

# デジタルメディア処理

担当: 井尻 敬

## フィルタ処理2：非線形フィルタ，ハーフトーニング

### 達成目標

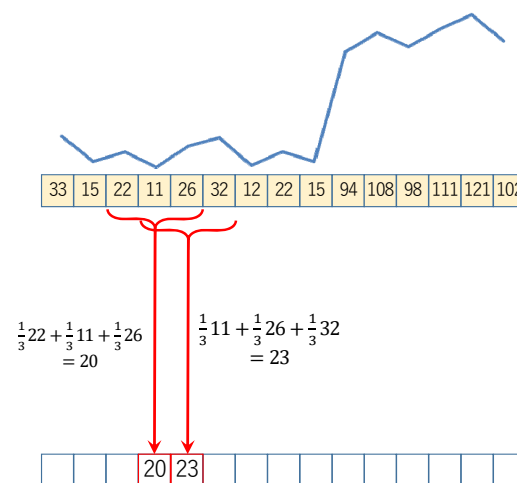
- 非線形フィルタ処理（エッジ保存平滑化フィルタ）の計算法と効果を説明できる
- ハーフトーン処理の計算法と効果を説明できる

### Contents

- 線形フィルタの復習
- 非線形フィルタ
- ハーフトーニング

## 線形フィルタの復習

### 線形フィルタの例 1D



復習

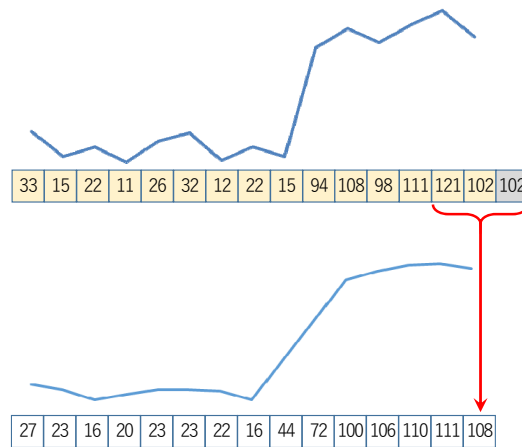
平滑化したい!

1/3 1/3 1/3

周囲3ピクセルの平均を取る

## 線形フィルタの例 1D

復習



※端ははみ出すので値をコピー（ほかの方法もある）

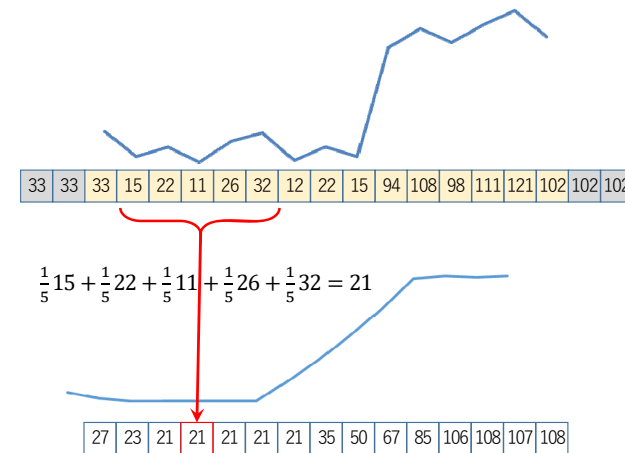
平滑化したい!

$\frac{1}{3} \quad \frac{1}{3} \quad \frac{1}{3}$

周囲 3 ピクセル  
の平均を取る

## 線形フィルタの例 1D

復習



$$\frac{1}{5} 15 + \frac{1}{5} 22 + \frac{1}{5} 11 + \frac{1}{5} 26 + \frac{1}{5} 32 = 21$$

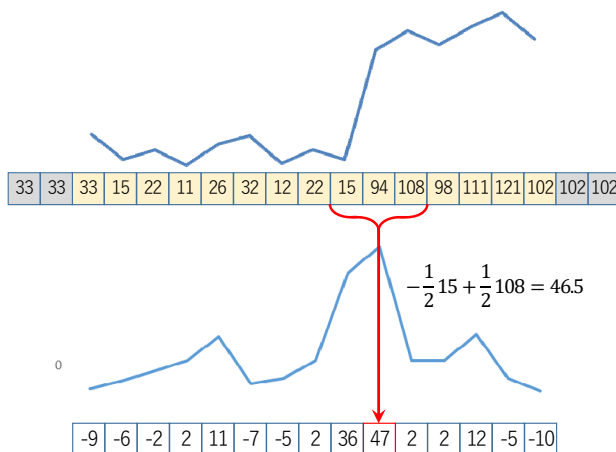
もっと  
平滑化したい!

$\frac{1}{5} \quad \frac{1}{5} \quad \frac{1}{5} \quad \frac{1}{5} \quad \frac{1}{5}$

周囲5ピクセル  
の平均を取る

## 線形フィルタの例 1D

復習



※端ははみ出すので値をコピー（ほかの方法もある）

エッジ  
(変化の大きい部分)  
を検出したい

$-0.5 \quad 0 \quad 0.5$

右と左のピクセルの  
差をとる

## Convolution(畳み込み)とは

予習・復習

二つの関数  $f(t)$   $g(t)$  を重ね合わせる演算で以下の通り定義される

連続関数  $(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$

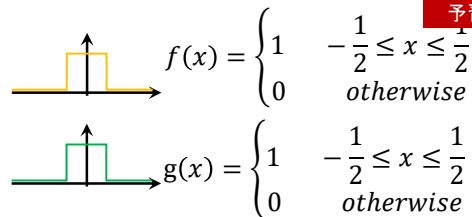
離散関数  $(f * g)(n) = \sum_{k=-\infty}^{\infty} f(k)g(n-k)$

$f(t)$  を固定し,  $g(t)$  を平行移動しながら  $f(t)$  に掛けあわせ, 得られた関数を積分するとみてもよいかも

### 例題)

2つの関数 $f, g$ の畳み込みを求めよ

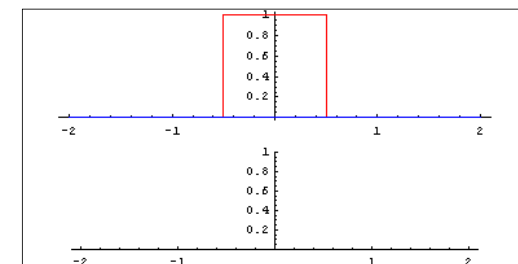
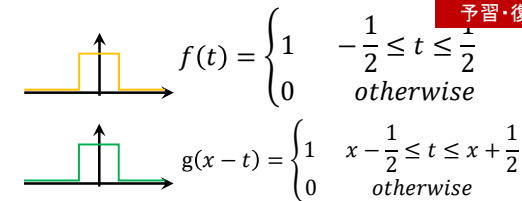
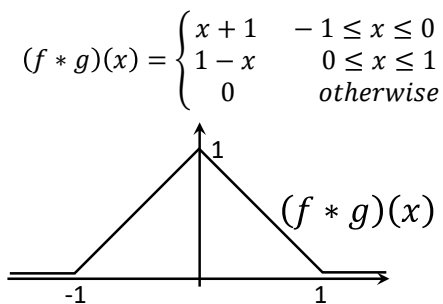
$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



### 例題)

2つの関数 $f, g$ の畳み込みを求めよ

$$(f * g)(x) = \int_{-\infty}^{\infty} f(t)g(x-t)dt$$



By Lautaro Carmona [CC-BY-SA] from wikipedia

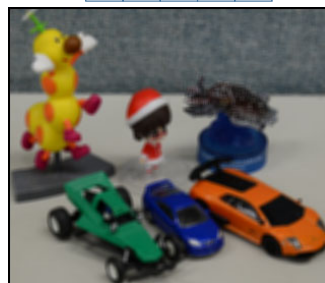
## 線形フィルタ：平滑化

$$\frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$$\frac{1}{25}$$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1



## 線形フィルタ：エッジ検出（微分）

- 前述の単純なフィルタはノイズにも鋭敏に反応する
- ノイズを押さえつつエッジを検出するフィルタが必要

横方向微分：横方向微分 し 縦方向平滑化 する

縦方向微分：縦方向微分 し 横方向平滑化 する

Prewitt filter

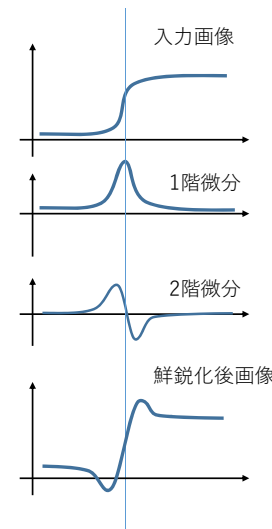
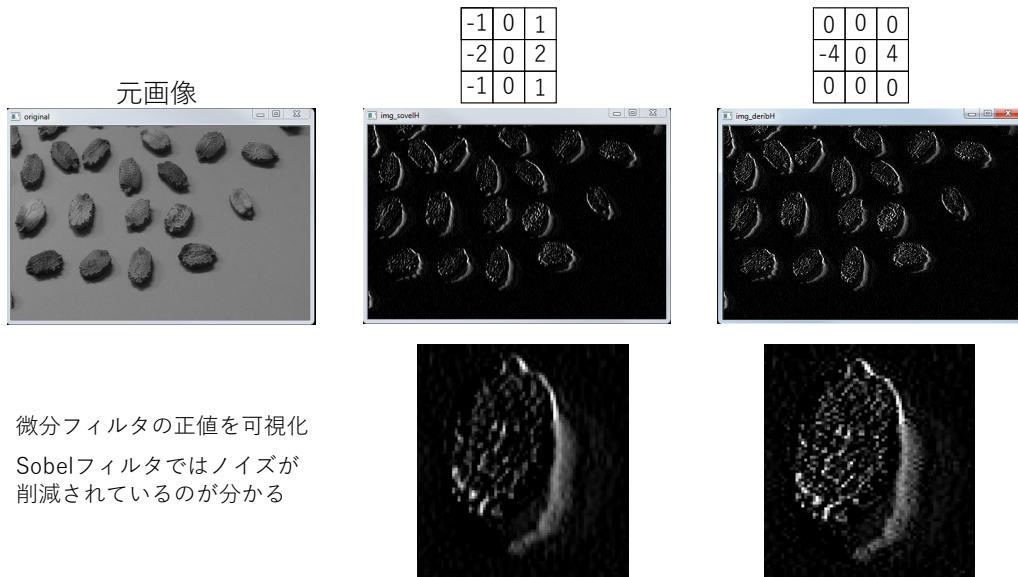
-1	0	1
-1	0	1
-1	0	1

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

Sobel filter

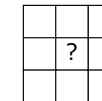
-1	0	1
-2	0	2
-1	0	1

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



## 線形フィルタ：鮮鋭化フィルタ

2回微分に関するラプラシアンフィルタを改良すると画像のエッジを強調する鮮鋭化フィルタが設計できる



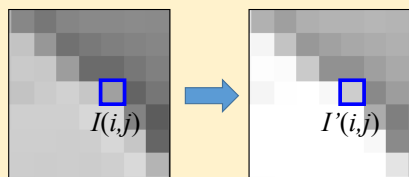
## 空間フィルタとは

- 空間フィルタとは周囲の情報を利用して画素値を決めるフィルタ
- 空間フィルタは、線形フィルタと非線形フィルタに分けられる

### トーンカーブ：

出力画素  $I'(i,j)$  を求めるのに  
入力画素  $I(i,j)$  のみを利用

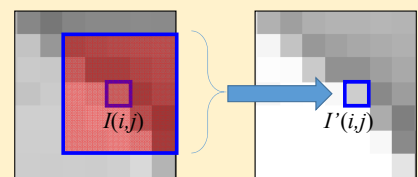
入力画像： $I(i,j)$       出力画像： $I'(i,j)$



### 空間フィルタ：

出力画素  $I'(i,j)$  を求めるのに  
入力画素  $I(i,j)$  の周囲画素も利用

入力画像： $I(i,j)$       出力画像： $I'(i,j)$

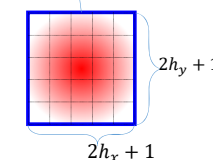
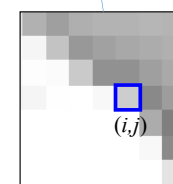


復習

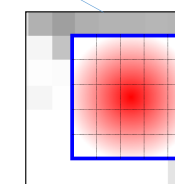
## 線形フィルタとは

出力画素値を、入力画像の周囲画素の重み付和で計算する

$$I'(i,j) = \sum_{m=-h_y}^{h_y} \sum_{n=-h_x}^{h_x} h(m,n) I(i+m, j+n)$$



各画素に重みが入っている



復習

## 非線形フィルタ

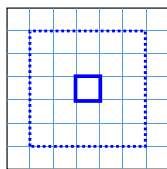
## 準備：平均と分散

実数値の集合  $\{x_i | i = 1, \dots, N\}$  が与えられたとき、

その平均は  $\mu = \frac{1}{N} \sum_{i=1}^N x_i$ 、分散は  $\sigma^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \mu)^2$  で与えられる

- 以下の集合の平均と分散を求めよ  
 $\{3, 0, 3, 5, 4, 3, 5, 1\}$
- 以下の集合AとBどちらが分散が大きい  
A:  $\{3, 4, 3, 4, 3, 2, 2\}$ , B:  $\{3, 5, 3, 5, 3, 1, 1\}$

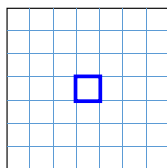
## エッジ保存平滑化フィルタ



入力画像

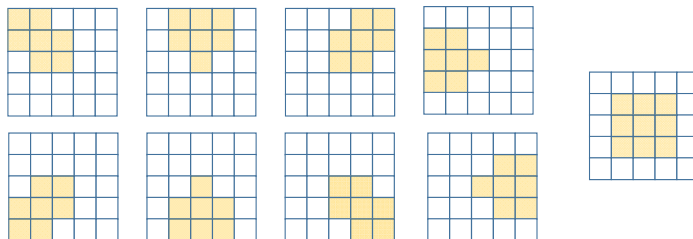
平滑化フィルタでは、画素  $(i, j)$  を計算するため周囲の画素の平均を計算した

1/25	1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25	1/25



出力画像

エッジ保存平滑化フィルタでは、以下9種の領域を考え、一番分散の小さな領域の平均値を、その画素の値とする



入力画像

平滑化フィルタ



エッジ保存平滑化フィルタ



## 中央値フィルタ(Median filter)

- 中央値 (median)とは…

数値の集合の代表値

数値の小さい順に並べ、ちょうど中央に位置する値

入力 : 6, 2, 1, 5, 3, 12, 1000

平均 :  $1/7 \times (6+2+1+5+3+12+1000) = 147$

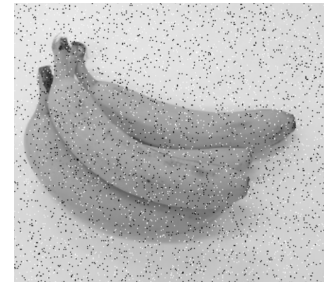
中央値 : 1, 2, 3, 5, 6, 12, 1000 → 5

中央値と平均値は、用途によって使い分ける

→ 年収など、外れ値の影響が大きい対象には中央値を

## 中央値フィルタ(Median filter)

ImageJ  
Process>Filters>Gaussian Blur  
Process>Filters>median



Salt & pepper noise image



Gaussian filter



Median filter

- + 画素( $i,j$ )を中心とする 幅 $h$ の窓内の中央値を新しい画素値とする
- + 外れ値 (スパイクノイズ) を除去出来る
- + 特徴(エッジ)をある程度保存する

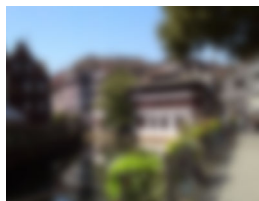
## バイラテラルフィルタ

画像中の領域境界(強いエッジ)をまたがずに平滑化

単純な平滑化

元画像

特徴保存平滑化



(Gaussian filter)

(bilateral filter)

写真は Shin Yoshizawa  
氏により提供されたもの

## バイラテラルフィルタ

ImageJ  
Plug in>Process > Bilateral Filters



Original image



Bi-Lateral Filer  
Spatial radi:3  
Range radi:50



Bi-Lateral Filer  
Spatial radi:5  
Range radi:80

ブラー効果により顔の"あら"が消える  
輪郭が保持されるのでフィルターをかけたことに気づきにくい  
あまり強くかけすぎると不自然な画像になる

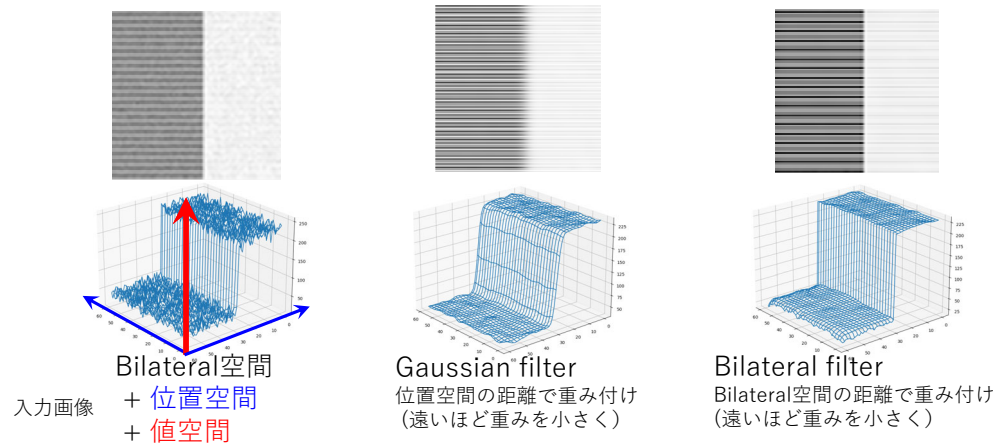


# バイラテラルフィルタ

画像は [CG-Arts協会 デジタル画像処理 図5.37] を参考に井尻が再作成したもの

最も有名な特徴保存フィルタの1つ

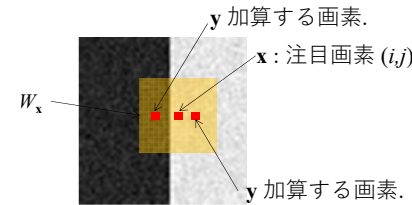
空間的距離だけでなく、画素値の差を利用して重み計算



# バイラテラルフィルタ

$$I_{new}(\mathbf{x}) = \frac{\sum_{\mathbf{y} \in W_{\mathbf{x}}} h(\mathbf{x}, \mathbf{y}) I(\mathbf{y})}{\sum_{\mathbf{y} \in W_{\mathbf{x}}} h(\mathbf{x}, \mathbf{y})}$$

$\mathbf{x}$  : 注目画素位置  
 $\mathbf{y}$  : 局所窓内の画素位置  
 $W_{\mathbf{x}}$  :  $\mathbf{x}$  が中心の局所窓



Gaussian filter :

$$h(\mathbf{x}, \mathbf{y}) = G_s(|\mathbf{x} - \mathbf{y}|)$$

Bilateral filter :

$$h(\mathbf{x}, \mathbf{y}) = \underbrace{G_s(|\mathbf{x} - \mathbf{y}|)}_{\text{Spatial Kernel}} \cdot \underbrace{G_h(|I(\mathbf{x}) - I(\mathbf{y})|)}_{\text{Intensity Kernel}}$$

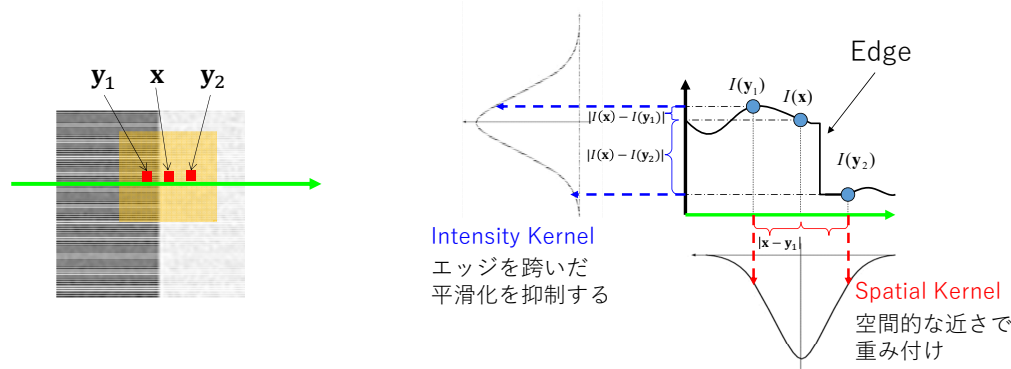
$G_{\sigma}$  は標準偏差  $\sigma$  のガウス関数

※ 『カーネル  $h$ 』 は窓内の画素値に依存するので線形フィルタではない

# バイラテラルフィルタ

注目画素位置  $\mathbf{x} = (i, j)$   
 窓内の画素位置  $\mathbf{y} = (i + m, j + n)$

$$h(\mathbf{x}, \mathbf{y}) = G_s(|\mathbf{x} - \mathbf{y}|) \cdot G_h(|I(\mathbf{x}) - I(\mathbf{y})|)$$



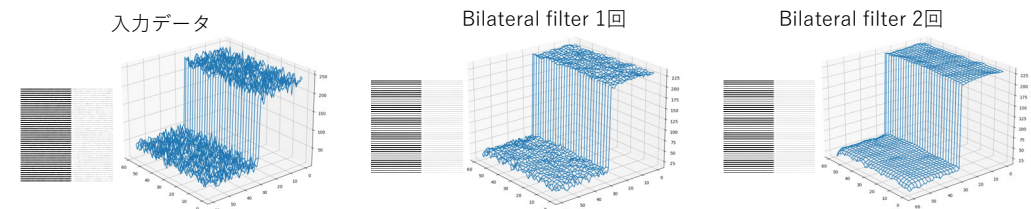
# バイラテラルフィルタ (パラメタ)

$$h(\mathbf{x}, \mathbf{y}) = G_s(|\mathbf{x} - \mathbf{y}|) \cdot G_h(|I(\mathbf{x}) - I(\mathbf{y})|)$$

パラメタ  $h$  : 平滑化したい領域の輝度値の標準偏差の 0.5-2.0 倍程度をよく用いる  
 複数回適用すると良い結果が出やすい

カラー画像はチャンネル毎でなく、以下を用いて同じ重みを利用するとよい

$$|I(\mathbf{x}) - I(\mathbf{y})| = \left| \begin{pmatrix} R(\mathbf{x}) - R(\mathbf{y}) \\ G(\mathbf{x}) - G(\mathbf{y}) \\ B(\mathbf{x}) - B(\mathbf{y}) \end{pmatrix} \right|$$



## まとめ：空間フィルタ（非線形）

- エッジ保存効果のあるフィルタを紹介した
  - エッジ保存平滑化
  - メディアンフィルタ
  - バイラテラルフィルタ
- 線形フィルタと比べ計算量は大きい、特殊な効果が得られる

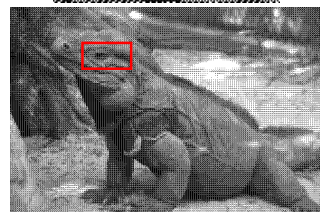
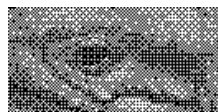


写真は Shin Yoshizawa氏により提供されたもの

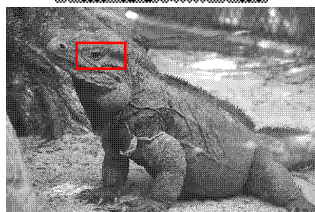
## ハーフトーン処理

### ハーフトーン処理

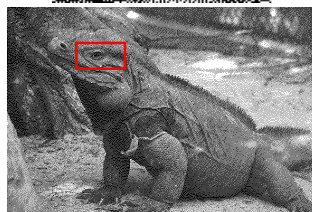
- グレースケール画像を白黒2値画像に変換する
  - 白黒画素の密度により濃淡を表現する
  - 画素が十分細かければ人の目に濃淡として認識される



濃度パターン法



ディザ法

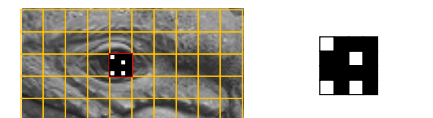
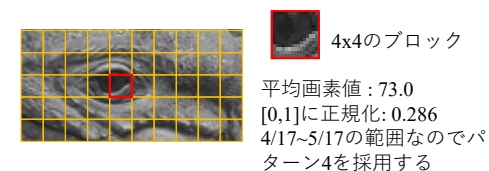
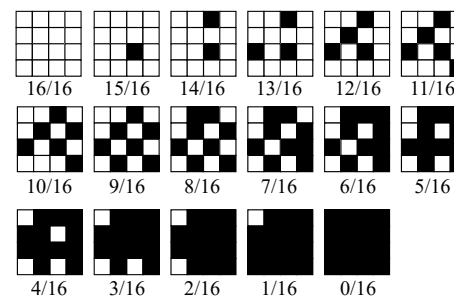


誤差拡散

### 濃度パターン法

1. 元画像を4 x 4のブロックに分割
2. 各ブロックの平均輝度値を計算
3. 各ブロックについて似た平均輝度値をもつパターンを選択し、置き換える

※ブロックのサイズは変更可（今回は4x4）



欠点：繰り返しパターンが目立つ

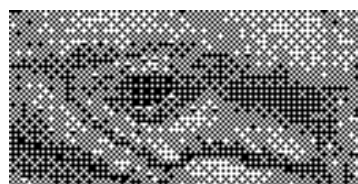
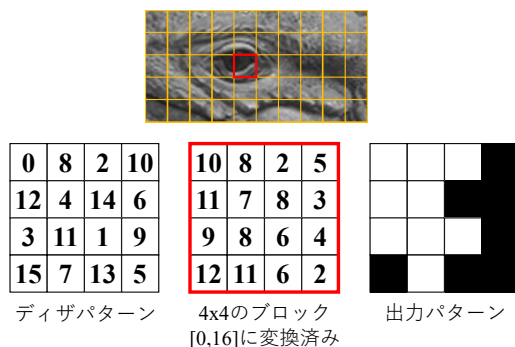


## ディザ法

1. 元画像を4x4のブロックに分割
2. 4x4のディザパターンを用意
3. 各ブロックの画素においてディザパターンと比較

ディザパターンの値以上 → 白  
ディザパターンの値より小さい → 黒

※比較する際、画像の画素値を[0,255]から[0,16]に変更しておく



欠点：繰り返しパターンが目立つ

## 誤差拡散法

- 左上からラスタスキャンし一画素ずつ以下の通り2値化する
- 注目画素の画素値が1のとき

### 1. 二値化処理

- $I > 127 \rightarrow$  注目画素を白に
- $I \leq 127 \rightarrow$  注目画素を黒に

### 2. 誤差拡散

上の二値化で以下の誤差 $e$ が発生した

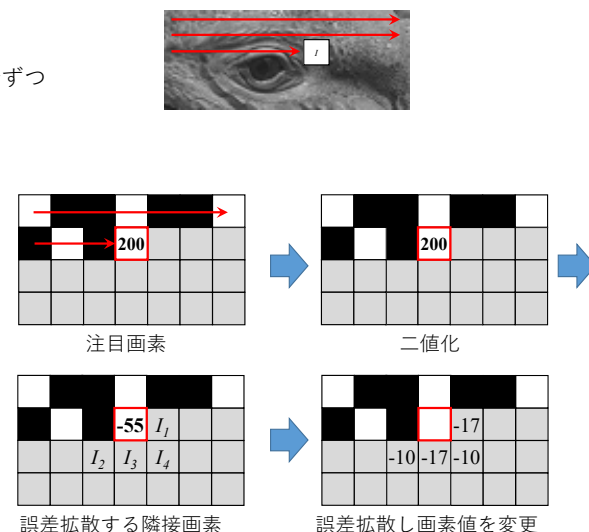
$$I > 127 \rightarrow e = I - 255$$

$$I \leq 127 \rightarrow e = I - 0$$

この誤差を隣接画素  $I_1, I_2, I_3, I_4$  分配 (画素値を変化させる)

$$I_1 \leftarrow I_1 + \frac{5}{16}e, I_2 \leftarrow I_2 + \frac{3}{16}e,$$

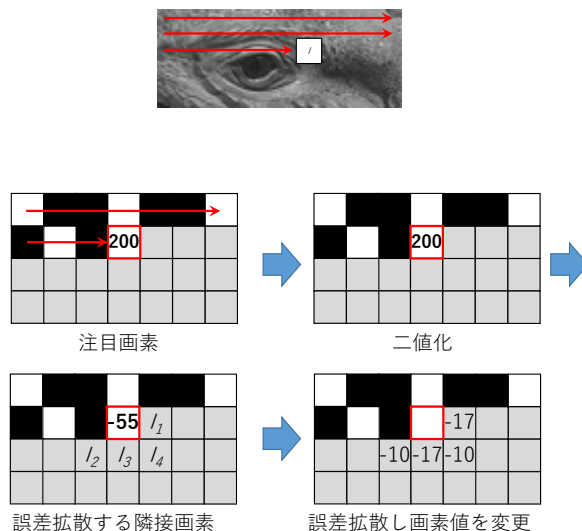
$$I_3 \leftarrow I_3 + \frac{5}{16}e, I_4 \leftarrow I_4 + \frac{3}{16}e$$



## 誤差拡散法

- 実装時の問題：右端や下端では誤差を拡散させる画素がない
- 解決策：右端や下端の計算時、誤差を拡散させる画素がない場合には誤差拡散を行わない

※小テストにて、誤差拡散法の欠点を問う課題を出しています。この右端・左端にておきる問題は、回避可能なのでこれ以外の欠点を回答してください。



## まとめ：ハーフトーン処理

- グレースケール画像を白黒2値画像で表現する手法
- 濃度パターン法**：ブロックの輝度値を利用し濃度パターンで置き換える
- ディザ法**：ディザパターンと画素値を比較し二値化
- 誤差拡散法**：ラスタスキャン順に二値化し、発生した誤差を隣接画素に拡散する

- プログラミング演習で実装します



時間が余ったら、フーリエ級数の話をする