卒業論文

題目

エッジ保存平滑化フィルタの
高性能化に関する研究

指導教員

近藤　利夫

2017年

三重大学 工学部 情報工学科
コンピュータアーキテクチャ研究室

小嶋　武(413817)

内容梗概

近年，平滑化フィルタはダイナミックレンジ圧縮，特徴量抽出での差分画像生成など，コンピュータビジョンやコンピュータグラフィックス等の様々な分野での前処理に利用されている．平滑化フィルタを使う主な目的はノイズ除去であるが，物体認識などの一部分野ではノイズ除去以外の目的でも利用されており，平滑化フィルタの性能向上は大事な課題である．とりわけ，特徴抽出アルゴリズムである SIFT では，ガウシアンフィルタを用い，フィルタの平滑化度合いを変化させながら複数回平滑化することで階層画像の生成を行っている．そのため平滑化の特性が最終的な認識精度に大きく影響する．しかし，このガウシアンフィルタでは画像において重要なエッジもぼかしてしまい，エッジ境界付近の特徴点誤対応で起こる開口問題を引き起こしてしまうという問題がある．

そこで，この平滑化フィルタをエッジ保存フィルタであるバイラテラルフィルタ等に置き換えることで，開口問題の影響を減らし，対応点精度を高めるという研究がされている．しかし，高度なエッジ保存フィルタは実行速度が著しく大きいという問題がある．そのため，本研究室の先行研究ではこの平滑化性能とエッジ保存の両立を図ろうと高度なカーネルと単純で高速なカーネルを切り替え，孤立点ではメディアンフィルタを使用する適応ラテラル型フィルタを提案していた．ガウシアンフィルタやバイラテラルフィルタ等従来のフィルタよりも高度な性能を得られる一方，平滑化がうまくいかないなどの問題もある．

本研究ではこの問題を解決すべく，勾配計算部分，勾配判定部分の改良を行った．勾配計算部分では，勾配の値をノイズの影響が出ないようにガウシアンフィルタ組み込みと勾配の強度を上げるために先鋭化フィルタ組み込みの 2 つを行った．勾配判定部分では，二値化処理に用いられる判別分析法を用いて判定値の決定を行った．これにより，先行研究のフィルタの速度性能を落とすことなく，エッジ保存平滑化性能の性能評価に用いた SIFT での対応点検出精度では 99 %に向上した．また，新たなフィルタアルゴリズムとして分散値によるフィルタ切り替えも提案する．こちらでは実行時間を約 60 %削減でき，対応点検出精度を向上することができ，単純なバイラテラルフィルタとガウシアンフィルタの分散値による切り替えでも性能を向上できることを明らかにした．

Abstract

In recent years, smoothing filters have been used for preprocessing in various fields, such as dynamic range compression, difference image generation with feature quantity extraction, computer-vision and computer-graphics. The main aim of using smoothing filters is to achieve noise rejection. However, it is important to improve the performance of smoothing filters because smoothing filters are also used for purpose other than noise rejection in some fields such as object recognition. In particular, the feature extraction algorithm - SIFT use the Gaussian filter to generate a hierarchical image by smoothing the filter several times while the degree of smoothing filter is varied. Therefore, the smoothing characteristic has a big influence on the final recognition precision. However, because Gaussian filter blur the important edge in the image, the aperture problem will be caused by the erroneous correspondent of characteristic point in proximity to the boundaries of edge.

Previous studies proposed method that replace the smoothing filter with a Bilateral filter(edge preserving filter) to reduce the influence by aperture problem and improve the precision of the corresponding point. However, the execution time for the advanced edge preserving filter is large. Therefore, to achieve both performance of smoothing and edge preserving, previous study of our laboratory proposed an Adaptive-lateral filter that can switch between the advanced edge-preserving kernel and the simple high-speed non-edge-preserving kernel. The isolated point use the median filter. Compared with the conventional filters such as Gaussian filter or Bilateral filter, the proposed filter can provide high performance, however, the smoothing problem is still not be eliminated.

In this paper, to solve this problem, we propose method that improve the performance by incorporating the filter in gradient calculation part. We also propose a new filter algorithm that switch the filter by variance value.

目 次

図 目 次

表 目 次

1　まえがき

ハードウェアの性能向上に伴い，コンピュータビジョンやコンピュータグラフィックスをはじめとする種々の分野で，画像をベースとする研究が盛んに行われている [1][2]．その画像を扱うアルゴリズムの多くの前処理に平滑化を行っており，画像処理など一部分野において，その平滑化フィルタの性能がのちの処理に大きく影響することから，平滑化フィルタの性能の重要度が高い．しかし，従来のガウシアンフィルタに代表される平滑化フィルタは平滑化の際に空間距離などの画素値に依存しない情報を用いるので，ノイズだけでなく画像において重要なエッジまでぼかしてしまう問題がある．そこでエッジを保ちながら平滑化できるバイラテラルフィルタ [3] に代表されるエッジ保存平滑化フィルタが広く利用されており，そのバイラテラルフィルタを改善したトリラテラルフィルタ [4] も提案されている．ところが，その改善には，処理時間の大幅な増加や平坦領域での平滑化性能の劣化など副作用を免れない．

これに対し，当研究室では，画素毎に求めた勾配情報を利用して領域検出をし，その結果に基づきエッジ部とそれ以外の領域で，これらフィルタを使い分けることによって，高度なエッジ保存，強力な平滑化能力，高速な実行を同時に満たす新しいエッジ保存平滑化フィルタが提案され

た [5]．だが，この先行研究のフィルタにも平滑化がうまくいかないなどの問題が生じる場合がある．

そこで本研究では，この欠点を解決することで更なる高性能なエッジ保存平滑化フィルタを実現する手法とともに，新たなフィルタ切り替えアルゴリズムとして，画素値の分散値を利用したフィルタの切り替え手法を提案する．そして，先行研究と同様に，既存のエッジ保存フィルタと共に SIFT に組み込み，特徴点対応精度を比較することでエッジ保存性能の評価・検証を行い，高いエッジ保存性能を持ちながら既存の高性能フィルタよりも高速であることを示す．

2　従来のフィルタとその平滑化性能

この章では，代表的な平滑化フィルタであるガウシアンフィルタ，エッジ保存フィルタで様々な研究者によって高速化の研究が行われているバイラテラルフィルタ，そして高度なエッジ保存平滑化が可能なトリラテラルフィルタについて述べる．

2.1　ガウシアンフィルタ (Gaussian Filter)

代表的な平滑化フィルタであるガウシアンフィルタは，座標 (i, j) における処理前の画像データを $f(i, j)$，フィルタ処理の出力を $g(i, j)$，ウィンドウサイズを w，空間の標準偏差を σ とした場合，

$$g(i,j) = \frac{\sum_{n=-w}^{w} \sum_{m=-w}^{w} f(i+m, j+n) \exp(-\frac{m^2+n^2}{2\sigma^2})}{\sum_{n=-w}^{w} \sum_{m=-w}^{w} \exp(-\frac{m^2+n^2}{2\sigma^2})}$$

と表されるようにカーネルの重みに画像に依存しない空間的距離を利用しているため，ノイズを除去する際にエッジも同時に平滑化してしまう．

2.2　バイラテラルフィルタ (Bilateral Filter)

ガウシアンフィルタではエッジも同時に平滑化してしまう問題があった．この欠点を改善すべく 1998 年に Tomasi ら [3] によって提案された強いエッジを保存する非線形の平滑化フィルタがバイラテラルフィルタで

ある．バイラテラルフィルタは，空間の標準偏差を σ_1，信号差の標準偏差を σ_2 のとき，以下の式

$$g(i,j)=\frac{\sum_{n=-w}^{w}\sum_{m=-w}^{w} f(i+m,j+n)\exp(-\frac{m^2+n^2}{2\sigma_1^2})\exp(-\frac{(f(i,j)-f(i+m,j+n))^2}{2\sigma_2^2})}{\sum_{n=-w}^{w}\sum_{m=-w}^{w} \exp(-\frac{m^2+n^2}{2\sigma_1^2})\exp(-\frac{(f(i,j)-f(i+m,j+n))^2}{2\sigma_2^2})}$$

で表されるようにガウシアンフィルタで使用されている画素間の空間的距離に加えて画素値の差である信号差も使用している．これによりエッジ保存が可能になっている．しかし，平坦領域での平滑化性能がガウシアンフィルタに劣るという欠点がある．

2.3 トリラテラルフィルタ (Trilateral Filter)

トリラテラルフィルタは2003年にChoudhuryら [4] によってダイナミックレンジ圧縮やメッシュノイズ除去を主な目的として提案されたフィルタであり，強力な平滑化能力とエッジ保存性能を備えている．バイラテラルフィルタとの大きな違いはカーネルの重み計算に画素間の信号差の代わりに勾配情報を使っている点と多くのパラメーターを動的に決めている点が挙げられる．勾配情報は単純な勾配差に対してバイラテラルフィルタを掛けたものを使用しており，これによりノイズの影響を減らしていると考えられる．また，トリラテラルフィルタは，内部的には多くのパラメーターを必要としているものの，ユーザーによる画像に合わせたパ

ラメーター最適化を不要とするため，１つのパラメーターσを除き，他のすべてのパラメーターを動的に計算するようにしている．この内，特筆すべきはウィンドウサイズの動的計算である．通常ウィンドウサイズはユーザーが任意に指定するのに対し，トリラテラルフィルタでは動的に，更に画像毎ではなく画素毎にフィルタサイズを決定するようにしている．この理由はエッジの影響を受けずに強く平滑化を行い性質の異なる領域に達しない最大限の範囲で平滑化できるようにするためと考えられる．しかし，この可変のウィンドウサイズは最大限の範囲での高品質な平滑化を可能にする一方，処理時間の大幅な増加を招いてしまっている．

3　適応ラテラルフィルタとその問題点

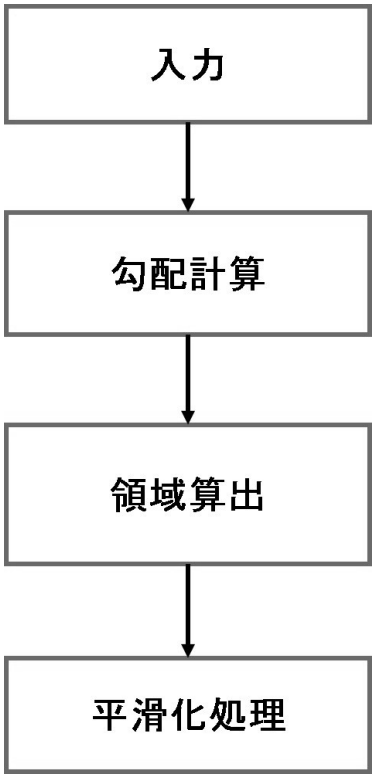

図 3.1: 適応ラテラルフィルタのフローチャート

先行研究では適応的に画素毎に複数のカーネルを切り替えていることから適応ラテラルフィルタ (Adaptive-lateral filter) と呼ばれるフィルタが提案された [5]．領域ごとにフィルタを切り替えることによって，高度なエッジ保存，強力な平滑化能力，高速な実行の両立を図っている．重要なエッジ境界付近をエッジ保存可能な高度なフィルタを用いて処理しつつも，高速化を図るために平坦領域を LUT（Look-Up-Table) による高

速実行が可能なガウスカーネルを用いているところが主要な特徴である．これにより，バイラテラルフィルタをはじめとする従来のエッジ保存フィルタよりも強力な平滑化が可能となっている．主な処理は，図 3.1 に示すように大きく分けて，勾配計算，領域検出，平滑化処理の 3 行程に分かれる．

3.1 勾配計算

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

水平方向

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

垂直方向

図 3.2: 各方向のソーベルフィルタカーネル

勾配計算にはソーベルフィルタを用いて縦・横方向の勾配を算出している．図 3.2 に示すカーネルを使い，水平方向と垂直方向の勾配を計算したのちに，その値を二乗して加算したものの平方根を勾配の値としている．ここで求めた勾配は次の領域拡張法で使用している．

3.2 領域検出

領域検出では領域拡張法と呼ばれる画像を同領域毎に分割するために用いられる手法を用いて平滑化の際の画素毎のウィンドウサイズ計算に用いている．同領域かどうかは，勾配計算で算出した勾配ベクトルとその勾配ベクトルから求めた画像全体の勾配の平均値に勾配調整用変数 Rp を掛け合わせたものを使用している．徐々に領域を広げていき，その求めた値以上の勾配差が出た地点までを同領域とし，またそれまで広げた距離をウィンドウサイズとしている．

3.3 平滑化処理

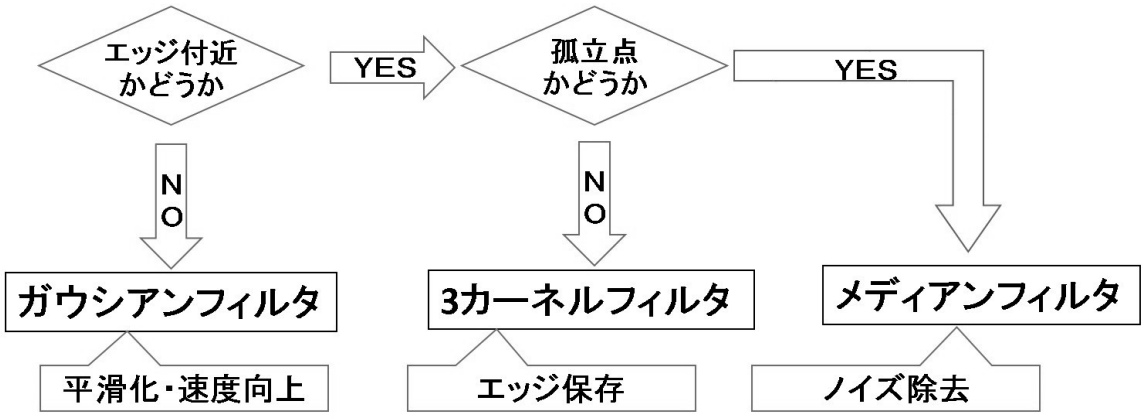

図 3.3: 平滑化処理

平滑化処理は図 3.3 に示すように，3 種類のフィルタを切り替えて行っ

ており，高速化を目的とした通常のガウシアンフィルタ，孤立点除去を目的としたメディアンフィルタ，エッジ付近のエッジ保存平滑化を目的とした 3 カーネルフィルタを使用している．このうち 3 つめのフィルタは，空間の標準偏差を σ_1，信号差の標準偏差を σ_2，勾配差の標準偏差を σ_3，座標 i, j における水平方向の勾配成分を $x(i, j)$，垂直方向の勾配成分を $y(i, j)$ とした場合，以下の三つの重みを利用する．

空間的距離：

$$A = \exp(-\frac{m^2 + n^2}{2\sigma_1^2}),$$

信号差：

$$B = \exp(-\frac{(f(i,j) - f(i+m, j+n))^2}{2\sigma_2^2}),$$

勾配差：

$$C = \exp(-\frac{\sqrt{D+E}}{2\sigma_3^2}),$$

$$D = (x(i,j) - x(i+m, j+n))^2,$$

$$E = (y(i,j) - y(i+m, j+n))^2.$$

これら 3 つの重みを掛け合わせたエッジ付近のエッジ保存平滑化を目的

とした3 カーネルのフィルタは以下の式で表される：

$$g(i,j) = \frac{\sum_{n=-w}^{w}\sum_{m=-w}^{w} f(i+m,j+n)\cdot A\cdot B\cdot C}{\sum_{n=-w}^{w}\sum_{m=-w}^{w} \cdot A\cdot B\cdot C}.$$

3 カーネルのフィルタは，勾配情報を使うことで従来バイラテラルフィルタよりも正確なエッジ保存を可能とし，事前に計算した勾配情報を用いているため計算量の増加を最小限に抑えられている．また，高速化のために空間的距離を用いるガウスカーネルを事前に計算しておくことで処理時間を低減している．また，エッジ保存可能なカーネルはノイズなどの孤立点の除去には向かないため，別途メディアンフィルタで孤立点除去を行っている．

先行研究フィルタの出力画像を 3.4 に示す．また，ガウシアンフィルタ，およびバイラテラルフィルタの出力画像をそれぞれ 3.5，3.6 に示す．ここからわかるように適応ラテラルフィルタは従来のフィルタよりもエッジ保存かつ平滑化ができていることがわかる．

3.4 適応ラテラルフィルタの実行結果

評価方法は後ほど説明するが，先行研究と同様に実行時間と SIFT に組み込んだ際の対応点検出精度を比較する．対応点検出精度にはテスト画像，およびテスト画像の一部を切り出し 2 倍拡大させた後，加工を施した

ペアを用いる．加工方法はノイズ付加（_n），コントラストの変化（_c），jpeg 圧縮（ダウンサンプリング）（_d）でテスト画像は２種類（Set1，Set2）用意した．

使用した画像は図 3.7，3.8 に示す．

実行時間を表 3.1 に示す．

対応点検出精度を表 3.2 に示す．

3.5　適応ラテラルフィルタの問題点

図 3.9，3.10 はそれぞれ適応ラテラルフィルタ，トリラテラルフィルタの出力画像の一部を拡大したものである．ここからわかるように条件によっては平滑化がうまくいかない問題がある．これは勾配算出がうまくできていない，もしくは，領域検出の際の勾配判定の調整用変数 Rp がすべて同じ値を使っていることが問題だと考えられる．

図 3.4: 適応ラテラルフィルタの出力画像

図 3.5: ガウシアンフィルタの出力画像

図 3.6: バイラテラルフィルタの出力画像

図 3.7: set1 の評価画像

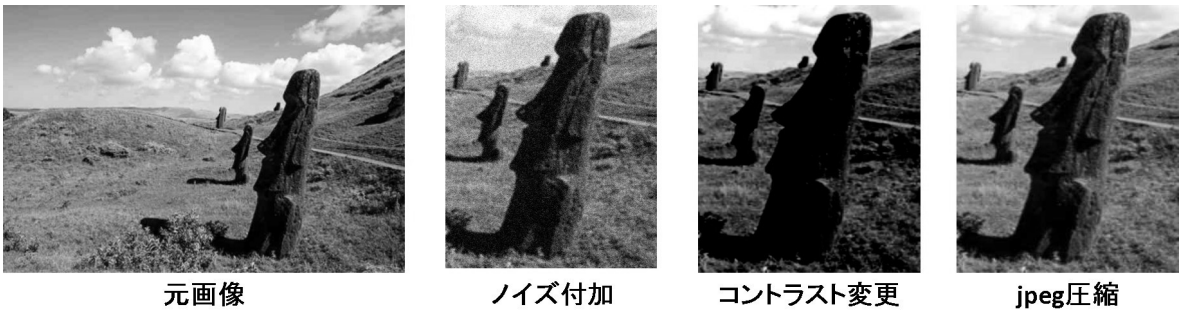

図 3.8: set2 の評価画像

表 3.1: フィルタ毎の実行時間 (秒)

評価画像 解像度	ガウシアン フィルタ	バイラテラル フィルタ	トリラテラル フィルタ	適応ラテラル フィルタ
202 × 139	0.04	0.09	5.37	0.06
300 × 400	0.18	0.37	424.14	0.22
900 × 600	0.83	1.56	5921.53	1.15

※ガウシアンフィルタ，バイラテラルフィルタでは $\sigma = 1.6$，ウィンドウサイズ ＝ 5，トリラテラルフィルタでは $\sigma = 3$，適応ラテラルフィルタではそれぞれ，$\sigma_1 = 3, \sigma_2 = 5, \sigma_3 = 21$ で実験を行った．

表 3.2: フィルタ毎の対応点検出精度

	Set1_n	Set1_c	Set1_d	Set2_n	Set2_c	Set2_d	Average
GF	12/25	11/15	15/19	61/66	55/61	108/117	0.792
BF	15/17	15/18	16/17	138/142	116/119	292/293	0.933
TF	4/4	25/25	40/46	117/119	14/14	34/38	0.958
ALF	25/25	18/18	30/32	41/41	150/158	258/260	0.980

※基本的なパラメーターはそれぞれ SIFT アルゴリズムで用いられるデフォルトのものを使用している．
バイラテラルフィルタ，トリラテラルフィルタでは $\sigma = 3$，適応ラテラルフィルタでは $\sigma_1 = 3$,$\sigma_2 = 5 * \sigma_1$,$\sigma_3 = 30 * \sigma_1$ で実験を行った．
GF：ガウシアンフィルタ，BF：バイラテラルフィルタ
TF:トリラテラルフィルタ，ALF：適応ラテラルフィルタ

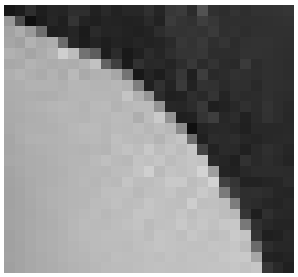

図 3.9: 適応ラテラルフィルタの画像の一部

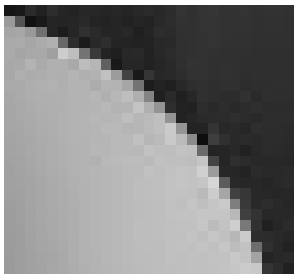

図 3.10: トリラテラルフィルタの出力画像の一部

4　適応ラテラルフィルタの安定化

適応ラテラルフィルタでは問題点はあるものの，先ほどの表3.1，3.2の性能結果からわかるように実行時間では従来のエッジ保存フィルタで代表されるバイラテラルフィルタよりも高速な結果が得られ，対応点検出精度の平均では性能の高いトリラテラルフィルタよりも更に高い精度が得られた．先ほどの問題点を解決することにより，この高い性能を安定的に得られるようになる．そこで，本研究では先ほど述べた問題部分にあたる勾配計算部分に既存のガウシアンフィルタ，先鋭化フィルタを組み込む手法と勾配判定の変数 R の決定を判別分析法によって動的に判定値を決定する手法を提案する．

4.1　勾配計算精度の改善

適応ラテラルフィルタでは図3.1の勾配計算部分で，ソーベルフィルタを水平方向，垂直方向にそれぞれかけたのち，その値の二乗和を平方根した値を用いていた．そのソーベルフィルタをかける前に別のフィルタをかけることで勾配の計算をより正確に行う．

4.1.1 ガウシアンフィルタの組み込み

勾配検出では画像中のノイズに対して敏感なため，ソーベルフィルタだけでは不十分である．そこでエッジ検出においてもっとも有効と言われているキャニー法 [6] の勾配抽出方法に基づき，ソーベルフィルタをかける前にガウシアンフィルタをかけた後に, 勾配を算出する手法を提案する．これによりノイズをあらかじめガウシアンフィルタによって平滑化することでノイズの影響を防ぐ．実行時間をなるべく増加させないよう，あらかじめフィルタのカーネルを準備しておき，勾配を検出する [7].

ガウシアンフィルタのフィルタカーネルを図 4.11 に示す.

2/159	4/159	5/159	4/159	2/159
4/159	9/159	12/159	9/159	4/159
5/159	12/159	15/159	12/159	5/159
4/159	9/159	12/159	9/159	4/159
2/159	4/159	5/159	4/159	2/159

図 4.11: ガウシアンフィルタのフィルタカーネル

4.1.2 先鋭化フィルタの組み込み

勾配の検出には画素の値に大きく依存している．また，エッジにはとても敏感である．そこで先鋭化フィルタをかけることによってエッジを強調させることで，勾配の強度を上げることを提案する．ガウシアンフィルタの組み込みと同様に先鋭化フィルタもあらかじめフィルタカーネルを準備しておくことで実行時間の増加を防いでいる．

先鋭化フィルタのフィルタカーネルを図 4.12 に示す．

-1	-1	-1
-1	16	-1
-1	-1	-1

図 4.12: 先鋭化フィルタのフィルタカーネル

4.2 勾配判定変数の動的判定への対応

適応ラテラルフィルタでは全勾配の平均に勾配調整変数 $Rp = 1.4$ を掛け合わせたものを勾配判定の変数 R に使用していた．本研究では勾配判

定に用いる変数 R を動的行えるよう判別分析法を用いる手法を提案する．

4.2.1 判別分析法

判別分析法（discriminant analysis method）[8] は大津の二値化とも言われ，分離度という値が最大となる値を求めて，自動的に二値化する手法である．分離度はクラス間分散とクラス内分散との比で求められる．元々この判別分析法は画像の二値化に用いられるものである．閾値 t で二値化したとき，閾値よりも輝度値が小さい側（黒側）の画素数を w_1，平均を m_1，分散を σ_1，輝度値が大きい側（白側）の画素数を w_2，平均を m_2，分散を σ_2，画素全体の画素数を w_t，平均を m_t，分散を σ_t としたときクラス内分散 σ_w^2 は

$$\sigma_w^2 = \frac{w_1\sigma_1^2 + w_2\sigma_2^2}{w_1 + w_2},$$

クラス間分散 σ_b^2 は

$$\begin{aligned}\sigma_b^2 &= \frac{w_1(m_1 - m_t)^2 + w_2(m_2 - m_t)^2}{w_1 + w_2} \\ &= \frac{w_1 w_2(m_1 - m_2)^2}{(w_1 + w_2)^2},\end{aligned}$$

とあらわすことができる．ここで，全分散は σ_t は

$$\sigma_t^2 = \sigma_b^2 + \sigma_w^2,$$

とあらわすことができることから求めるクラス間分散とクラス内分散との比である分離度は

$$\frac{\sigma_b^2}{\sigma_w^2} = \frac{\sigma_b^2}{\sigma_t^2 - \sigma_b^2},$$

となり，この分離度が最大となる閾値 t を求めればよい．

本研究ではこの判別分析法を勾配の値を用いて行った．具体的には，本来輝度を使う部分を勾配の値を用いて行った．ここで勾配の値はガウシアンフィルタをかけ，ノイズをぼかしたのちにソーベルフィルタをかけて求めた勾配の値を用いる．

5 分散値によるフィルタ切り替えの提案

適応ラテラルフィルタでは勾配の値を用いて領域算出を行い，フィルタの切り替えを行っていた．フィルタの切り替えによって実行速度が大幅に削減できることがわかっている．そこでフィルタの切り替えを勾配ではなく画素の分散値を用いた手法を提案する．

5.1 フィルタアルゴリズム

このフィルタは分散の計算・平滑化処理の 2 行程にわかれる．まず，入力画像から注目画素の近傍領域から分散値を求める．この分散の算出か

らフィルタの切り替えを行う．使用するフィルタは平滑化に最適なガウシアンフィルタとエッジ保存フィルタであるバイラテラルフィルタを用いる．分散値が大きいとき，注目画素付近がエッジを含んでいる可能性があり，バイラテラルフィルタを用いる．分散値が小さいとき，注目画素付近は輝度のばらつきが小さいため，平坦領域の可能性があり，ガウシアンフィルタを用いる．切り替えの閾値は今回は全分散値の平均の値を採用している．

5.2 出力画像

出力画像を図 5.13 に示す．また，バイラテラルフィルタ適応部分（黒線部分）を図 5.14 に示す．

図 5.13 と図 3.5 を比較すると画像処理において重要なエッジ部をガウシアンフィルタよりも残すことができている．これは図 5.14 からわかるようにエッジ部ではバイラテラルフィルタを適応しているからである．また，ほとんどの領域でガウシアンフィルタを用いているため，高速化が見込まれる．

図 5.13: 分散値によるフィルタの出力画像

図 5.14: バイラテラルフィルタ適応部分

6　性能評価

6.1　評価方法

本研究では，異なるいくつかの画像にフィルタをかけ，フィルタの処理時間を用いて速度評価を行う．また，フィルタ性能に関しては，先行研究と同様にSIFT（Scale-Invariant Feature Transform)[9] を利用して評価する．SIFTは画像認識に必要なキーポイントを抽出し，それに対応付けるアルゴリズムであり，内部で用いる平滑化フィルタのエッジ保存性能が，抽出した特徴点同士の対応点検出率に大きく影響する．この性質に基づき，対応点検出率をフィルタ性能の客観的指標として利用する．

6.1.1　物体認識アルゴリズム SIFT(Scale-Invariant Feature Transform) について

SIFTはD. G. Loweによって提案された局所特徴量検出アルゴリズムで，検出した特徴量を用いて画像間でマッチングをとることができる．SIFTの処理は主にキーポイント検出と特徴量記述の二段階からなっており，各処理は以下の流れになっている：

1. キーポイント検出

    - キーポイントの検出

- キーポイントのローカライズ

2. 特徴量記述

- オリエンテーションの算出
- 特徴量の記述

キーポイント検出ではキーポイントを抽出し不要なキーポイントの削除を行い，特徴量記述ではオリエンテーションを算出し，それを用いて特徴量の記述を行っている．キーポイントの検出では DoG(Difference-of-Gaussian) 画像と呼ばれる差分画像を使っている．DoG 画像は入力画像に対して少しずつ平滑化の度合いを変化させながらガウシアンフィルタを掛けた画像間で差分を取り生成する．この DoG 画像の中で出力が大きい点をキーポイントとしているため，互いに識別が困難なエッジ上の点も，エッジが保存されない限りキーポイントとして検出されてしまうために，局所特徴量を扱うアルゴリズム共通の課題である開口問題によりマッチングの際に誤対応を起こしやすい．そこで SIFT アルゴリズムの改良法の一つとして，エッジをぼかしてしまうガウシアンフィルタの代わりにエッジ保存フィルタを使う方法が提案されている [10][11]．エッジ保存フィルタを用いることで差分画像の DoG を生成する際にエッジ付近で差

分が生じないようにして，誤対応の主因となっているエッジ付近のキーポイントが検出されないようにする方法である．

6.2　評価結果（適応ラテラルフィルタの安定化）

適応ラテラルフィルタの勾配計算および勾配判定の改善について，実行時間を表 6.3 に示す．また対応点検出精度を表 6.4 に示す．

実行時間に使用した画像でのフィルタの使用数をそれぞれ図 6.15,6.16,6.17 に示す．

表 6.3: 4 章の改良による実行時間 (秒)

評価画像 解像度	適応ラテラル フィルタ	ガウシアン フィルタ 組み込み	先鋭化 フィルタ 組み込み	判別 分析法
202 × 139	0.06	0.05	0.06	0.05
300 × 400	0.22	0.21	0.21	0.19
900 × 600	1.15	1.13	1.12	1.10

※パラメーター σ は表 3.1 と同様のものを使用している．ただし，判別分析法のみ $\sigma = 2$ を使用している．

表 6.4: 4 章の改良による対応点検出精度

	Set1_n	Set1_c	Set1_d	Set2_n	Set2_c	Set2_d	Average
ALF	25/25	18/18	30/32	41/41	150/158	258/260	0.980
GF 組み込み	19/19	20/21	13/13	219/221	165/165	310/310	0.990
SF 組み込み	28/28	39/39	41/43	122/123	174/175	292/292	0.990
判別分析法	26/33	33/34	16/34	92/95	77/84	97/103	0.842

※パラメーター σ は表 3.2 と同様のものを使用している．ただし，判別分析法のみ $\sigma = 2$ を使用している．画像セットには先と同様に図 3.7，3.8 を使用した．
ALF：適応ラテラルフィルタ，GF：ガウシアンフィルタ，SF：先鋭化フィルタ

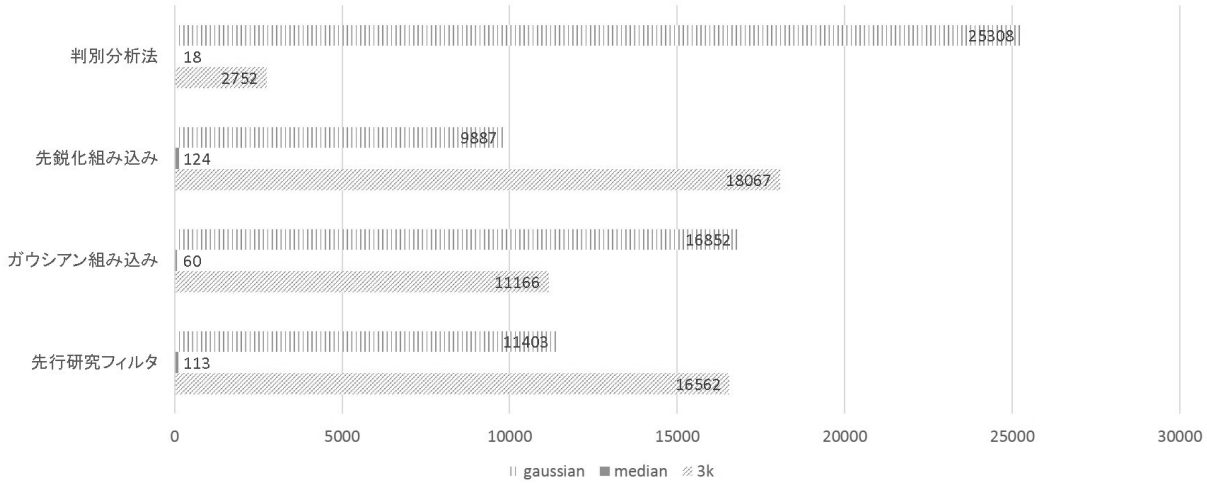

図 6.15: フィルタ使用数　200 × 139

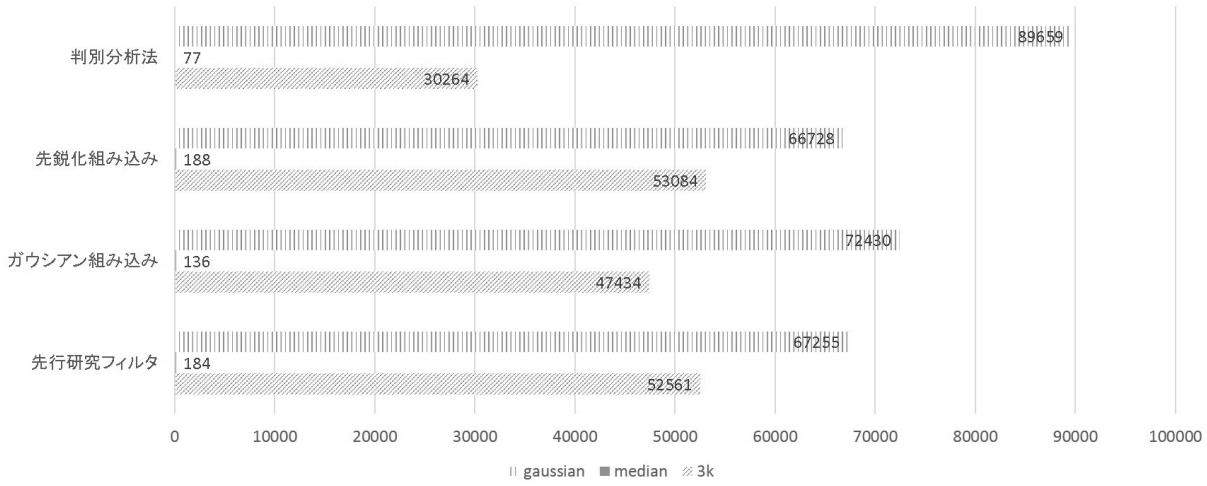

図 6.16: フィルタ使用数　300 × 400

6.3　評価結果（分散値によるフィルタ切り替え）

分散値によるフィルタ切り替え手法の評価は先と同様に行う．実行時間を表 6.5 に示す．また，対応点検出精度を表 6.6 に示す．

実行時間に使用した画像でのフィルタの使用数を図 6.18 に示す．

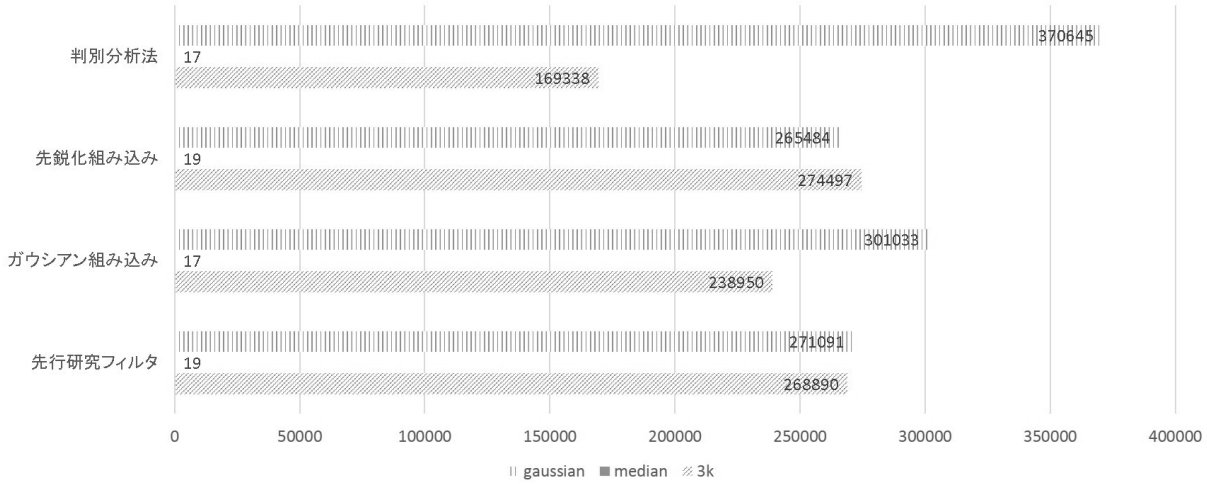

図 6.17: フィルタ使用数　900 × 600

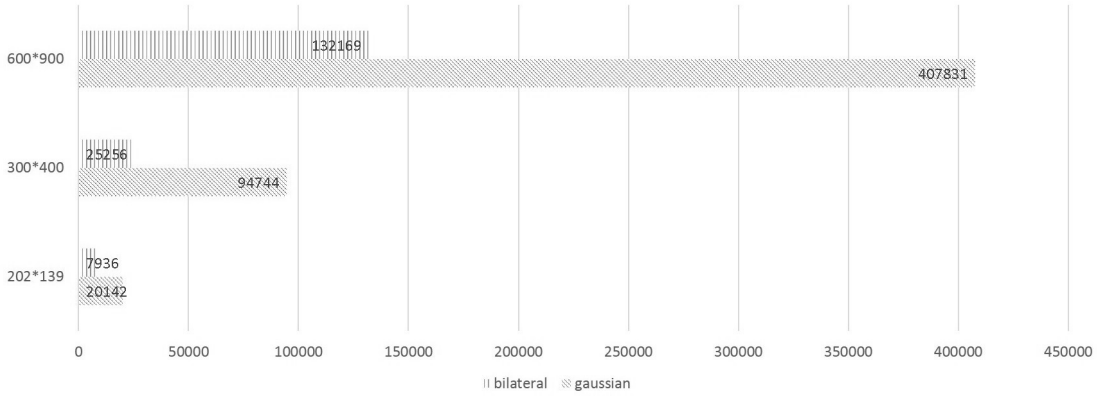

図 6.18: 分散値フィルタのフィルタ使用数

6.4　考察

表 6.3 の結果から，適応ラテラルフィルタの安定化では実行時間に大きな変化は見られなかったがガウシアンフィルタ組み込みと判別分析法で

表 6.5: 5 章の分散値切り替え手法の実行時間 (秒)

評価画像 解像度	適応ラテラル フィルタ	分散値 切り替え
202 × 139	0.06	0.02
300 × 400	0.22	0.08
900 × 600	1.15	0.42

※パラメーター σ は表 3.2 と同様のものを使用している．

表 6.6: 5 章の分散値切り替え手法の対応点検出精度

	Set1_n	Set1_c	Set1_d	Set2_n	Set2_c	Set2_d	Average
ALF	25/25	18/18	30/32	41/41	150/158	258/260	0.980
分散値切り替え	15/16	16/16	28/28	141/143	141/141	292/292	0.987

※パラメーター σ は表 3.2 と同様のものを使用している．
ALF：適応ラテラルフィルタ

少し高速化された．だが，これは図 6.15,6.16,6.17 からわかるようにガウシアンフィルタの使用頻度が多いのが要因と考えられる．表 6.4 の結果から，ガウシアンフィルタ組み込みと先鋭化フィルタ組み込みでは，従来のフィルタよりも性能の高い適応ラテラルフィルタよりも更に高い精度が得られた．さらに，ガウシアンフィルタの組み込みでは実行時間も高速化され，かつ対応点検出精度が上がったことから，適応ラテラルフィルタよりも効率的にガウシアンフィルタの使用頻度を上げることができている．反対に判別分析法では，対応点検出精度が大きく低下してしまった．これは，フィルタの切り替えがうまくできておらず，ガウシアンフィルタが特徴点抽出において重要なエッジもぼかしてしまい，開口問題を引

き起こしていることが原因を考えられる．このため，分離度を用いた勾配判定はこのフィルタには不適切であることが明らかになった．しかし，Set1 の加工画像では 126 × 128 というサイズの小さい画像を用いており，ただ単純に勾配の平均の値を調整した固定値の判定の ALF より対応点の数が増加している．先鋭化フィルタの組み込みでは精度が高くなった上に，組み込む前よりも対応点の数が増えた．これはエッジを強調したのちに勾配計算を行ったことにより，勾配の値が大きくばらつき，フィルタの切り替えが顕著になったことが原因と考えられる．

次に表 6.5 の結果から，分散値による切り替えでは適応ラテラルフィルタよりもかなり高速化された．これは，単純なガウシアンフィルタとバイラテラルフィルタの切り替えのみで行っていることが大きな要因だと考えられる．また図 6.18 からもわかるように大部分が高速なガウシアンフィルタを用いていることも要因である．そして，オリジナルのガウシアンフィルタよりも高速化されたのは LUT を適応したからである．表 6.6 から対応点検出精度においても適応ラテラルフィルタよりも高い結果が得られた．これは図 6.19 からわかるように，先行研究でうまく平滑化できていない部分もうまく平滑化できている点でフィルタを効率よく切り替えられていることが要因だと思われる．以上から分散値でのフィル

タ切り替え，そしてガウシアンフィルタとバイラテラルフィルタの切り替えだけでも SIFT の対応点検出精度では高い結果が得られることがわかった．

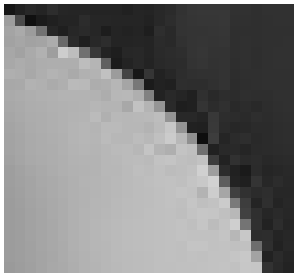

図 6.19: 先行研究での平滑化の失敗部分

7　あとがき

7.1　まとめ

本研究では従来のフィルタよりも高性能な先行研究の適応ラテラルフィルタの更なる安定化とフィルタ切り替えに分散値を取り入れた手法を提案した．その結果，安定化のために勾配計算にガウシアンフィルタを用いたのちにソーベルフィルタをかける手法により，効率的にフィルタの切り替えが行えることが明らかになった．分散値による新たなフィルタ切り替える手法では，高速かつ高性能の結果が得られると共に，単純なガウシアンフィルタとバイラテラルフィルタの切り替えのみでも高い性能が出ることが明らかになった．また，判別分析法の勾配判定では，精度悪化が起こってしまったが，ALF の固定値による判定よりサイズの小さい画像でも対応点の数が安定的に得られる．信頼性を得るためにはガウシアンフィルタなどのフィルタの切り替えがしっかり行えていないと高速化されても対応点抽出精度が悪くなるので，そのフィルタ切り替えの見極めを動的かつ適切に求めるプログラムの考案が重要である．

7.2　今後の課題

図 7.20 は適応ラテラルフィルタの 3 カーネルフィルタ適応部分を黒くしたものである．このバイラテラルフィルタとガウシアンフィルタの組み合わせでは平滑化やエッジ保存がうまくいくにも関わらず，適応ラテラルフィルタではうまくいかないことが 3 カーネルフィルタを用いてる部分に生じうる．適応ラテラルフィルタの性能を更によくするためにはこの 3 カーネルフィルタの見直しが必要である．また，今回は勾配改善と勾配判定の改善を行ったが，勾配改善の先鋭化フィルタ組み込みを行ったのちに判別分析法を適応させた場合，勾配の強度が上がった状態であるため，精度改善が見込まれる可能性がある．そして最後に提案した分散値によるフィルタ切り替えの手法ではノイズが密集している箇所などではうまく平滑化できない可能性があるため，フィルタ切り替えの判定や使用するフィルタを見直すことで更なる安定化が見込まれる．そして今回提案したフィルタを SIFT 以外の平滑化フィルタが利用されているアルゴリズムに組み込んで評価し，様々なアルゴリズムで有用であることを示していく必要もある．

図 7.20: 適応ラテラルフィルタの 3 カーネルフィルタ部分

謝辞

本研究を遂行するにあたり，日頃から御指導，御助言を頂きました近藤利夫教授，佐々木敬泰助教，深澤祐樹研究員に感謝いたします．また，研究に協力していただいた計算機アーキテクチャ研究室の方々に感謝の意を表します.

参考文献

[1] 金澤靖，金谷健一，“コンピュータビジョンのための画像の特徴点の抽出 ”,
＜ http://www.suri.cs.okayama-u.ac.jp/ kanatani/papers/harris.pdf ＞,（2017 年 2 月 6 日アクセス）

[2] “コンピュータビジョン特論 Advanced Computer Vision 第 5 回 ”,
＜ http://www.wakayama-u.ac.jp/ wuhy/CV05.pdf ＞,（2017 年 2 月 6 日アクセス）

[3] Tomasi, C., and Manduchi, R, (1998, January), “Bilateral filtering for gray and color images, ” In Computer Vision, 1998, Sixth International Conference on (pp. 839-846), IEEE.

[4] Choudhury, P., and Tumblin, J, (2005, July), “The trilateral filter for high contrast images and meshes, ” In ACM SIGGRAPH 2005 Courses (p. 5), ACM.

[5] 水野，近藤，深澤，佐々木，“エッジ保存型適応ラテラルフィルタの提案 ”，電子情報通信学会技術研究報告，vol. 115, no. 459, IE2015-109, pp. 85-90, 2016 年 2 月.

[6] MathWorks，“エッジ検出 ”，
＜ https://jp.mathworks.com/help/images/edge-detection.html ＞
（2017 年 2 月 6 日アクセス）

[7] じゅんじ，“しぼりたてブログ　Canny Edge Detection-アルゴリズム ”，
＜ http://d8yd.blog105.fc2.com/blog-entry-84.html ＞（2017 年 2 月 6 日アクセス）

[8] FC2，画像処理ソリューションー判別分析法（大津の二値化），
＜ http://imagingsolution.blog107.fc2.com/blog-entry-113.html ＞，
(2017 年 1 月 25 日アクセス)

[9] Lowe, D. G, (2004), “Distinctive image features from scale-invariant keypoints, ” International journal of computer vision, 60(2), pp.91-110.

[10] Wang, S., You, H., and Fu, K, (2012), “BFSIFT: A novel method to find feature matches for SAR image registration, ”Geoscience and Remote Sensing Letters, IEEE, 9(4), pp.649-653.

[11] 水野，近藤，深澤，佐々木. (2015), “特徴量検出向上に対するバイラテラル拡張型エッジ保存フィルタの性能比較. ”電気電子・情報関係学会東海支部連合大会 (M3-3)

A フィルタプログラムコード

A.1 適応ラテラルフィルタ

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <iostream>
#include <opencv2/opencv.hpp>
#include <cv.h>
#include <cxcore.h>
#include <highgui.h>
#include "afilter.h" //original header file
#define MAX_LEVEL 5
#define MEDIAN_THRESHOLD 10
#define DSFHS 2 //Difficult Smoothing Filter Half Size
using namespace std;
using namespace cv;

Mat adaptivelateralFilter(Mat input,
        float sigmaC1,float sigmaC2,float sigmaC3)
{
  Raw2D destImg;      //Output Image
  Raw2D fTheta;
  //stores Adaptive neighborhood size for each pixel
  Raw3D minGradientStack, maxGradientStack;
  //stores the min and max stack of image gradients
  int level;  //level = log2(xsize) or log2(ysize)
  float sigmaR, sigmaCTheta, R;
  //domain variance for the two filters: sigmaC, sigmaCTheta
  Raw2D* pSrcImg = new Raw2D;
  Gradients imgrad;
  Raw2D blur;
  pSrcImg->sizer(input.cols,input.rows);
  for(int y=0;y<input.rows;y++){
    for(int x=0;x<input.cols;x++){
      pSrcImg->put(x,y,
       input.data[y*input.step+x*input.elemSize()]);
    }
  }
  sigmaCTheta = sigmaC1;
  //Variance of the Domain Filter, the only user set parameter

  //gazo saizu gotoni level wo kimeru
  if(pSrcImg->getXsize()*pSrcImg->getYsize() < 400000) level = 4;
  else level = MAX_LEVEL;

  minGradientStack.sizer(pSrcImg->getXsize(),pSrcImg->getYsize(),level);
  maxGradientStack.sizer(pSrcImg->getXsize(),pSrcImg->getYsize(),level);

  fTheta.sizer(pSrcImg);
  destImg.sizer(pSrcImg);
  imgrad.sizer(pSrcImg);
  blur.sizer(pSrcImg);

  imgrad.computeFullGradients(pSrcImg,&blur);

  sigmaR = pSrcImg->buildImageStack(&imgrad,&minGradientStack,&
      maxGradientStack,level);
```

```
55
56    R = sigmaR * Rp;
57    fTheta.findAdaptiveRegion(&minGradientStack, &maxGradientStack, R,
          level);
58    destImg.filtering(pSrcImg, &fTheta, sigmaC1,sigmaC2,sigmaC3,&imgrad);
59    //Raw to Mat
60    for(int y=0;y<input.rows;y++){
61      for(int x=0;x<input.cols;x++){
62        input.data[y*input.step+x*input.elemSize()] = (uchar)destImg.get(x
              ,y);
63      }
64    }
65    return input;
66  }
67
68
69  void Gradients::computeFullGradients(Raw2D *src,Raw2D *blur){
70    int xmax, ymax;
71    float gauss_kernel[5][5] =
          {{2,4,5,4,2},{4,9,12,9,4},{5,12,15,12,5},{4,9,12,9,4},{2,4,5,4,2}};
72    float sobel_x[3][3] = {{-1,0,1},{-2,0,2},{-1,0,1}};
73    float sobel_y[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
74
75    xmax=getXsize();
76    ymax=getYsize();
77
78    /***********************gauss_blur**********************************
          */
79    for(int y =0;y<ymax;y++){
80      for(int x=0;x<xmax;x++){
81        float tmp = 0;
82        float normFactor = 0;
83        for(int i=-2;i<3;i++){
84    for(int j=-2;j<3;j++){
85      if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
86        tmp += gauss_kernel[i+2][j+2]*src->y[xmax*(y+i)+(x+j)];
87        normFactor += gauss_kernel[i+2][j+2];
88      }
89    }
90        }
91        tmp /= normFactor;
92        blur->y[xmax*y+x] = tmp;
93      }
94    }
95    /*
          ***********************************************************************
          */
96    for(int y =0;y<ymax;y++){
97      for(int x=0;x<xmax;x++){
98        /***********************sobel*********************************
              */
99        float fx=0,fy=0;
100       for(int i=-1;i<2;i++){
101   for(int j=-1;j<2;j++){
102     if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
103       fx += sobel_x[i+1][j+1]*src->y[xmax*(y+i)+(x+j)];
104       fy += sobel_y[i+1][j+1]*src->y[xmax*(y+i)+(x+j)];
105     }
106   }
107       }
```

```
108         /************************gradient_compute
                ******************************/
109         grad[xmax*y+x] = sqrt(fx*fx+fy*fy);
110         gx[xmax*y+x] = fx;
111         gy[xmax*y+x] = fy;
112       }
113     }
114     /*
            ***************************************************************
            */
115 }
116
117
118 float Raw2D::buildImageStack(Gradients *imgrad, Raw3D* pMinStack,
119               Raw3D* pMaxStack , int level)
120 {
121   int imax, jmax, i, j, lev, m, n;
122   float min, max, tmp, tmpMin, tmpMax,aveG=0;
123
124   imax=getXsize();  //get image size
125   jmax=getYsize();
126   for(lev=0; lev < level; lev++) {
127     for(j=0; j < jmax; j++) {
128       for(i=0; i < imax; i++) {
129   if( lev == 0) {
130     //stack level 0 is the magnitude of the gradients of original imag
131     tmp = imgrad->grad[imax*j+i];
132     max = min = tmp;
133     pMinStack->put(i,j,0,min);
134     pMaxStack->put(i,j,0,max);
135     aveG+=tmp;
136   }
137   if( lev > 0) {
138     //Gradients at each level of the image stack is computed from the
            level below
139     min = pMinStack->get(i,j,lev-1);
140     max = pMaxStack->get(i,j,lev-1);
141
142     for(m=-1; m <= 1; m++) {
143       for(n=-1; n <= 1; n++) {
144         //Computes the maximum and minimum gradient value in the
                neighborhood
145         if((i+m) >=0 && (i+m) < imax && (j+n) >=0 && (j+n) < jmax ) {
146     tmpMin = (float) pMinStack->get(i+m,j+n,lev-1);
147     tmpMax = (float) pMaxStack->get(i+m,j+n,lev-1);
148     if(min > tmpMin)
149       min = tmpMin;
150     if(max < tmpMax)
151       max = tmpMax;
152         }
153       }
154     }
155
156     pMinStack->put(i,j,lev,min);
157     pMaxStack->put(i,j,lev,max);
158   }
159       } // for(i...
160     } // for(j...
161   } // for(lev...
162
163   aveG /= jmax*imax;
```

```
164   return aveG;
165
166 }
167
168
169 void Raw2D::findAdaptiveRegion(Raw3D* pMinStack, Raw3D* pMaxStack, float
        R, int level)
170 {
171   int imax, jmax,i,j,lev;
172
173   imax=getXsize();  //get image size
174   jmax=getYsize();
175   for(j=0; j<jmax; j++) {
176     for(i=0; i<imax;i++) {
177       for(lev=0; lev < level; lev++) {
178   if ( pMaxStack->get(i,j,lev) > (pMaxStack->get(i,j,0) + R) ||
179        pMinStack->get(i,j,lev) < (pMaxStack->get(i,j,0) - R) )
180     break;
181       }
182       put(i,j,(float) (lev-1) );
183     } // for(i...
184   } // for(j...
185 }
186
187
188 void Raw2D::filtering(Raw2D* srcImg,
189           Raw2D* fTheta,
190           float sigmaCTheta1,
191           float sigmaCTheta2,
192           float sigmaCTheta3,
193           Gradients *imgrad)
194 //filtering
195 {
196   int i,j,m,n,imax,jmax,halfSize,maxSize = MAX_LEVEL*2+1;
197   float tmp,domainWeight, normFactor,brightWeight,gradWeight;
198   float bufD = 2 * sigmaCTheta1 * sigmaCTheta1;
199   float bufB = 2 * sigmaCTheta2 * sigmaCTheta2;
200     //2 * sigmaCTheta * sigmaCTheta * 5;    //3
201   float bufG = 2 * sigmaCTheta3 * sigmaCTheta3;
202     //2 * sigmaCTheta * sigmaCTheta * 30;   //30
203   int medi[9];//3*3 squea
204   float distW[maxSize][maxSize];
205   //pre-compute disttance weightemacs
206   for(i=-MAX_LEVEL;i<=MAX_LEVEL;i++){
207     for(j=-MAX_LEVEL;j<=MAX_LEVEL;j++){
208       distW[i+MAX_LEVEL][j+MAX_LEVEL] =
209   (float) pow(M_EXP, (double) (-(i*i+j*j)/bufD));
210     }
211   }
212   imax=getXsize();  //get image size
213   jmax=getYsize();
214   for(j=0; j<jmax; j++) { //Y scanline...
215     for(i=0; i<imax; i++) { //X scankline...
216       normFactor=0.0;
217       tmp=0.0;
218       halfSize=(int) fTheta->y[i+imax*j];
219       //difficult smoothing
220       if(halfSize <= 2){
221   bool f = false;
222   int co = 0;
223   //noised pixel checking
```

```
224   for(m=-1;m<=1;m++){
225     for(n=-1;n<=1;n++){
226       if( (i+m) >= 0 && (i+m) <imax && (j+n) >=0 && (j+n) < jmax) {
227         if(srcImg->y[i+imax*j] - srcImg->y[i+m+imax*(j+n)]
228      < MEDIAN_THRESHOLD && !(m==0 && n==0)){
229     f = true;
230     break;
231         }
232         else{
233     medi[co] = srcImg->y[i+m+imax*(j+n)];
234     co++;
235         }
236       }
237     }
238   }
239   //no noised pixel
240   if(f){
241     if(halfSize == 0)
242       halfSize = 1;
243     for(m = -halfSize; m<=halfSize; m++) {
244       for (n = -halfSize; n<=halfSize; n++) {
245         if( (i+m) >= 0 && (i+m) <imax && (j+n) >=0 && (j+n) < jmax) {
246     /*****************************************************************
247                     Appeared magic namber is adjasting the kernel
                            weight.
248     *****************************************************************/
249     //distance
250     domainWeight = distW[m+MAX_LEVEL][n+MAX_LEVEL];
251           //brightness
252     brightWeight =  srcImg->y[i+imax*j] - srcImg->y[i+m+imax*(j+n)];
253     brightWeight = (float) pow(M_EXP, (double) (-(brightWeight*
            brightWeight)/(bufB)));
254     //gradient
255     float dx,dy,gradDiff;
256     dx = imgrad->gx[i+imax*j] - imgrad->gx[i+m+imax*(j+n)];
257     dy = imgrad->gy[i+imax*j] - imgrad->gy[i+m+imax*(j+n)];
258     gradDiff = sqrt(dx*dx+dy*dy);
259     gradWeight = (float) pow(M_EXP, (double) (-(gradDiff*gradDiff)/bufG)
            );
260     //convolution
261     tmp += srcImg->y[i+m+imax*(j+n)] * domainWeight * brightWeight *
            gradWeight;
262     normFactor +=  domainWeight * brightWeight * gradWeight;
263         }
264       }
265     }
266   }
267   //noised pixel
268   else{
269     sort(medi,medi+co);
270     normFactor = 1;
271     tmp = medi[4]; //center factor;
272   }
273       }
274       //easy smoothing
275       else{
276   for(m = -halfSize; m<=halfSize; m++) {
277     for (n = -halfSize; n<=halfSize; n++) {
278       if( (i+m) >= 0 && (i+m) <imax && (j+n) >=0 && (j+n) < jmax) {
279         //distance
280         domainWeight = distW[m+MAX_LEVEL][n+MAX_LEVEL];
```

```
281           tmp += srcImg->y[i+m+imax*(j+n)]* domainWeight;
282           normFactor +=  domainWeight;
283         }
284       }
285     }
286         }
287         if(normFactor != 0){
288     tmp /= normFactor;
289         }
290         else{
291     tmp = srcImg->y[i+imax*j];
292         }
293         put(i,j,tmp);
294       }
295     }
296   }
297
298   //     ============================================================================

299   //Some helper functions for Raw2D and Raw3D class.
300   //     ============================================================================

301
302   void Raw2D::sizer(int ixsize, int iysize) {
303     y = new float[ixsize*iysize]; // & allocate memory.
304     xsize = ixsize;       // set new image size,
305     ysize = iysize;
306   }
307
308   void Raw2D::sizer(Raw2D* src) {
309     int ix, iy;
310     ix = src->getXsize();
311     iy = src->getYsize();
312     sizer(ix,iy);
313   }
314
315   void Raw3D::sizer(int ixsize, int iysize, int izsize) {
316     int i;
317
318     z = new Raw2D[izsize];      // make room for the 2D objects,
319     zsize = izsize;
320     for(i=0; i< zsize; i++)
321       z[i].sizer(ixsize,iysize);  // use the Raw2D sizer.
322   }
323
324   void Gradients::sizer(int ixsize, int iysize) {
325
326     grad = new float[ixsize*iysize];  // & allocate memory.
327     gx = new float[ixsize*iysize];  // & allocate memory.
328     gy = new float[ixsize*iysize];  // & allocate memory.
329     xsize = ixsize;       // set new image size,
330     ysize = iysize;
331   }
332
333   void Gradients::sizer(Raw2D* src) {
334     int ix, iy;
335     ix = src->getXsize();
336     iy = src->getYsize();
337     sizer(ix,iy);
```

```
338 }
```

A.2 ガウシアンフィルタ組み込み

```
1  void Gradients::computeFullGradients(Raw2D *src,Raw2D *blur){
2    int xmax, ymax;
3    float gauss_kernel[5][5] =
         {{2,4,5,4,2},{4,9,12,9,4},{5,12,15,12,5},{4,9,12,9,4},{2,4,5,4,2}};
4    float sobel_x[3][3] = {{-1,0,1},{-2,0,2},{-1,0,1}};
5    float sobel_y[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
6
7    xmax=getXsize();
8    ymax=getYsize();
9
10   /*********************gauss_blur*********************************
         */
11   for(int y =0;y<ymax;y++){
12     for(int x=0;x<xmax;x++){
13       float tmp = 0;
14       float normFactor = 0;
15       for(int i=-2;i<3;i++){
16   for(int j=-2;j<3;j++){
17     if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
18       tmp += gauss_kernel[i+2][j+2]*src->y[xmax*(y+i)+(x+j)];
19       normFactor += gauss_kernel[i+2][j+2];
20     }
21   }
22       }
23       tmp /= normFactor;
24       blur->y[xmax*y+x] = tmp;
25     }
26   }
27   /*
         ***********************************************************************
         */
28   for(int y =0;y<ymax;y++){
29     for(int x=0;x<xmax;x++){
30       /*********************sobel*********************************
             */
31       float fx=0,fy=0;
32       for(int i=-1;i<2;i++){
33   for(int j=-1;j<2;j++){
34     if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
35       fx += sobel_x[i+1][j+1]*blur->y[xmax*(y+i)+(x+j)];
36       fy += sobel_y[i+1][j+1]*blur->y[xmax*(y+i)+(x+j)];
37     }
38   }
39       }
40       /*********************gradient_compute
             ******************************/
41       grad[xmax*y+x] = sqrt(fx*fx+fy*fy);
42       gx[xmax*y+x] = fx;
43       gy[xmax*y+x] = fy;
44     }
45   }
```

```
46    /*
          ************************************************************************
          */
47  }
```

A.3 先鋭化フィルタ組み込み

```
1   void Gradients::computeFullGradients(Raw2D *src,Raw2D *blur){
2     int xmax, ymax;
3     float sobel_x[3][3] = {{-1,0,1},{-2,0,2},{-1,0,1}};
4     float sobel_y[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
5
6     xmax=getXsize();
7     ymax=getYsize();
8
9     float sharpen_kernel[3][3] =
10      {{-1,-1,-1},{-1,16,-1},{-1,-1,-1}};
11    for(int y =0;y<ymax;y++){
12      for(int x=0;x<xmax;x++){
13        float tmp = 0;
14        float normFactor = 0;
15        for(int i=-1;i<2;i++){
16      for(int j=-1;j<2;j++){
17        if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
18          tmp += sharpen_kernel[i+1][j+1]*src->y[xmax*(y+i)+(x+j)];
19          normFactor += sharpen_kernel[i+1][j+1];
20        }
21      }
22        }
23        tmp /= normFactor;
24        blur->y[xmax*y+x] = tmp;
25      }
26    }
27    for(int y =0;y<ymax;y++){
28      for(int x=0;x<xmax;x++){
29        /************************sobel*********************************
              */
30        float fx=0,fy=0;
31        for(int i=-1;i<2;i++){
32    for(int j=-1;j<2;j++){
33      if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
34        fx += sobel_x[i+1][j+1]*blur->y[xmax*(y+i)+(x+j)];
35        fy += sobel_y[i+1][j+1]*blur->y[xmax*(y+i)+(x+j)];
36      }
37    }
38        }
39        /************************gradient_compute
              *******************************/
40        grad[xmax*y+x] = sqrt(fx*fx+fy*fy);
41        gx[xmax*y+x] = fx;
42        gy[xmax*y+x] = fy;
43    }
44    /*
          ************************************************************************
          */
45  }
```

A.4 判別分析法（勾配に適応）

```
1  float computeR(Gradients* grad, int size){//size 勾配の最大値
2    //大津の方法
3    int imax,jmax;
4    int maxsize = (int)size + 1;
5
6    imax=grad->getXsize();  //get image size
7    jmax=grad->getYsize();
8
9    int g[maxsize];
10   float sum = 0;
11   float max_no;
12   float data = 0;
13   float average, average1, average2;
14
15   float max = 0.0;
16   float count1, count2;
17   float breakup1, breakup2;
18   float class1, class2;
19   float tmp;
20
21   //syokika
22   for(int i = 0;i < maxsize;i++)
23     g[i] = 0;
24
25   for(int j=0;j<jmax;j++){   //は画像の縦幅y_size
26     for(int i=0;i<imax;i++){   //は画像の横幅x_size
27       //勾配の値の数値をそれぞれカウント
28       g[(int)grad->grad[i+imax*j]]++;
29       sum += grad->grad[i+imax*j];
30     }
31   }
32   average = sum / (imax * jmax);
33   for(int i=0;i<maxsize;i++){
34     count1 = count2 = 0;
35     data = 0;
36     breakup1 = breakup2 = 0.0;
37     tmp = 0;
38     for(int j=0;j<i;j++){
39       count1 += g[j];
40       data += g[j] * j;
41     }
42     /*クラスの平均1*/
43     /*平均 データの総和=( / 個数)*/
44     if(count1 != 0){
45       average1 = data / count1;
46       /*分散*/
47       /*分散データ=( - 平均値)の総和個数^2/ */
48       for(int j=0;j<i;j++){
49   breakup1 += pow( (j- average1), 2 ) * g[j];
50       }
51       breakup1 /= count1;
52     }
53     data = 0;
54     for(int j=i;j < maxsize;j++){
55       count2 += g[j];
56       data += g[j] * j;
57     }
58     if(count2 != 0){
```

```
59        average2 = data / count2;
60        for(int j=i;j < maxsize;j++){
61    breakup2 += pow( (j - average2), 2 ) * g[j];
62        }
63        breakup2 /= count2;
64      }
65      /*クラス内分散*/
66      class1 = (count1 * breakup1 + count2 * breakup2);
67      /*クラス間分散*/
68      class2 = count1 * pow( (average1 - average), 2 ) + count2 * pow( (
            average2 - average), 2 );
69      tmp = class2 / class1;
70      if(max < tmp){
71        max = tmp;
72        max_no = i;
73      }
74    }
75    return max_no;
76  }
```

A.5 分散値による切り替えフィルタ

```
1   #include <stdio.h>
2   #include <stdlib.h>
3   #include <math.h>
4   #include <iostream>
5   #include <opencv2/opencv.hpp>
6   #include <cv.h>
7   #include <cxcore.h>
8   #include <highgui.h>
9   #include "newfil.h" //original header file
10
11  #define filtersize 5
12
13  using namespace std;
14  using namespace cv;
15
16  //FUNCTION CALL THAT IMPLEMENTS TRILATERAL FILTER
17  //
        ==========================================================================================
18  Mat newfilter(Mat input, float sigmaC)
19  //
        ==========================================================================================
20  {
21    data* inputdata = new data;//input data
22    data outputdata;//output data
23    data disperdata;//dispersion data
24
25    float maxdisper,thre_value;
26
27    inputdata->sizer(input.cols,input.rows);
28    outputdata.sizer(inputdata);
29    disperdata.sizer(inputdata);
30
31    for(int y = 0;y < input.rows;y++){
```

```
32      for(int x = 0;x < input.cols;x++){
33        inputdata->put(x,y,input.data[y*input.step + x*input.elemSize()]);
34      }
35    }
36
37    maxdisper = disperdata.compute_dispersion(inputdata);
38
39    thre_value = disperdata.compute_threshold(maxdisper);
40
41    outputdata.filtering(inputdata,&disperdata,thre_value,sigmaC);
42
43    for(int y = 0;y < input.rows;y++){
44      for(int x = 0;x < input.cols;x++){
45        input.data[y*input.step + x*input.elemSize()] =
46    (uchar)outputdata.get(x,y);
47      }
48    }
49
50    return input;
51  }
52
53  float data::compute_dispersion(data* inputdata){
54    //bunsan keisan
55    int xsize = getxsize();
56    int ysize = getysize();
57    int halfsize = (filtersize-1)/2;//filtersize
58
59    float maxdisper = 0;
60
61    for(int y = 0;y < ysize;y++){
62      for(int x = 0;x < xsize;x++){
63        float ave,sum = 0;
64        float disper;//bunnsan ti
65        int N = 0;//sousuu
66        for(int i = -halfsize;i <= halfsize;i++){
67    for(int j = -halfsize;j <= halfsize;j++){
68      if((i+y) >=0 && (i+y) < ysize && (j+x) >=0 && (j+x) < xsize){
69        sum += inputdata->y[(i+y)*xsize + (j+x)];
70        N++;
71      }
72    }
73        }
74        ave = sum/N;
75        sum = 0;
76        for(int i = -halfsize;i <= halfsize;i++){
77    for(int j = -halfsize;j <= halfsize;j++){
78      if((i+y) >=0 && (i+y) < ysize && (j+x) >=0 && (j+x) < xsize){
79        float tmp = inputdata->y[(i+y)*xsize + (j+x)];
80        sum += (tmp - ave) * (tmp - ave);
81      }
82    }
83        }
84        disper = sum/N;
85
86        if(maxdisper < disper)
87    maxdisper = disper;
88        put(x,y,disper);
89      }
90    }
91    return maxdisper;
92  }
```

```
93
94  float data::compute_threshold(float size){
95    float value = 0.0;
96    int xsize = getxsize();
97    int ysize = getysize();
98    for(int y = 0;y < ysize;y++){
99      for(int x = 0;x < xsize;x++){
100       value += get(x,y);
101     }
102   }
103   return value / (xsize*ysize);
104 }
105
106 void data::filtering(data* input,data* disper,float t,float sigmaC){
107   int xsize = getxsize();
108   int ysize = getysize();
109   float sigma = 2 * sigmaC * sigmaC;
110   int halfsize = (filtersize-1)/2;//filtersize
111   float distW[filtersize][filtersize];
112
113   float diff,tmp,domainWeight,normFactor,brightWeight;
114
115   for(int i = -halfsize;i <= halfsize;i++){
116     for(int j = -halfsize;j <= halfsize;j++){
117       distW[i+halfsize][j+halfsize] =
118   (float)pow(M_EXP,(double)(-(i*i+j*j)/sigma));
119     }
120   }
121   //filtering
122   for(int y = 0;y < ysize;y++){
123     for(int x = 0;x < xsize;x++){
124       tmp = 0;
125       normFactor = 0;
126       if(disper->y[y*xsize + x] < t){
127   //gaussian
128   for(int m=-halfsize;m<=halfsize;m++){
129     for(int n=-halfsize;n<=halfsize;n++){
130       if( (y+m)>=0 && (y+m)<ysize && (x+n)>=0 && (x+n)<xsize) {
131         domainWeight = distW[m + halfsize][n + halfsize];
132         tmp += input->y[(y+m)*xsize+(x+n)] * domainWeight;
133         normFactor += domainWeight;
134       }
135     }
136   }
137   tmp /= normFactor;
138       }
139       else{
140   //bilateral
141   for(int m=-halfsize;m<=halfsize;m++){
142     for(int n=-halfsize;n<=halfsize;n++){
143       if( (y+m)>=0 && (y+m)<ysize && (x+n)>=0 && (x+n)<xsize) {
144         domainWeight = distW[m + halfsize][n + halfsize];
145         //bright
146         diff = input->y[y*xsize+x] - input->y[(y+m)*xsize+(x+n)];
147         brightWeight = (float) pow(M_EXP,
148            (float) (-(diff*diff)/sigma));
149         tmp += input->y[(y+m)*xsize+(x+n)] *
150     domainWeight * brightWeight;
151         normFactor += domainWeight * brightWeight;
152       }
153     }
```

```
154   }
155   tmp /= normFactor;
156       }
157       put(x,y,tmp);
158     }
159   }
160 }
161
162 void data::sizer(int ixsize,int iysize){
163   y = new float[ixsize * iysize];
164   xsize = ixsize;
165   ysize = iysize;
166 }
167
168 void data::sizer(data* idata){
169   int ix,iy;
170   ix = idata->getxsize();
171   iy = idata->getysize();
172   sizer(ix,iy);
173 }
```

B ヘッダファイル

B.1 適応ラテラルフィルタ

```
1
2  #define M_EXP 2.71828182845904523536028747135 27
3  #define Rp 1.4
4  //=====================
5  // Forward Declarations
6  //=====================
7  class Raw2D;
8  class Raw3D;
9
10 class Gradients;
11
12 cv::Mat adaptivelateralFilter(cv::Mat input,
13         float sigmaC1,float sigmaC2,float sigmaC3);
14
15 class Raw2D /*: public CObject*/ {
16  public:         //-----------------DATA-----------------
17   int xsize;     // # of pixels per scanline,
18   int ysize;     // # of scanlines in this Raw2D.
19   float *y;   // 1D array of PIXTYPE that are accessed as a 2D array.
20
21  public:         //----------------init fcns-------------
22       Raw2D(void){}   // constructor for 'empty' Raw2Ds
23       ~Raw2D(void){}    // destructor; releases memory
24     void sizer(int ixsize, int iysize);   // get mem for rectangle of
           pixels
25     void sizer(Raw2D* src);          // get same amt. of mem as 'src'
26   int getXsize(void) {return xsize;};   // get # pixels per scanline
27   int getYsize(void) {return ysize;};   // get # of scanlines.
28     void put(int ix, int iy, float val) { // write 'val' at location ix,
           iy.
```

```
29    y[ix + xsize*iy] = val;
30    };
31      float get(int ix, int iy) { // read the value at ix,iy.
32      return(y[ix + xsize*iy]);
33    };
34    float getXY(int ixy){   // read value at 1D address ixy
35      return y[ixy];
36    };
37    void putXY(int ixy,float val){// write value at 1D address ixy
38      y[ixy] = val;
39    };
40    //Computes X and Y gradients of the input image
41      void ComputeGradients(Raw2D* pX, Raw2D* pY);
42
43    //from the min-max gradient stack
44      void findAdaptiveRegion(Raw3D* pMinStack, Raw3D* pMaxStack, float R,
            int levelMax);
45
46    //Filters the detail signal and computes the final output image
47    void filtering(Raw2D* srcImg,
48              Raw2D* fTheta,
49              float sigmaCTheta1,
50              float sigmaCTheta2,
51              float sigmaCTheta3,
52              Gradients *imgrad);
53
54    float buildImageStack(Gradients *imgrad, Raw3D* pMinStack,
55                  Raw3D* pMaxStack , int level);
56  };
57  /**
58    A 3D array of pixels, consisting of a 1D array of Raw2D objects.
59    Use this class for images, with one Raw2D object for each
60    a 'field' of that image, such as R,G,B,A, etc.
61  **/
62
63  class Raw3D /*: public CObject*/ {
64  public:
65        Raw2D *z; // dynam. allocated space for a set of Raw2D objects.
66        int zsize;  // # of Raw2D objects stored.
67
68  public:
69        Raw3D(void){} // 'empty' Raw3D constructor.
70        ~Raw3D(void){}  // destructor.
71        void sizer(int ixsize, int iysize, int izsize); // reserve memory
72        void sizer(Raw3D* src);   // get same amt. of mem as 'src
73    int getZsize(void) {      // return # of Raw2D's we hold;
74      return(zsize);
75    };
76    int getYsize() {      // # of Raw1D's in zval-th Raw2D;
77      return(z[0].getYsize());
78    };
79    int getXsize(){       // # of pixels on yval,zval-th line
80      return(z[0].getXsize());
81    };
82        float get(int ix, int iy, int iz) {
83          return(z[iz].get(ix,iy)); // write 'val' at location ix,iy,iz.
84        };
85        void put(int ix, int iy, int iz, float val) {
86          z[iz].put(ix,iy,val); //write 'val' at location ix,iy,iz.
87        };
88    void wipecopy(Raw3D& src);  // copy, resize as needed.
```

```
89 };
90
91 class Gradients{
92  public:
93   int xsize;
94   int ysize;
95   float *gx;
96   float *gy;
97   float *grad;
98   /* int *direct; */
99
100  public:
101       Gradients(void){}
102       ~Gradients(void){}
103   int getXsize(void) {return xsize;};   // get # pixels per scanline
104   int getYsize(void) {return ysize;};   // get # of scanlines.
105     void sizer(int ixsize, int iysize);   // get mem for rectangle of
          pixels
106     void sizer(Raw2D* src);         // get same amt. of mem as 'src'
107   void computeFullGradients(Raw2D *src,Raw2D *blur);
108
109 };
```

B.2 分散値による切り替えフィルタ

```
1  #define M_EXP 2.71828182845904523536028747135 27
2
3  class data;
4
5  class data{
6   public:
7    int xsize;
8    int ysize;
9    float *y;
10
11   public:
12    data(void){}
13    ~data(void){}
14
15    void sizer(int ixsize,int iysize);
16    void sizer(data* idata);
17
18    int getxsize(void){return xsize;};
19    int getysize(void){return ysize;};
20    float get(int ix,int iy){return y[ix + iy*xsize];};
21    void put(int ix,int iy,float val){y[ix + iy*xsize] = val;};
22
23    float compute_dispersion(data* inputdata);
24    float compute_threshold(float size);
25    void filtering(data* input,data* disper,float t,float sigmaC);
26 };
```