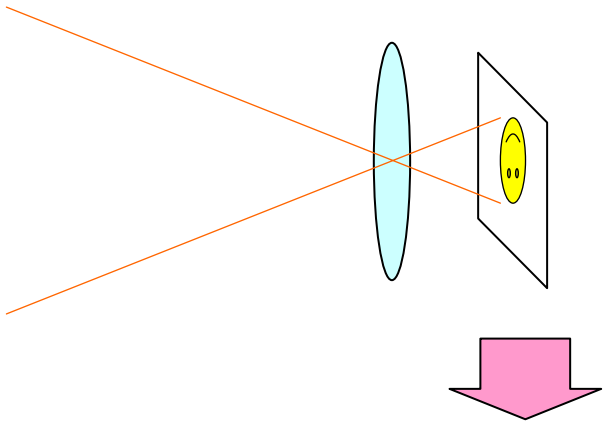
知能制御システム学

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

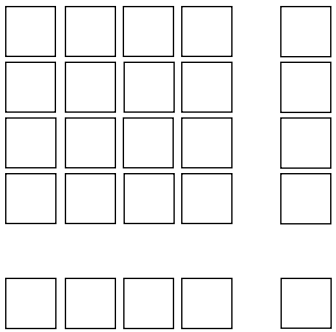
2009.06.23

ディジタル画像



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）



ディジタル画像:

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

「電荷量」なり「電圧」なりというアナログの量を，ディ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

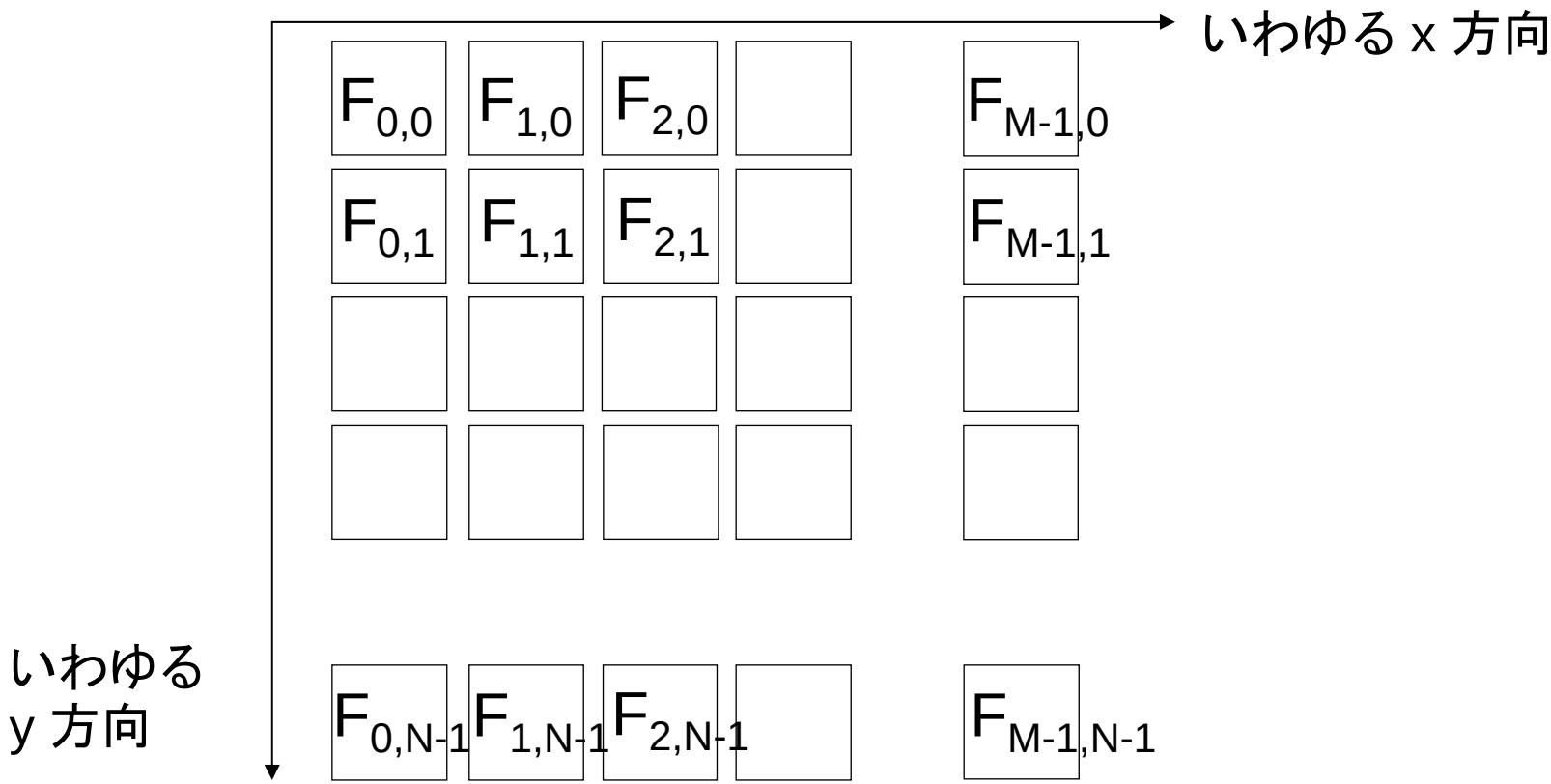
0～255 の 256 階調に一様量子化する例
→ 8ビット多値画像
（8-bit grayscale image）

特に1ビット画像のことを2値画像 (binary image) と呼ぶ

ディジタル画像の表現

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

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



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

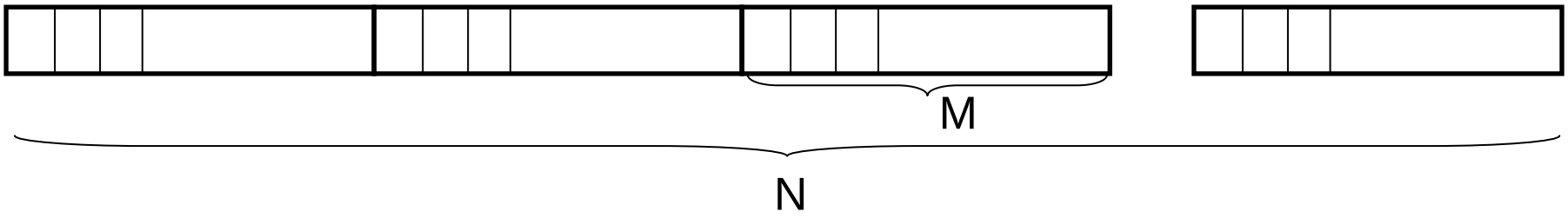
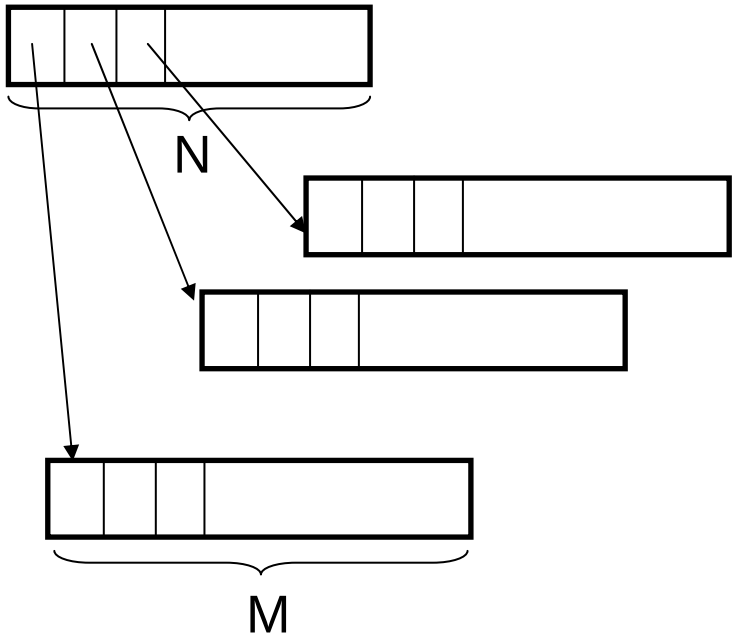


image2



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もある

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

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

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 2005 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¥cxcore¥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
```

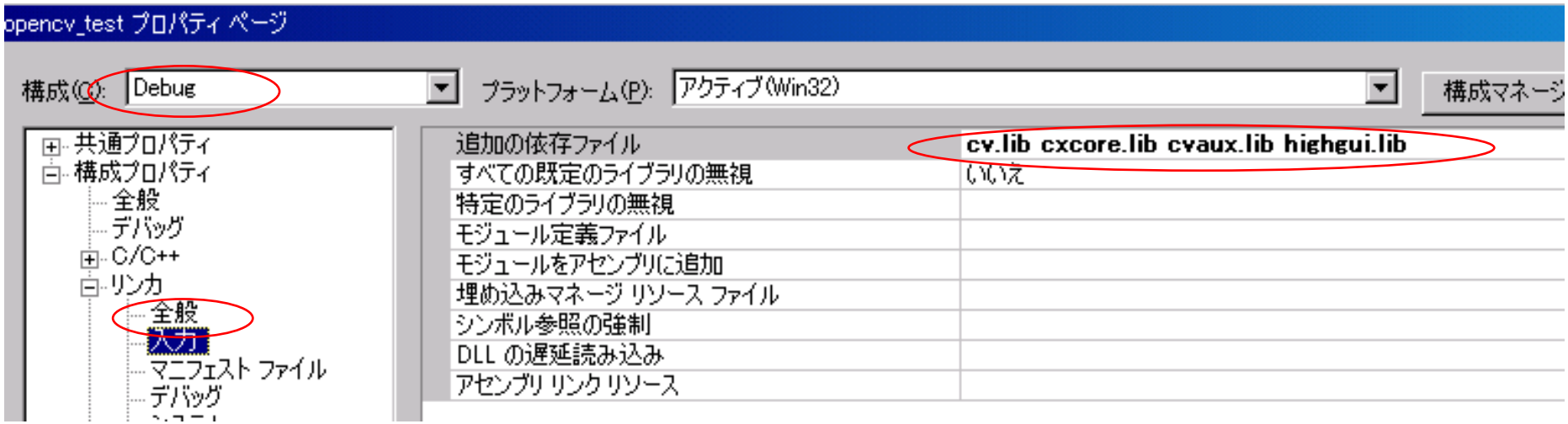
- See e.g.
  http://opencvlibrary.sourceforge.net/VisualC%2B%2B

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

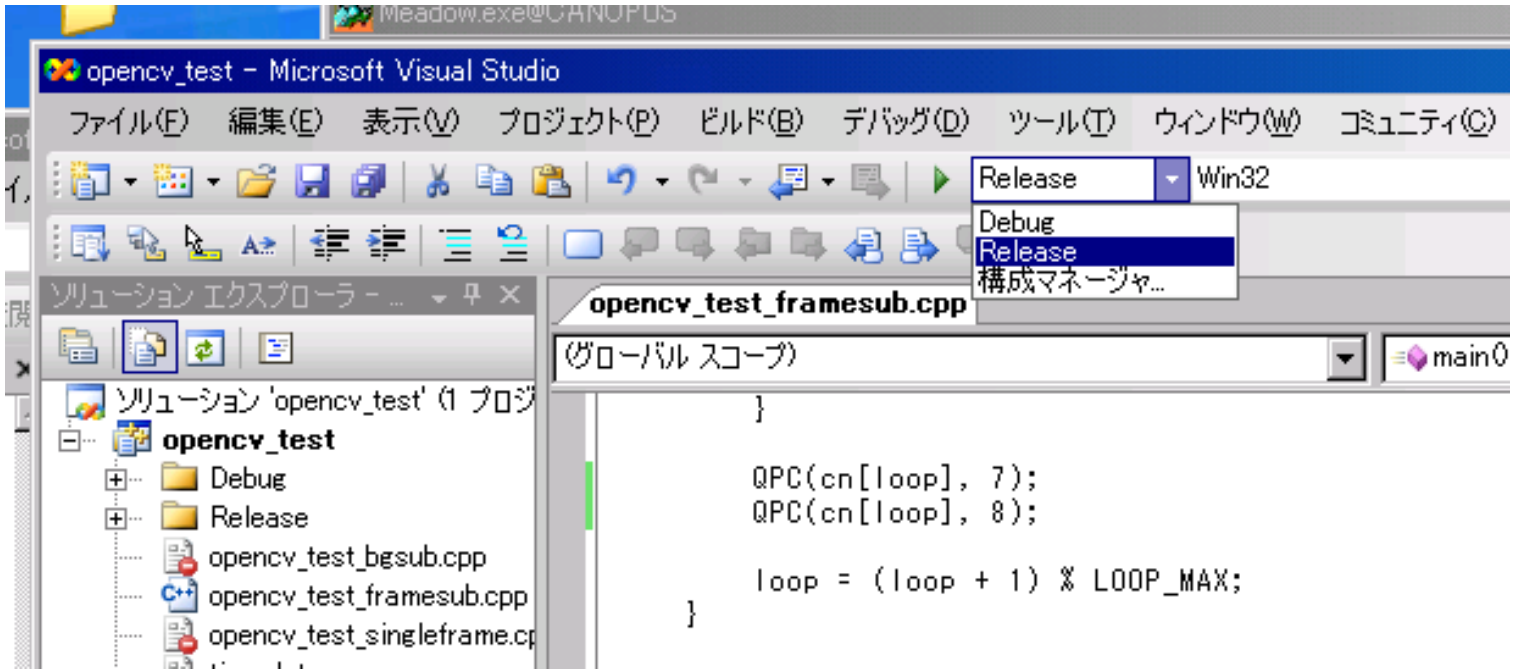
```
cv.lib cxcore.lib highgui.lib
(cvaux.lib cvcam.lib  ← 必要に応じて)
```

- Release構成でも設定しておくのを忘れないこと



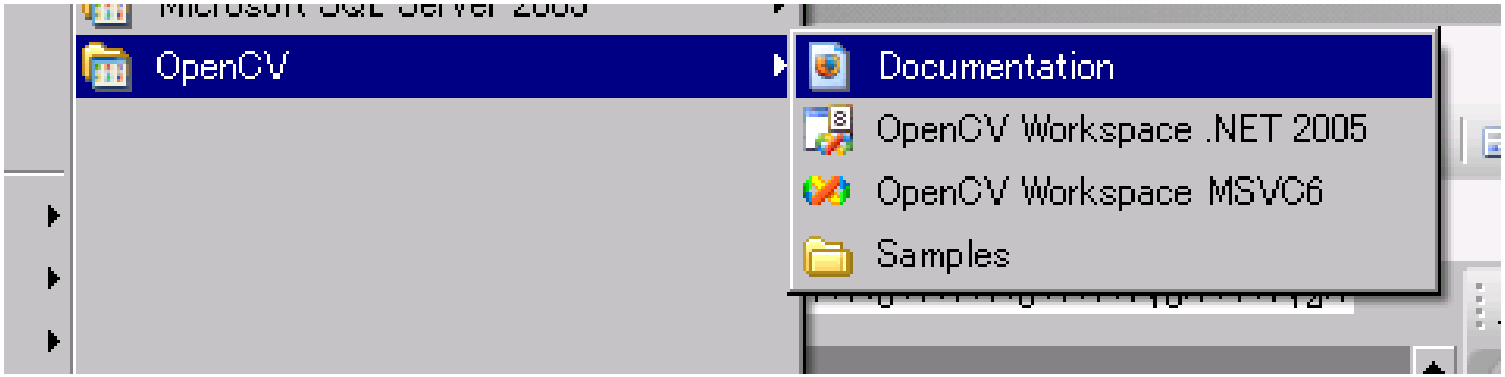
Debug 構成と Release 構成

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

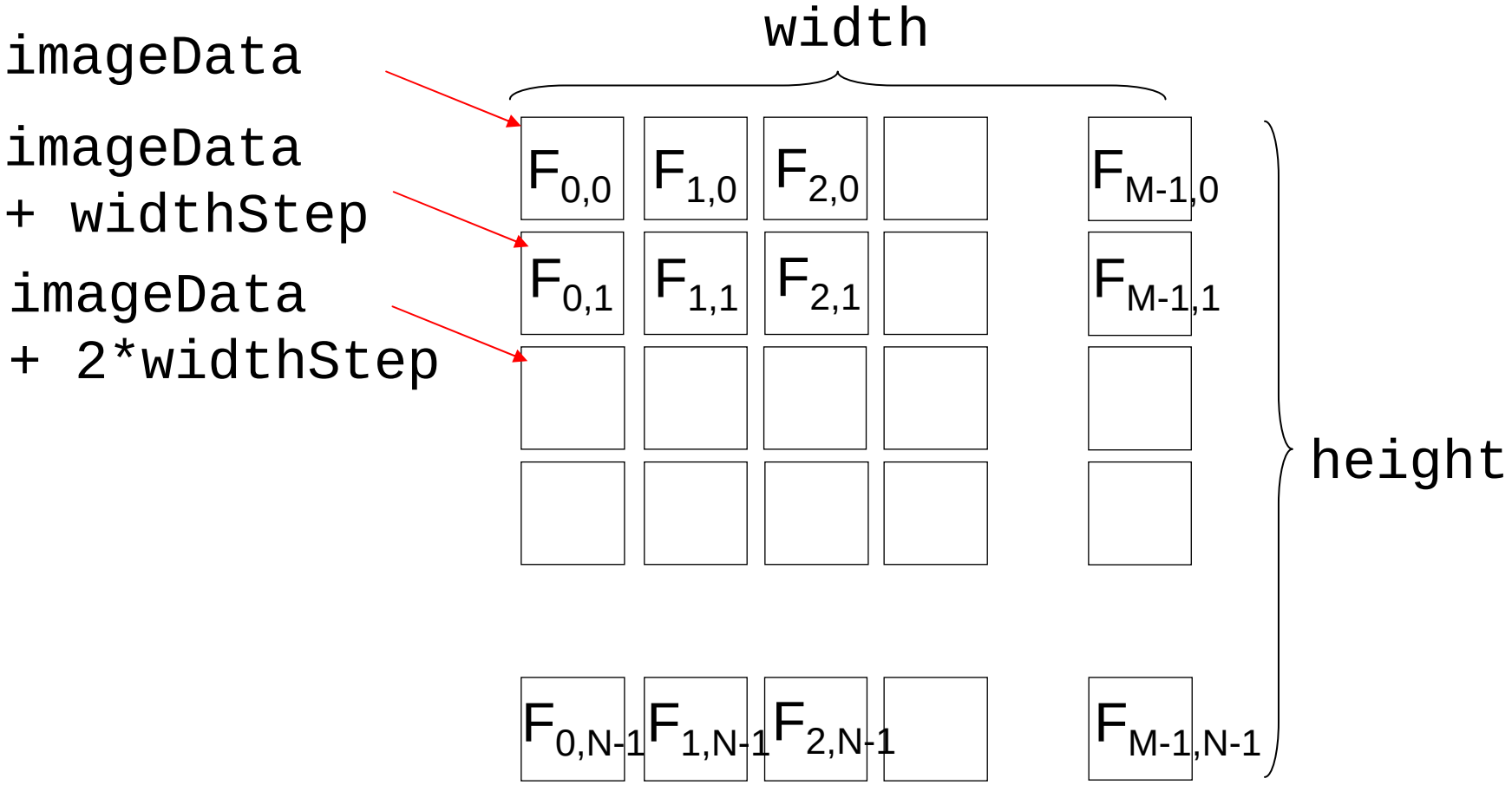
IplImage Structure

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

IplImage Structure

```
nChannel
   1: grayscale images
   3: RGB 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）



1-dimensional array of char continuous in the x (width) direction

画素データへのアクセス

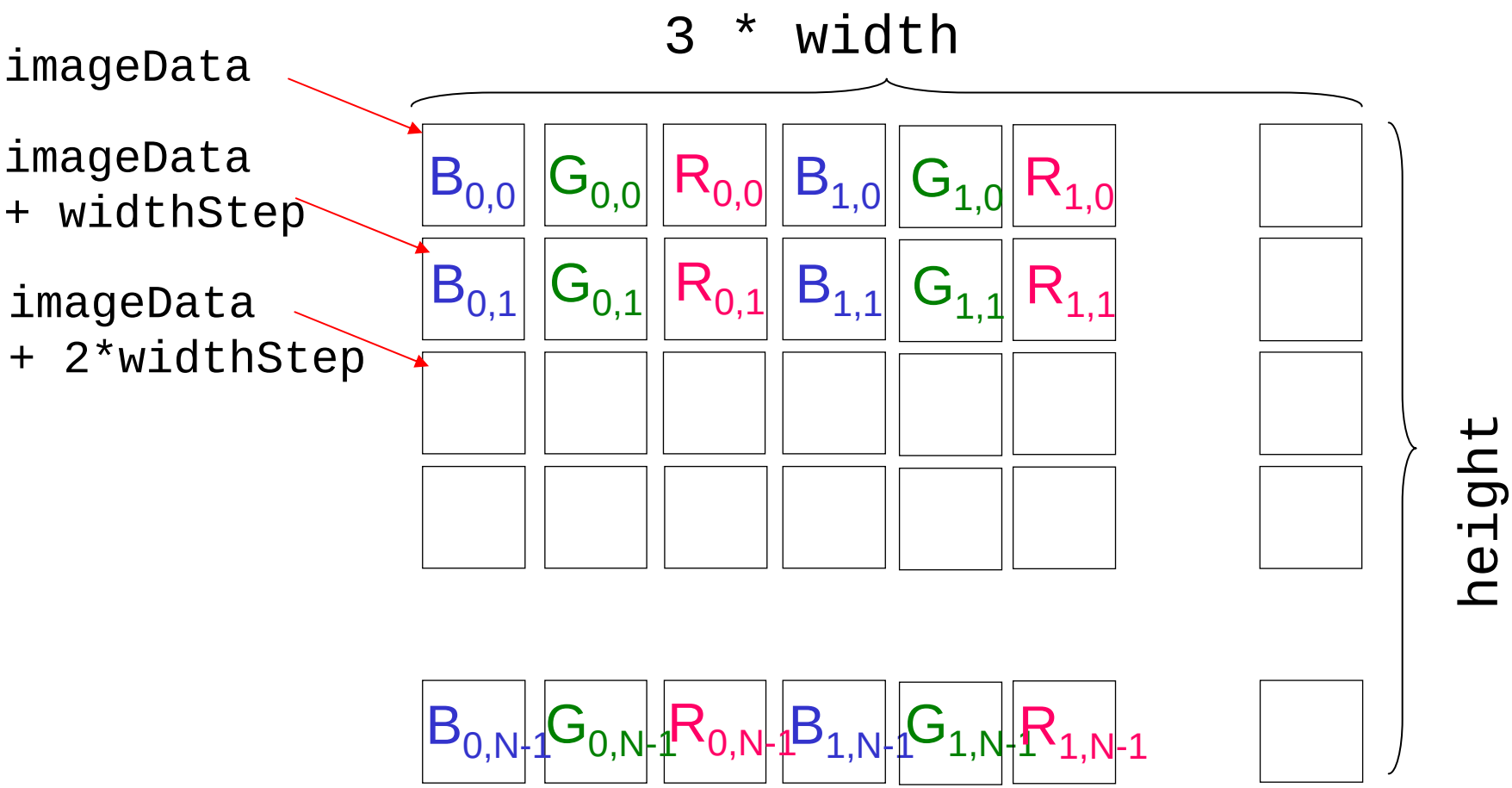
depth = IPL_DEPTH_8U, nChannel = 1 の場合

$F_{x,y}$ ~ (*(uchar *)(img->imageData
+ y * img->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，この方法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 > 1 の場合

$F_{x,y,c}$ ~ (*(uchar *)(img->imageData
+ y * img->widthStep
+ x * img->nChannel + c))

```
#define PIXVALC(iplimagep, x, y, c) ¥
  (*(uchar *)((iplimagep)->imageData ¥
   + (y) * (iplimagep)->widthStep +
   + (x) * (iplimagep)->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()
{
  // Initiallize

  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

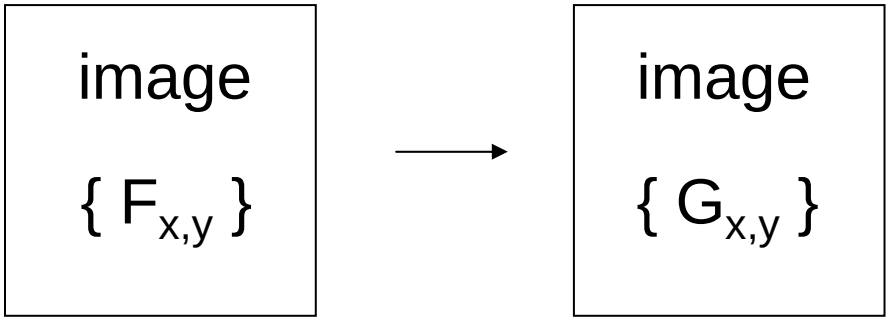
```
CvCapture *capture;
IplImage *frame;

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

画像処理の分類

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

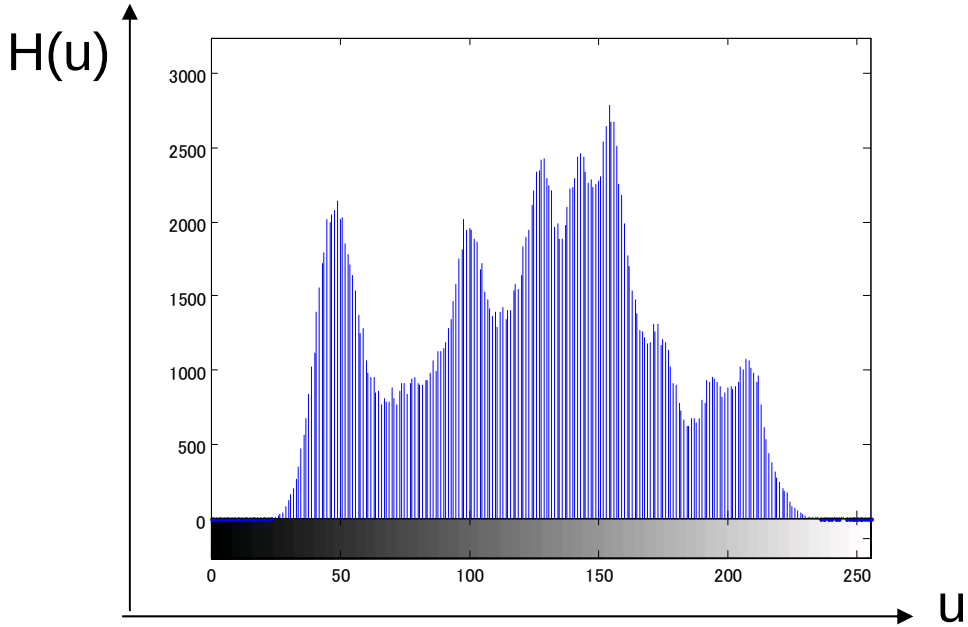
濃度変換，色変換など
- 2値化，色反転，ガンマ変換などが代表的な例

```
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

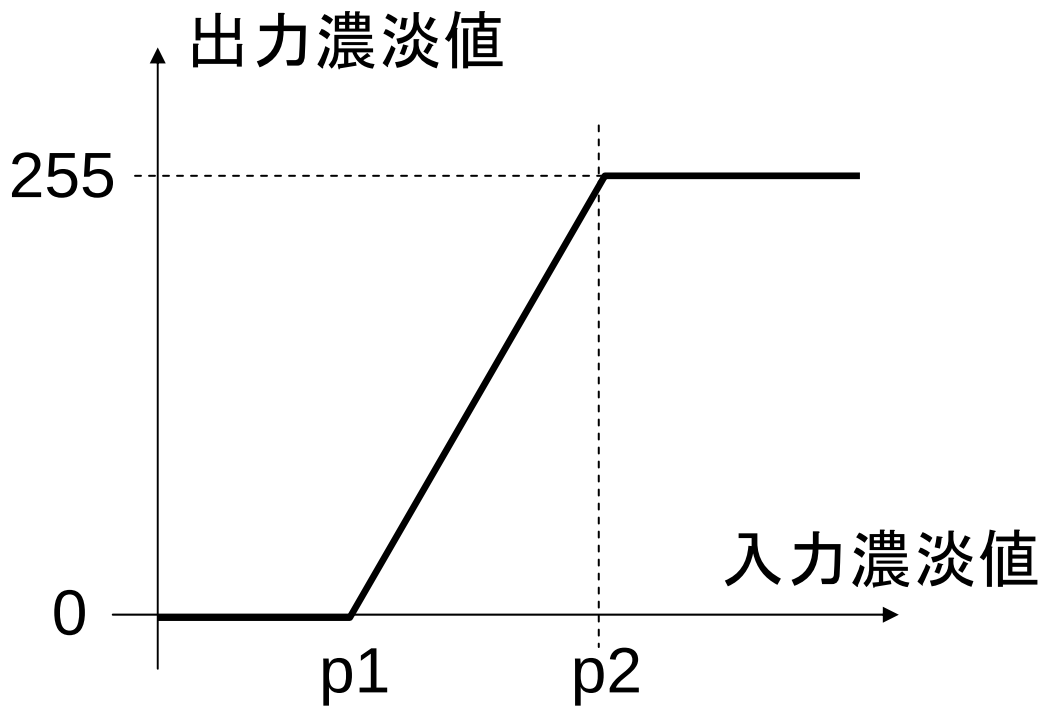
（濃淡値）ヒストグラム



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

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] http://sourceforge.net/projects/opencvlibrary/

[3] http://opencvlibrary.sourceforge.net/

[4] G. Bradski and A. Kaebler: Learning OpenCV, O'Reilly, 2008.

[5] 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, 毎日コミュニケーションズ, 2007.

[6] 永田: 実践OpenCV, カットシステム, 2009.