知能制御システム学

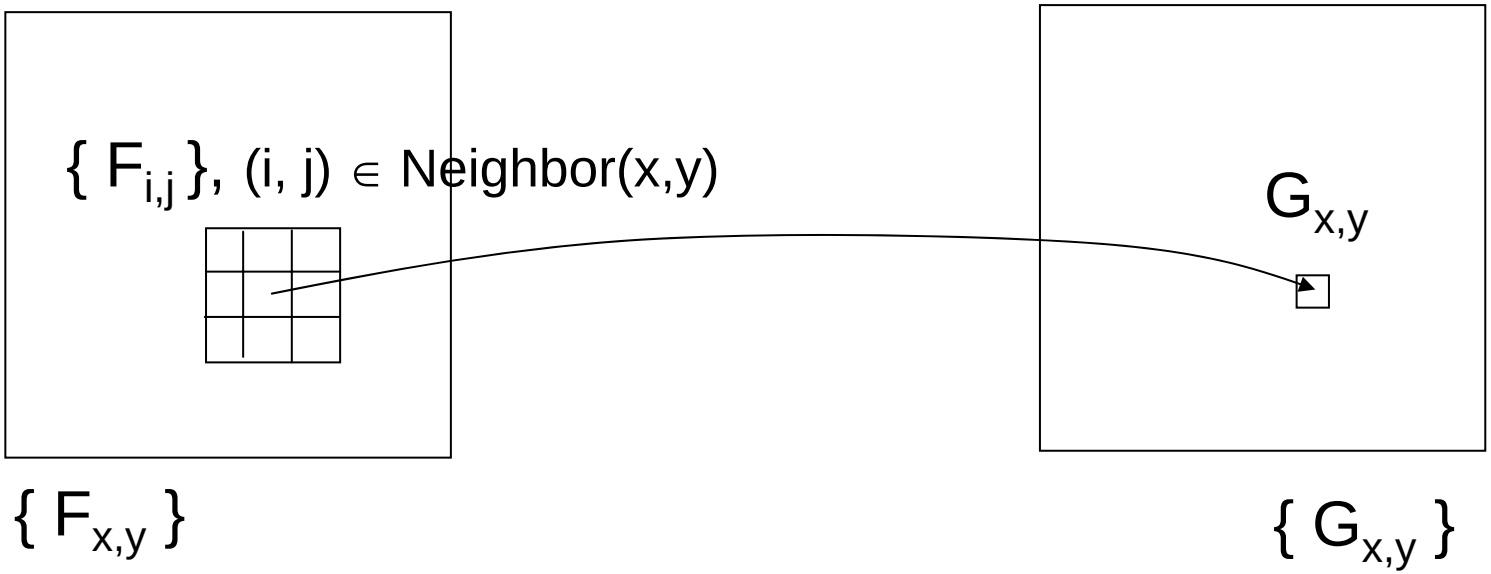
画像処理の基礎 (2)
— OpenCV による基本的な例 —

東北大学 大学院情報科学研究科
鏡 慎吾
swk(at)ic.is.tohoku.ac.jp

2010.07.06

局所処理の例 — 空間フィルタリング

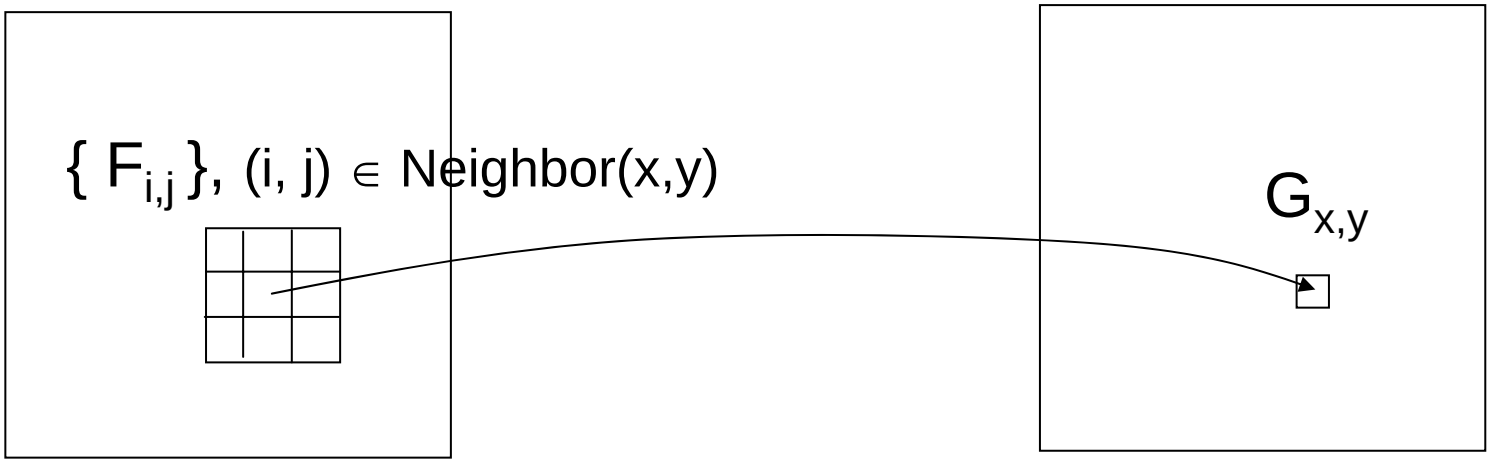
注目点の近傍（典型的には3x3画素，5x5画素, ... など）の画素値から，出力 $G_{x,y}$ を定める



典型例: 平滑化 (smoothing)，エッジ検出 (edge detection)

典型かつ重要な例: 平滑化 (smoothing)

- ノイズを除去したいときや，微細な構造を無視したいときに適用
- 近傍の画素値集合の代表値（平均など）を出力とする



- 例えば周囲の3x3画素 {$F_{i,j}$ }の平均を出力 $G_{x,y}$ とする（例1）
- あるいは，中心画素から遠いほど小さくなるような重みをつけた平均を出力とする（例2）

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

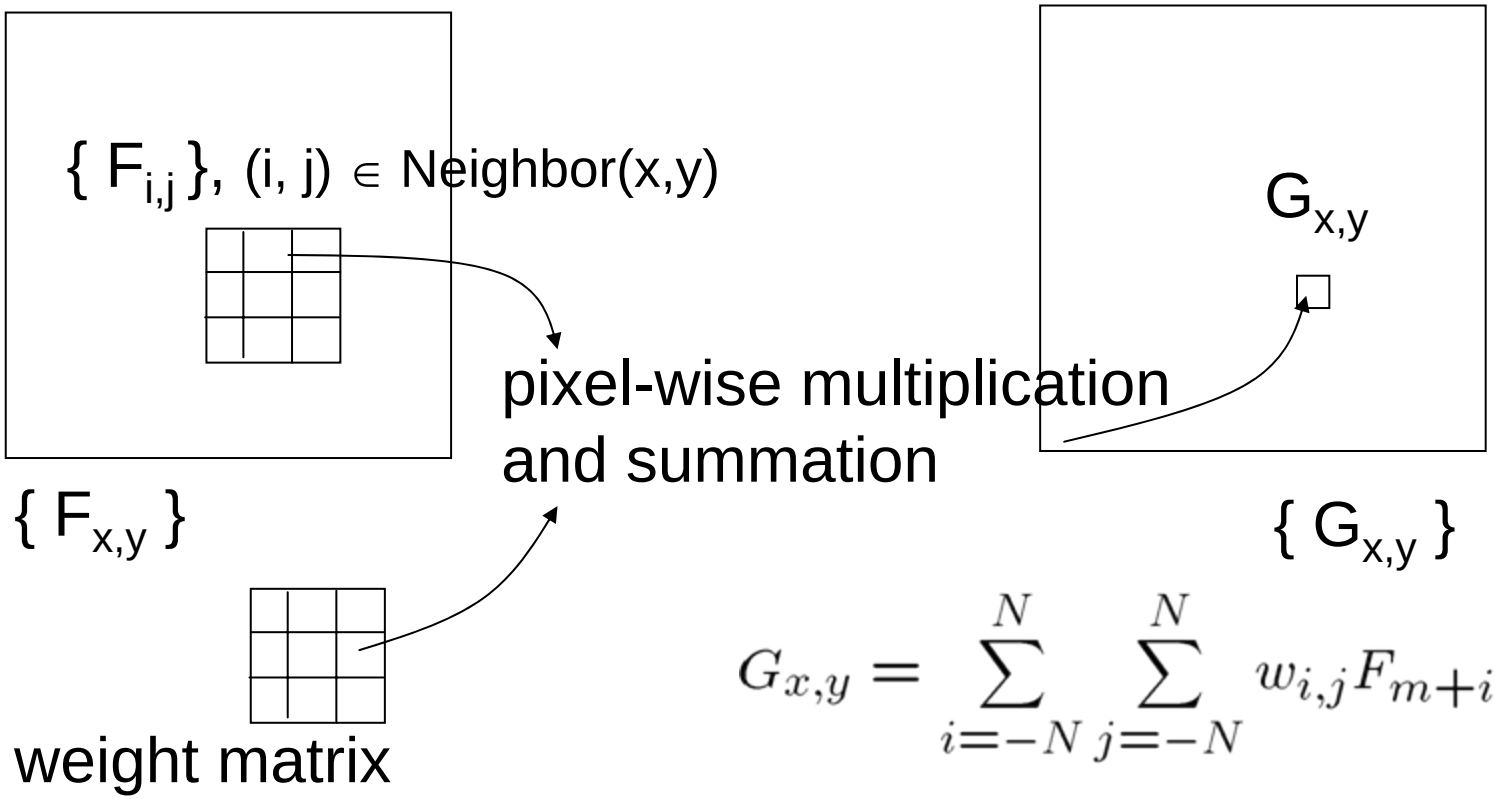
（例1）

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

（例2）

線形空間フィルタリング

- 前ページの例のように，空間フィルタのうち線形でシフト不変なもの線形空間フィルタ，あるいは単に線形フィルタと呼ぶ
- 加重マトリックス（マスク，カーネル，フィルタ係数，あるいは単にフィルタとも呼ばれる）のたたみこみとして表すことができる



$$G_{x,y} = \sum_{i=-N}^{N} \sum_{j=-N}^{N} w_{i,j} F_{m+i,n+j}$$

3x3平滑化の例

1	1	1
1	1	1
1	1	1

1	1	1
1	2	1
1	1	1

0	1	0
1	4	1
0	1	0

3x3線形フィルタのコード例

```
double w[3 * 3] = { 0.0,  1.0,  0.0,
                    1.0,  4.0,  1.0,
                    0.0,  1.0,  0.0 };
double scaling = 1.0 / 8.0;

for (j = 0; j < img->height; j++) {
    for (i = 0; i < img->width; i++) {
        double sum = 0.0;
        for (n = 0; n < 3; n++) {
            for (m = 0; m < 3; m++) {
                sum += pval(img, i - 1 + m, j - 1 + n)
                    * w[m + 3 * n];
            }
        }
        PIXVAL(result, i, j) = clipvalue(scaling * sum);
    }
}
```

sample program:
- filter3x3.cpp

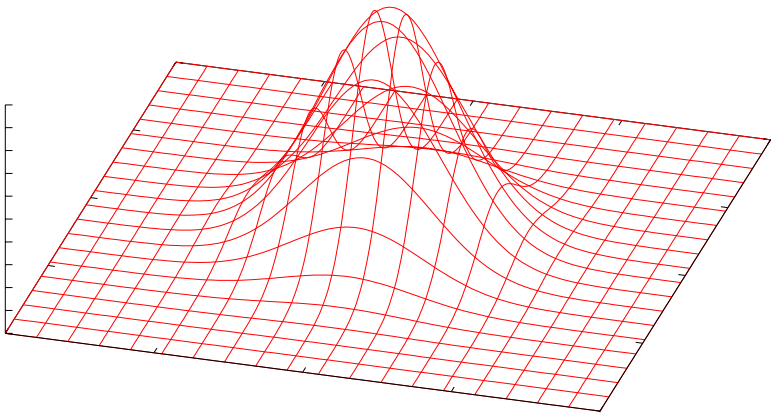
※ OpenCVでは実際には，cvFilter2D() を使うとよい．あるいは平滑化であれば cvSmooth() を使う方がよい場合もある．後述するエッジ検出等の例も cvSobel(), cvLaplace などを使い分けるとよい

- 画像の読み出しの際に，座標範囲を越えてしまわないように注意する（サンプルでは pval() の仕事）
- 画素値の範囲を越えてしまわないように注意する（サンプルでは clipvalue() の仕事）

ガウシアンによる平滑化

平滑化を実現するような加重マトリックスの中で，最も広く用いられているのが2次元ガウス関数（ガウシアン）である．

$$g_\sigma(x,y) = \frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{x^2}{2\sigma^2}\}\cdot\frac{1}{\sqrt{2\pi}\sigma}\exp\{-\frac{y^2}{2\sigma^2}\}$$
$$= \frac{1}{2\pi\sigma^2}\exp\{-\frac{x^2+y^2}{2\sigma^2}\}$$



- 適当に離散化して用いる
- 場合によって，加重値を適当に整数にまるめて用いる
- σによって平滑化の強さを表現できる（大きいσを使う場合にはマトリックスのサイズも大きくしないと正確に表現できない）

なぜガウシアンなのか? (1)

Marr and Hildreth (1980) の主張

- 理想的な平滑化フィルタは
  - 周波数領域でも滑らかでかつ局在しているべきである(平滑化とはそもそも帯域制限することである)
  - 空間領域で滑らかでかつ局在しているべきである(空間的な明るさの変化は, 照明変化や物体の境界などのように局所的なモノによって発生する)
- 周波数領域で局在していることと, 空間領域で局在していることは相反する要求である.
- この相反する要求を「両領域での分布の標準偏差の積が最小になる」という意味で最大限に満たすのがガウシアンである

なぜガウシアンなのか? (2)

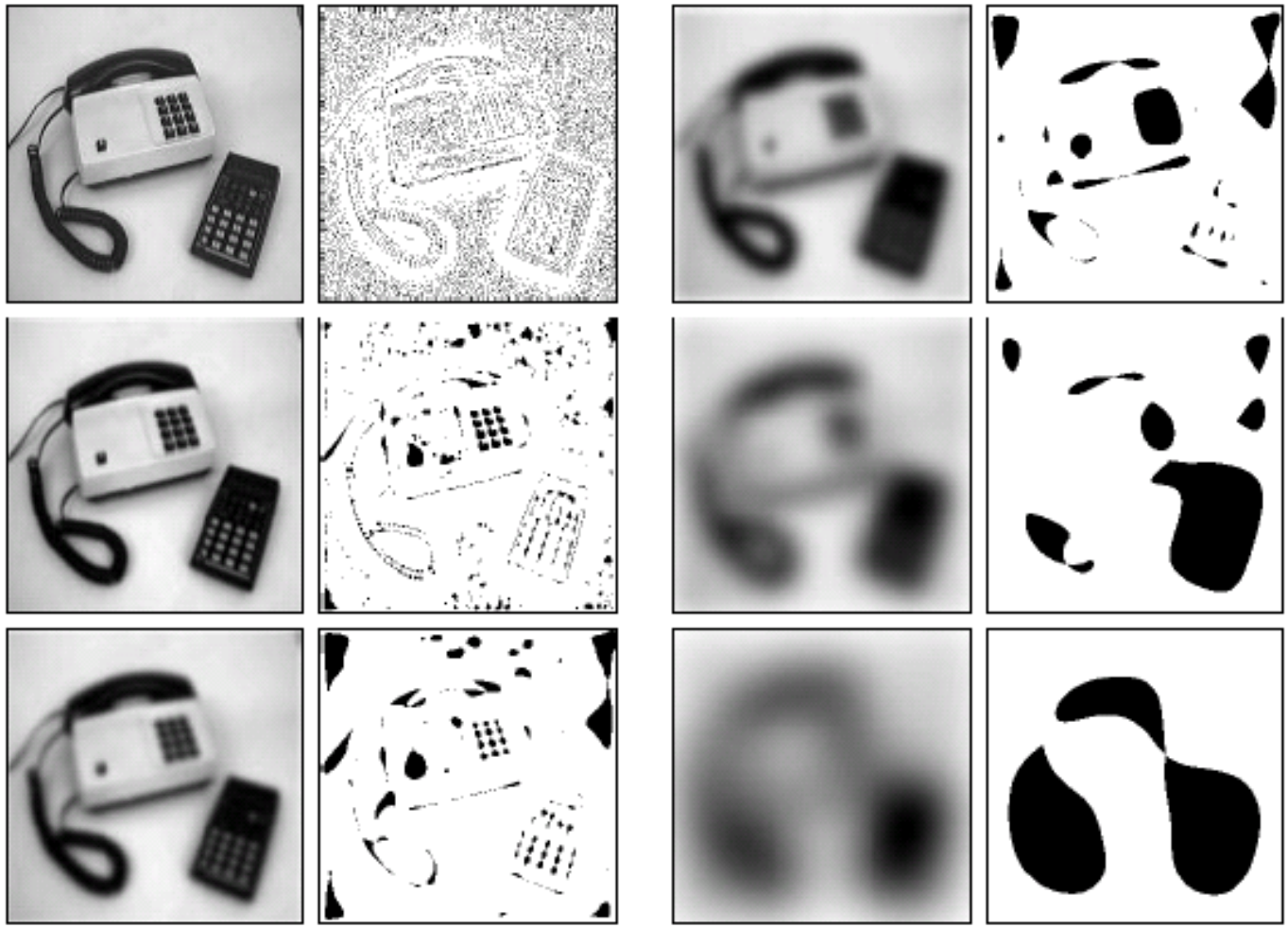
スケールスペース:

- 画像 I(x, y) をある量 t だけ平滑化した I(x, y; t) を考えて，この3次元空間で画像を理解しようという考え方
- この考え方のもとで，適当な条件を仮定すると，それらをすべて満たすような加重マトリックスはガウシアンに一意に定まる

例) Florack et al. (1992) の条件:

- 線形かつシフト不変
- スケール不変（適当に正規化した無次元量によって平滑化の処理を表現できる）
- 平滑化という操作が「半群」の構造を持つ（ある画像をある量 t_1 だけ平滑化してからさらに t_2 だけ平滑化した結果は，元の画像を $t_1 + t_2$ だけ平滑化したのと等価になる）
- 等方的である

スケールスペース



[Lindeberg 1996]

エッジ検出 (edge detection) の例

- 空間微分（実際には差分）を計算する

-1	0	1
-1	0	1
-1	0	1

単純な水平方向
1次微分

-1	-1	-1
0	0	0
1	1	1

単純な垂直方向
1次微分

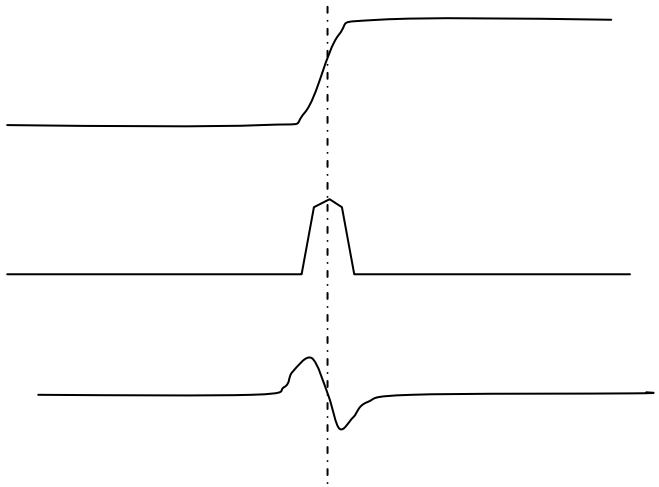
-1	0	1
-2	0	2
-1	0	1

注目画素の近くに重み
(Sobel フィルタ)

-1	-2	-1
0	0	0
1	2	1

2階微分によるエッジ検出

- あるいは2階微分を計算してそのゼロ交差を求める
- ラプラシアン $\partial^2/\partial x^2 + \partial^2/\partial y^2$ は最低次元の等方性微分演算子であり，方向によらずにエッジを得ることができる.



- 1次元の2階微分（差分） $f_{i+1} - 2f_i + f_{i-1}$ の縦横方向を足し合わせた 3x3マトリックスによってラプラシアンを表現できる

0	1	0
1	–4	1
0	1	0

LoG フィルタと DoG フィルタ

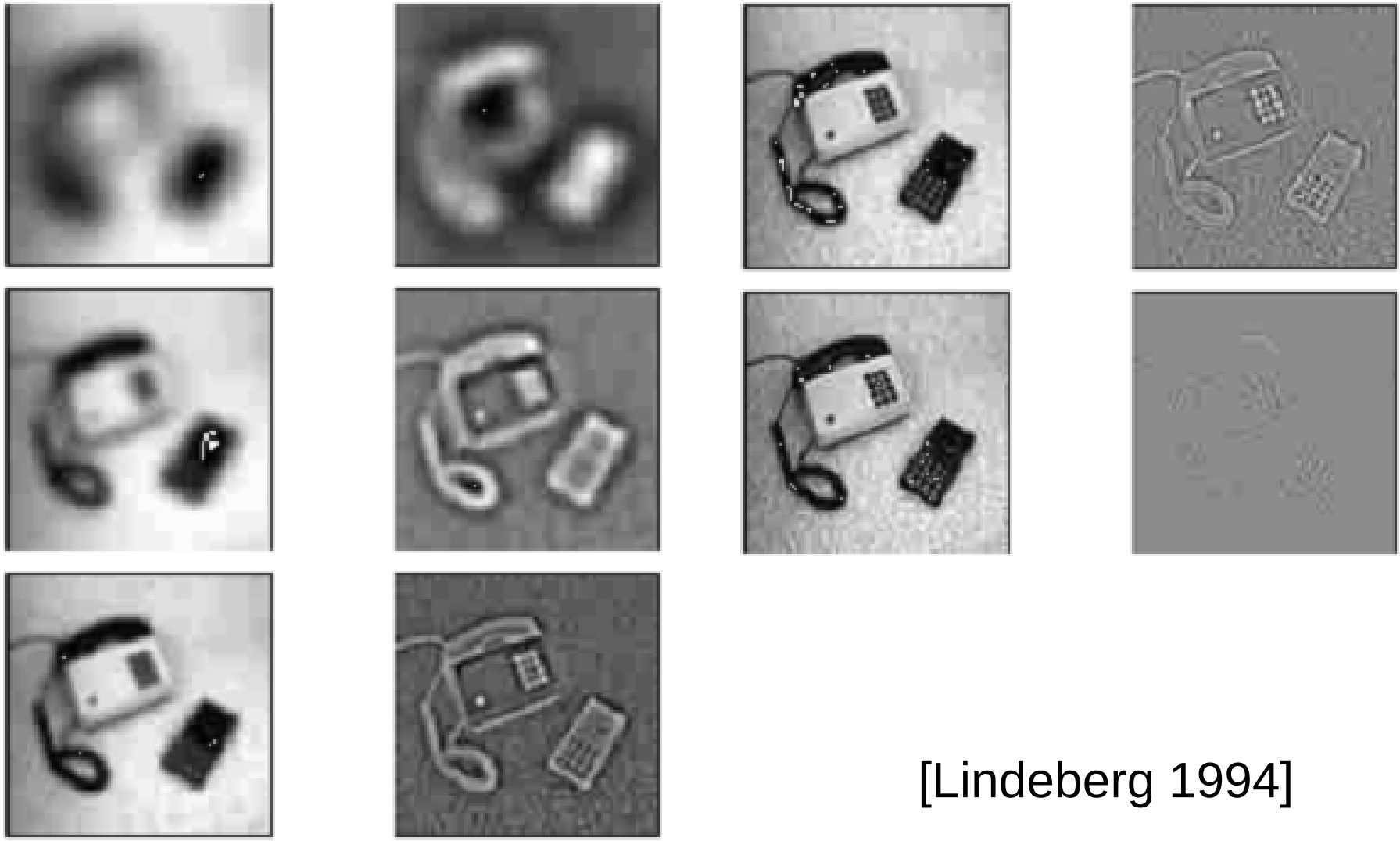
- ガウシアンフィルタによって段階的に平滑化した画像に対してラプラシアンフィルタを適用することで，異なるスケールでの画像特徴を抽出することができる．Laplacian of Gaussian (LoG) フィルタと呼ばれる．

- LoG は，平滑化の量がわずかに異なる2つのガウシアンフィルタ画像の差分として近似できる．Difference of Gaussian (DoG) と呼ばれる．スケールスペースの画像群がそもそも必要な場合は，こちらの方が効率がよい．

$$\mathrm{DoG}(\cdot, t) = G(\cdot, t + \delta t) - G(\cdot, t)$$

- 脊椎動物の視覚野の反応が DoG でよく近似できることが報告されている．

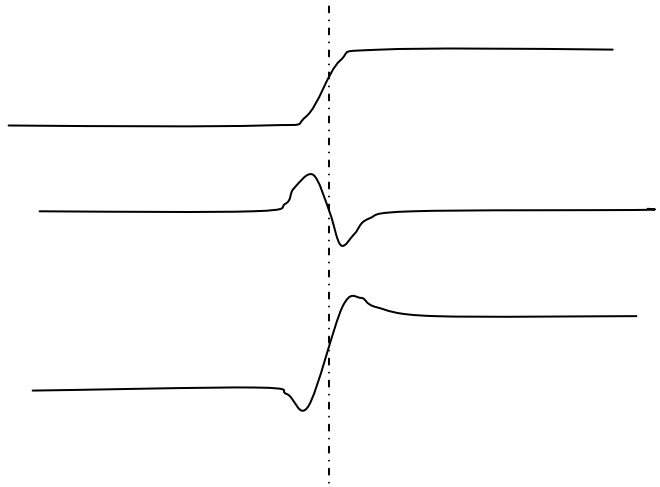
LoG の例



[Lindeberg 1994]

先鋭化 (sharpening) の例

ラプラシアン画像を元画像から引くことで高域強調の効果を得る



0	0	0
0	8	0
0	0	0

−

0	1	0
1	–4	1
0	1	0

=

0	-1	0
-1	12	-1
0	-1	0

2値画像処理

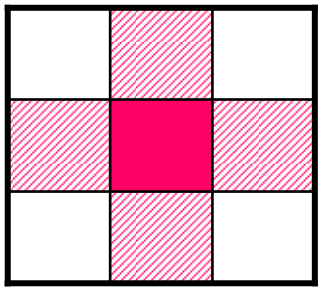
2値化された後の画像 (binary image) の処理は，それだけで一分野をなすほど独特に発展している.

- 実用上重要であった
- 幾何学的に明確な議論がしやすく，体系的な議論が進んだ

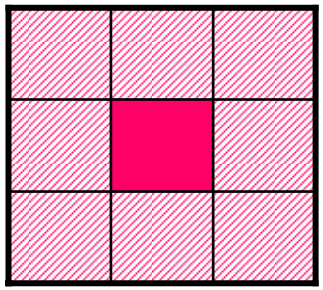
- 画素に離散化されているため，従来の連続な図形の幾何学で考えられてきた「連結」や「距離」の概念を考え直す必要がある → ディジタル幾何学

連結性

近傍 (neighbor): ある画素の近くにある画素の集合. いろいろな定義が可能.



4-neighbor



8-neighbor

互いに n-近傍の関係にある 2 つの画素は「n-隣接している」(n-adjacent) と呼ばれる.

同じ画素値を持つ2つの画素 a, b に対して画素の系列 p_0 (= a), p_1, p_2, ..., p_{n-1}, p_n (= b) が存在し, p_i はすべて同じ画素値を持ち, p_i と p_{i-1} が n-隣接するとき, 画素 a と b は「n-連結している」(n-neighbor connected) という

数学的モルフォロジ

dilation: 近傍の誰かが 1 だったら自分も 1 になる

$$G_{i,j} = F_{i,j} \mid F_{i-1,j} \mid F_{i+1,j} \mid F_{i,j-1} \mid F_{i,j+1} \qquad \text{(4-近傍の場合)}$$

erosion: 近傍の誰かが 0 だったら自分も 0 になる

$$G_{i,j} = F_{i,j} \,\&\, F_{i-1,j} \,\&\, F_{i+1,j} \,\&\, F_{i,j-1} \,\&\, F_{i,j+1} \qquad \text{(4-近傍の場合)}$$

opening: erosion + dilation

closing: dilation + erosion

sample program:
•morph.cpp

※ OpenCVでは実際には，cvErode(), cvDilate() などを使うとよい

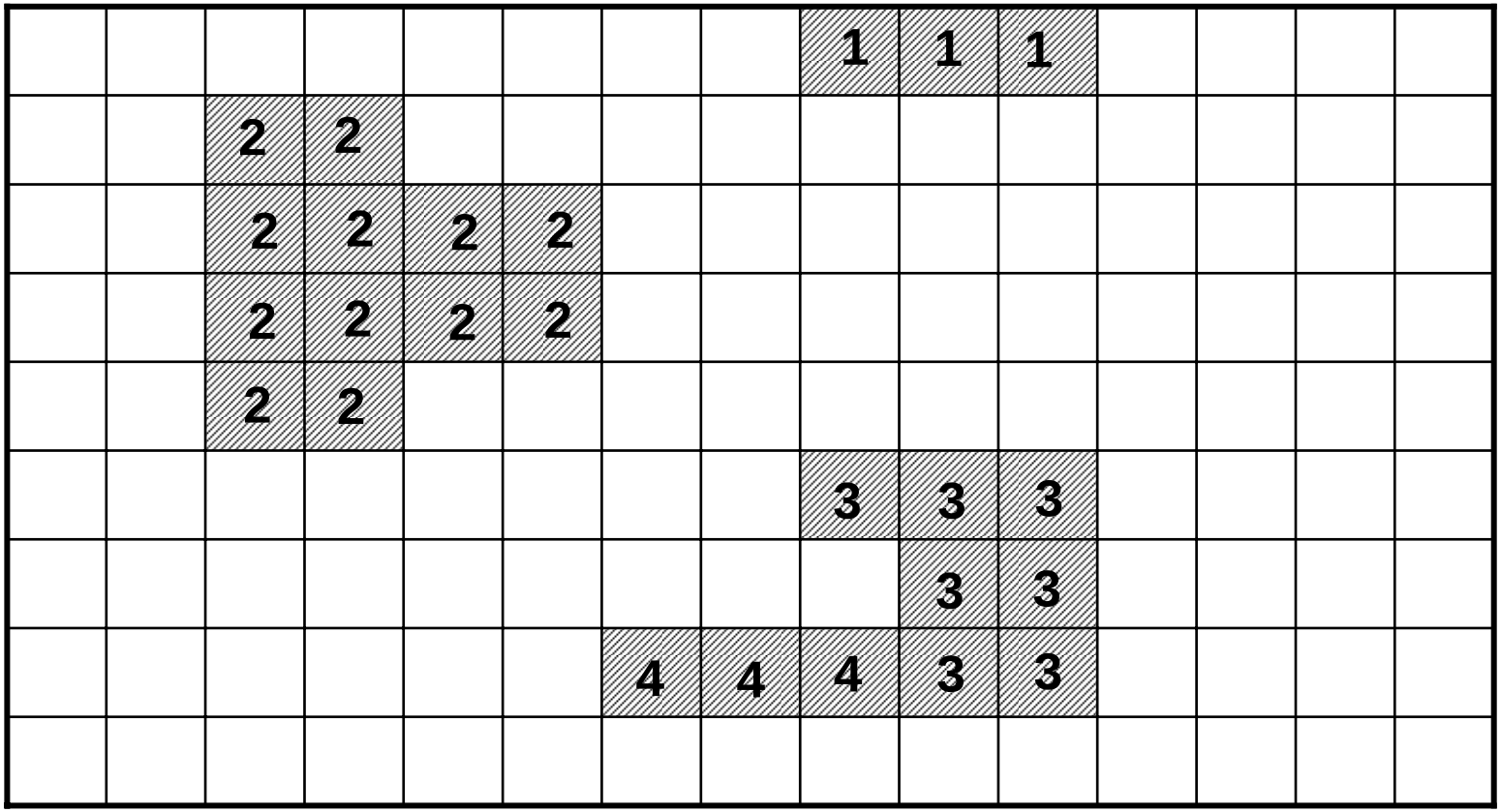
連結領域ラベリング

- あるいは単にラベリングとも呼ぶ.
- 2値画像を個々の連結領域に分割し，連結領域ごとに異なる番号（ラベル）を割り当てる.
- ラベルを画素値とする画像をラベル画像と呼ぶ.

sample program:
•labeling.cpp

4-連結の場合の処理の例:

- 左上→右下に順に画像をスキャン．画素値0ならスキップ.
- 上画素，または左画素にラベルが割り当て済みならそのラベルを現在画素に割り当てる.
  - ただし，上画素と左画素に異なるラベルが割り当て済みの場合は，どちらか一方（例えば小さい方）のラベルを現在画素に割り当て，両ラベルが同一の領域を指すことを記録.
- 上画素，左画素ともラベル無しなら新規ラベルを割り当てる.
- 全画素スキャン後，同一領域の情報を整理してから，再度スキャンしてラベル画像の画素値を書き換える



3 ⇔ 4

画像からスカラ値への変換

モーメント特徴

$$m_{p,q} = \sum_i \sum_j i^p j^q F_{i,j}$$

$$m_{0,0} = \sum_i \sum_j F_{i,j}$$ 0次モーメント: 2値画像の場合，面積

$$m_{1,0} = \sum_i \sum_j i F_{i,j}$$ x方向の1次モーメント

$$m_{0,1} = \sum_i \sum_j j F_{i,j}$$ y方向の1次モーメント

0～1次モーメントまでの情報から，面積と重心が分かる．

$(g_x, g_y) = (m_{1,0}/m_{0,0}, m_{0,1}/m_{0,0})$

さらに高次のモーメントから，形状に関するより詳細な情報が得られる．

sample program:
•moment.cpp

※ OpenCV では cvMoments() が便利な場合もある．

画像処理プログラミングと高速化

- 「リアルタイム」=「高速」ではない
- 目標となる時間制約が定められているのがリアルタイム処理である
  - 34 ms → 33 ms
  - 33 ms → 32 ms
  - どちらも 1 ms の短縮だが，例えばビデオレートでのリアルタイム処理が目的なのであれば，両者の意味は大きく違う
- 平均実行時間 ←→ 最悪実行時間
- 計算時間を短くするには計算量を減らすのが基本．**だがそれだけではない**

実行時間を手軽に測る

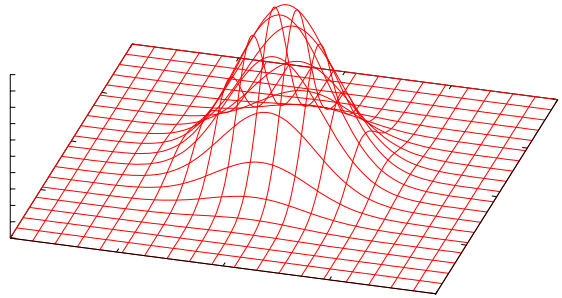
```
#include <windows.h>

LARGE_INTEGER freq;
LARGE_INTEGER cn1, cn2;
double elapsed_time_in_microseconds;

if (QueryPerformanceFrequency(&freq) == 0) {
        fprintf(stderr, "cannot use performance counter¥n");
        return 1;
}

QueryPerformanceCounter(&cn1);

// do something

QueryPerformanceCounter(&cn2);

elapsed_time_in_microseconds =
 (1000000.0 * (cn2.QuadPart – cn1.QuadPart)) / freq.QuadPart);
```

例: ガウス平滑化

- m × n のフィルタマスクによるガウス平滑化は，単純に実行すると 1 画素当たり O(mn) の計算量が必要

$$G_{x,y} = \sum_i \sum_j w_{i,j} F_{m+i,n+j}$$

$$w_{x,y} = \frac{1}{2\pi\sigma^2} \exp\{-\frac{x^2+y^2}{2\sigma^2}\}$$



- X方向・Y方向それぞれ1次元ガウス平滑化に分解できることを思い出せば，1画素当たり O(max(m,n)) の計算量で実行可能

$$w_{x,y} = \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{x^2}{2\sigma^2}\} \cdot \frac{1}{\sqrt{2\pi}\sigma} \exp\{-\frac{y^2}{2\sigma^2}\}$$

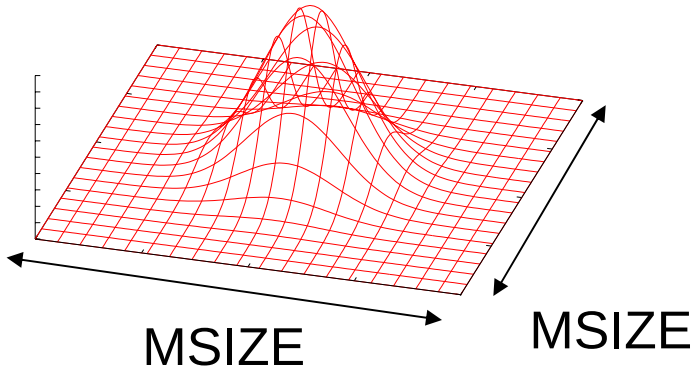
- このような性質を持つフィルタを「分離可能 (separable)」という

実行例

- time_gauss.cpp
  (時間計測は PerformanceCounter を用いている. ソースコードは perftimer.cpp)

```
//#define OPT_NOSEP
#define OPT_SEP
```

で，分離型，非分離型を切り替え




```
#define MARGIN 7
```

MSIZE = 2 * MARGIN + 1

でフィルタマスクのサイズを指定

わかったこと

- 計算量が減ったからといって，そのまま計算時間の減少につながるわけではない．
- それどころか，条件によっては遅くなる場合すらある．

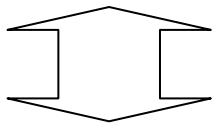
- 画像処理のように，扱うデータ量が比較的多い場合，メモリアクセスが律速になることが多い．
- PCのCPUで画像処理を行う場合，キャッシュメモリの特性を考慮することが重要．
- キャッシュメモリの前提:
  - 時間的局所性
  - 空間的局所性

ループ交換

```
#define OPT_INTERCHANGE
```

メモリアクセスのパターンによって速度が変わる例．キャッシュの効果が現れやすい．

```
for (j = 0; j < height; j++) {
    for (i = 0; i < width; i++) {
        PIXVAL(img, i, j) = ...

    }
}
```



```
for (i = 0; i < width; i++) {
    for (j = 0; j < height; j++) {
        PIXVAL(img, i, j) = ...
    }
}
```

sample program
- time_framediff.cpp
  (同様に perftimer.cpp が必要)

画素アドレス計算の最適化

```
#define OPT_INDEX
```

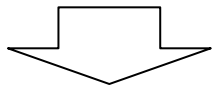
画素データが格納されているメモリアドレスを計算する部分は，普通画像処理アルゴリズムの計算量に含めて考えないが，実は馬鹿にできない．毎回計算するのは（特にシーケンシャルにアクセスする場合は）無駄であり，可読性を犠牲にすればより高速化可能．

つまり，この講義で多用している PIXVAL マクロは，性能面からいうとあまり使うべきではない．

sample program
- time_framediff.cpp
  (同様に perftimer.cpp が必要)

```
#define PIXVAL(img, i, j)
(*(uchar *)((img)->imageData + (y) * (img)->widthStep + (x)))
```

```
for (j = 0; j < height; j++) {
    for (i = 0; i < width; i++) {
        PIXVAL(img, i, j) = ...
    }
}
```



```
uchar *ptr;
int gap = img->widthStep – width;

for (j = 0, ptr = img->imageData; j < height; j++, ptr += gap) {
    for (i = 0; i < width; i++, ptr++) {
        *ptr = ...
    }
}
```

References

[1] 田村: コンピュータ画像処理, オーム社, 2002.

[2] T. Lindeberg: Scale-space: A framework for handling image structures at multiple scales, Proc. CERN School of Computing, 1996.

[3] T. Lindeberg: Scale-space theory: A basic tool for analysing structures at different scales, J. Applied Statistics, vol.21, no.2, pp.225-270, 1994.

[4] D. Marr: ビジョン — 視覚の計算理論と脳内表現 —, 産業図書, 1987.

[5] D. Marr and E. Hildreth: Theory of Edge Detection, Proc. Roy. Soc. London, Series B, Biological Sciences, vol.207, no.1167, pp.187-217, 1980.

[6] L. M. J. Florack, B. M. T. Romeny, J. J. Koenderink and M. A. Viergever: Scale and the Differential Structure of Images, Image and Vision Computing, vol.10, no.6, pp.376-388, 1992.