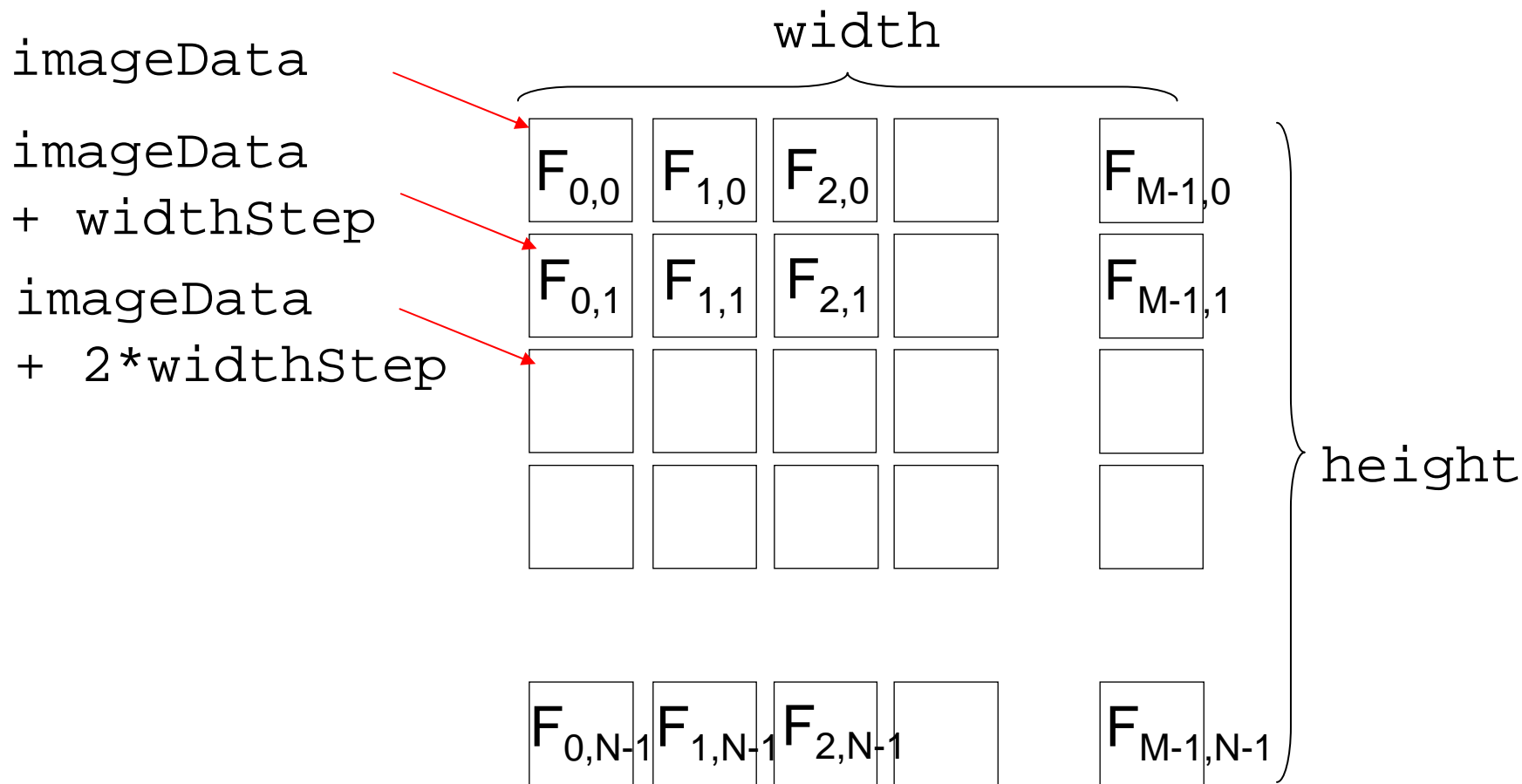
IPL\_DEPTH\_64F: 64-bit real number (i.e. double)

origin

0: origin at top-left

1: origin at bottom-left (e.g. Windows BMP)

# imageData (モノクロ画像用)



画像の水平方向が連続に並んだ char 型の配列

# 画素データへのアクセス

depth = IPL\_DEPTH\_8U, nChannel = 1 の場合

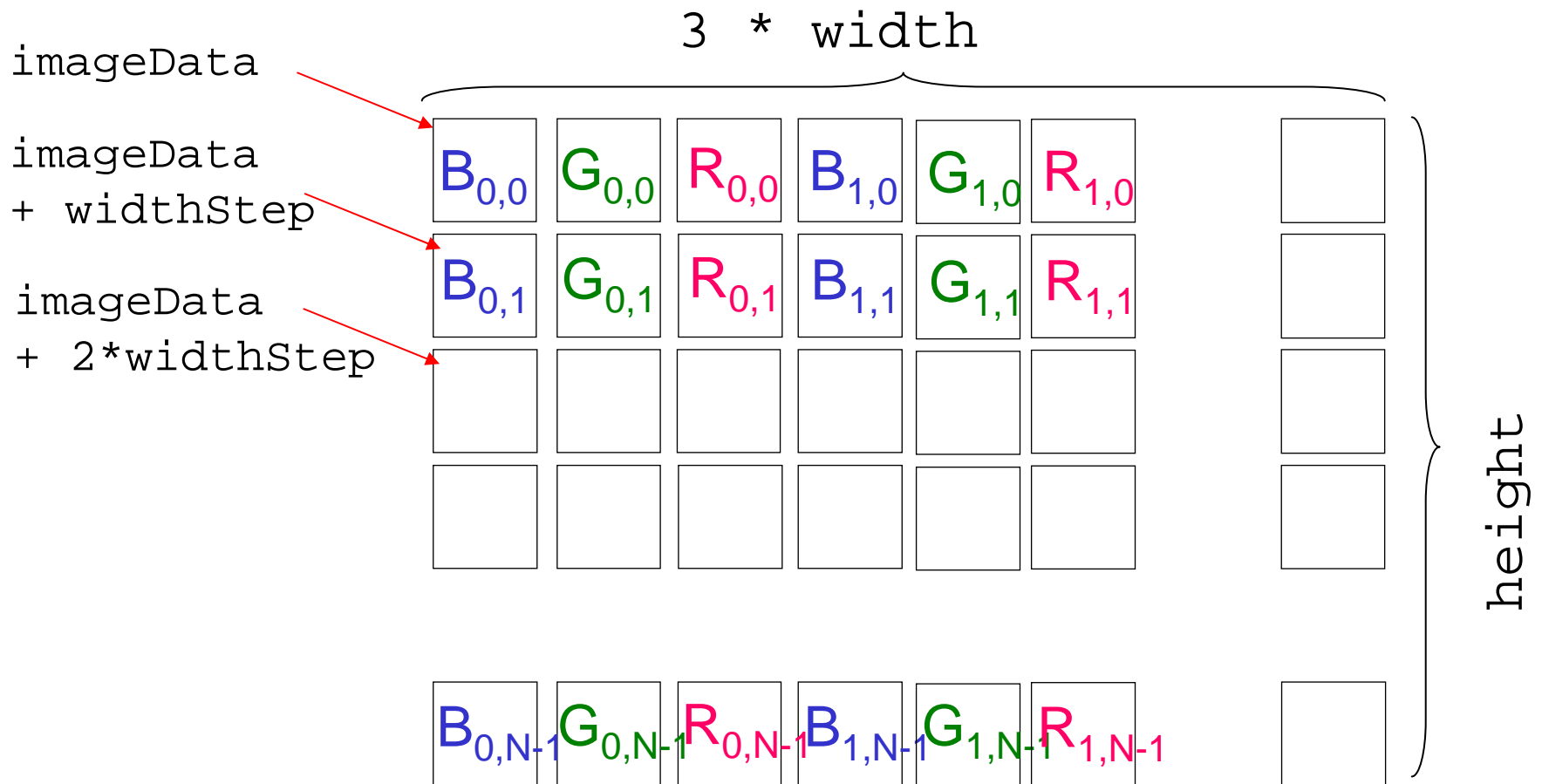
$$F_{x,y} \sim (*(\text{uchar } *) (\text{img} \rightarrow \text{imageData} \\ + y * \text{img} \rightarrow \text{widthStep} \\ + x))$$

```
#define PIXVAL(iplimagep, x, y) \
    (*(uchar *) ((iplimagep) \
    ->imageData \
    + (y) * (iplimagep) \
    ->widthStep + (x)))
```

```
PIXVAL(img, 10, 20) = 30;
x = PIXVAL(img, 128, 150);
```

ただし、このマクロで画素にアクセスすると、シーケンシャルアクセスの場合でも毎回アドレス計算が発生するので、速度面では必ずしも好ましくない

# imageData (RGBカラー画像用)



# 画素データへのアクセス (多チャンネル)

depth = IPL\_DEPTH\_8U, nChannel = 3 の場合

$$F_{x,y,c} \sim ( * (uchar *) (img \rightarrow imageData + y * img \rightarrow widthStep + x * img \rightarrow nChannel + c) )$$

```
#define PIXVALC(iplimagep, x, y, c) \
    ( * (uchar *) ((iplimagep) \
    \rightarrow imageData \
    + (y) * (iplimagep) \
    \rightarrow widthStep + \
    + (x) * (iplimagep) \
    \rightarrow nChannel) \
    + (c) )
```

```
PIXVALC(img, 10, 20, 0) = 30; \
x = PIXVALC(img, 128, 150, 2);
```

# 典型的な流れ(複数フレーム)

```
int
main()
{
    // Initialize

    while (1) {

        // capture image from camera

        // Process the Image

        // (Display or output the result)

    }

    // (Finalize)

    return 0;
}
```



# USBカメラから画像を得る

```
CvCapture *capture;  
IplImage *frame;  
  
capture = cvCaptureFromCAM(0); // 0: the first camera  
/* returns NULL if no camera is found */  
  
frame = cvQueryFrame(capture);  
/* Note that this image must not be modified, so copy it  
   before you modify */
```

sample program :  
•framediff.cpp

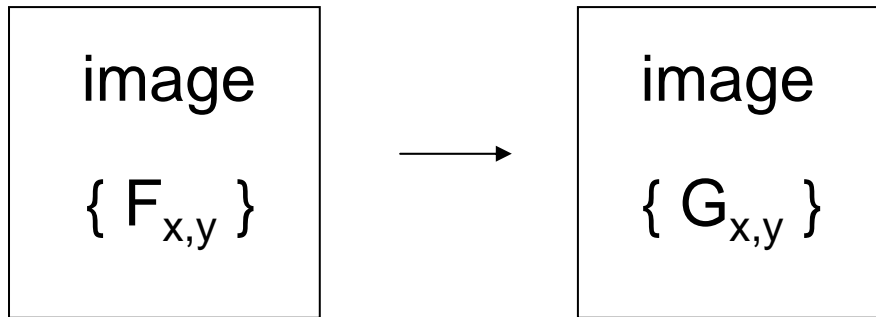
# USBカメラを持っていない人は

1. デジタルカメラ等で撮った写真を `cvLoadImage` で読み込めばよい (連続写真を撮っておいて連番のファイル名に格納して, 順次読み出せば動画処理ができる)
2. `cvCaptureFromCAM` の代わりに `cvCreateFileCapture` を使ってビデオファイルを読み込める

# 画像処理の分類

input	output	example
image	image (2-D data)  1-D data  scalar values	image to image processing Fourier trans., label image  projection, histogram  position, recognition
image sequence	image (2-D data)  ...	motion image processing

# 画像から画像への変換



point operation (点処理)

$G_{i,j}$  が  $F_{i,j}$  にのみ依存

local operation / neighboring operation (局所処理)

$G_{i,j}$  が  $\{ F_{i,j} \}$  のうち  $(i, j)$  の近傍の画素のみに依存

global operation (大域処理)

$G_{i,j}$  が  $\{ F_{i,j} \}$  の(ほぼ)すべての画素に依存

# 点処理の例

濃度変換, 色変換など

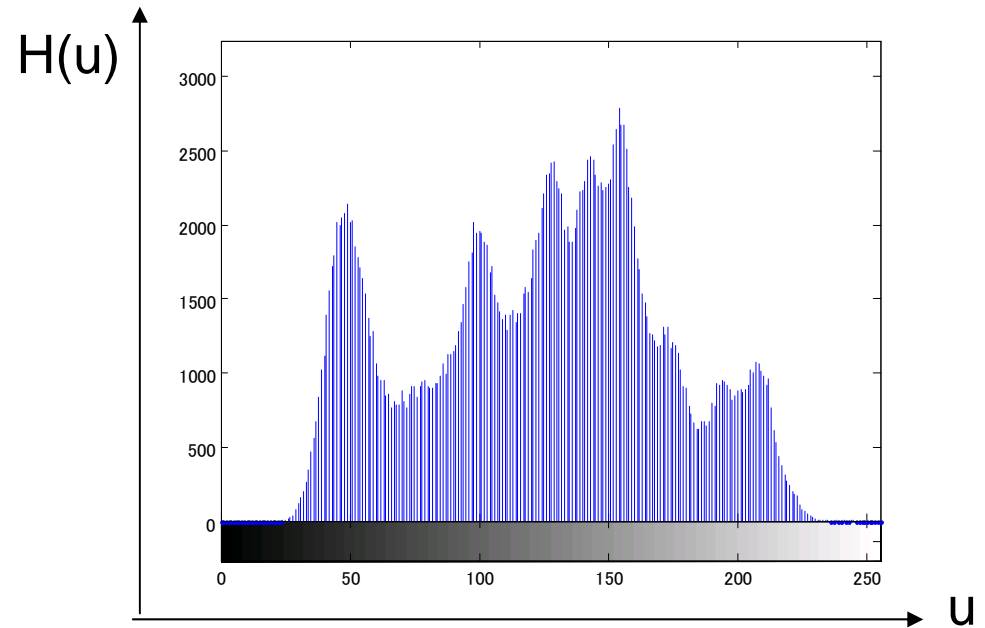
- 2値化, 色反転, ガンマ変換などが代表的な例

```
for (j = 0; j < img->height; j++) {  
    for (i = 0; i < img->width; i++) {  
        // PIXVAL(img, i, j) の操作  
    }  
}
```

sample program:  
•binarize.cpp

- ※ OpenCVで実際にプログラミングする際は
  - 2値化を行うには cvThreshold() を使うとよい

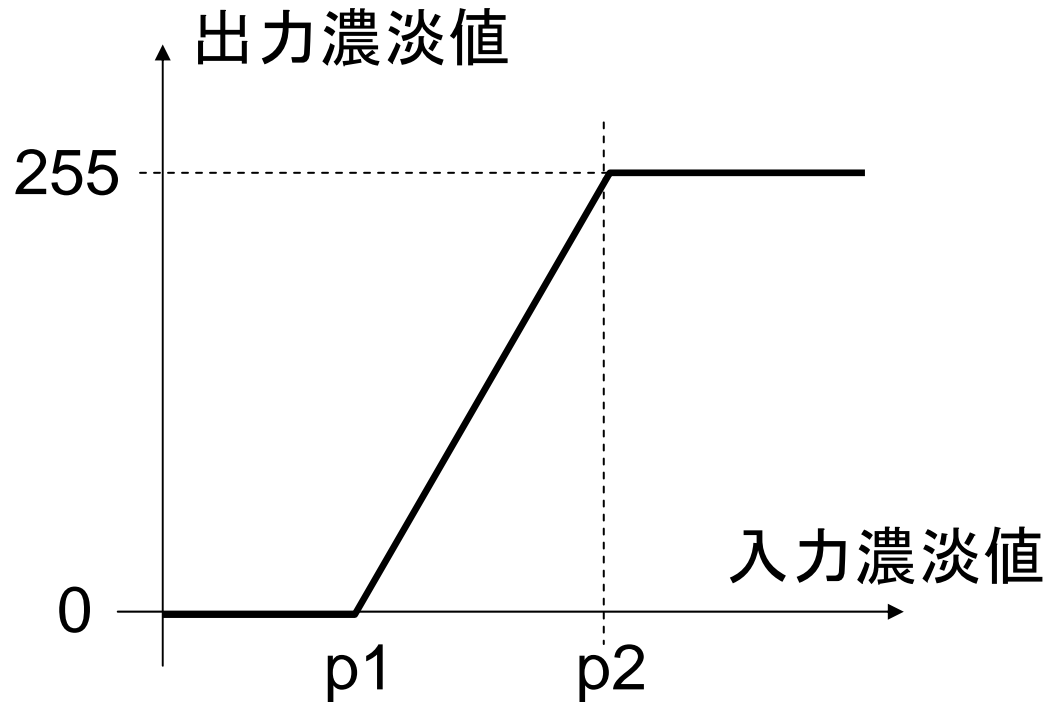
# (濃淡値)ヒストグラム



$$H = \{H_u\}_{u=1,2,\dots,m}, \quad H_u = \sum_{x \in S(u)} 1$$

ただし画素値がビン  $u$  に含まれる画素の集合を  $S(u)$  と書いた

# 濃度変換の簡単な例



sample program:  
• histogram.cpp

※ OpenCVにはヒストグラムを扱うデータ構造 CvHistogram が用意されているので利用を検討するとよい

# References

- [1] 田村: コンピュータ画像処理, オーム社, 2002.
- [2] デジタル画像処理編集委員会: デジタル画像処理, CG-ARTS協会, 2004.
- [3] <http://sourceforge.net/projects/opencvlibrary/>
- [4] <http://opencv.willowgarage.com/wiki/>
- [5] G. Bradski and A. Kaebler: Learning OpenCV, O'Reilly, 2008. (松田 訳: 詳解OpenCV, 2009)
- [6] 奈良先端科学技術大学院大学 OpenCVプログラミングブック制作チーム: OpenCVプログラミングブック 第2版, 毎日コミュニケーションズ, 2009.
- [7] 永田: 実践OpenCV, カットシステム, 2009.