修士論文

題目

エッジ保存型適応ラテラルフィルタの提案

指導教員

近藤 利夫

平成 27 年度

三重大学大学院 工学研究科 情報工学専攻
コンピュータ・アーキテクチャ研究室

水野 篤 (414M521)

内容梗概

ハードウェアの性能向上にともな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動画像を扱う高度なアルゴリズムが盛んに研究されている．これらのアルゴリズムのほとんどで前処理に平滑化を行っており，使用する平滑化フィルタの特性がその後の処理結果に大きく影響するために，平滑化フィルタの重度性が増してきている．しかし，ガウシアンフィルタに代表される従来の平滑化フィルタは，平滑化の際に空間的距離などの画素値に依存しない情報を用いるので，ノイズや細部だけでなく動画像において重要なエッジもぼかしてしまう問題がある．

そこで，エッジを保ちつつ平滑化が可能なエッジ保存型フィルタが注目を集めている．エッジ保存型平滑化フィルタは，フィルタカーネルの計算に従来使用していなかった画素値に依存する重みを使っており，これによりエッジ付近の平滑化の度合いを抑えることで，エッジを残した平滑化を可能にしている．中でも，ガウシアンフィルタをベースに重みの計算に空間的距離と注目画素との画素値の差を利用するバイラテラルフィルタは最もベーシックなエッジ保存型平滑化フィルタとなっているが，エッジ強調特性に起因する不自然なノイズがエッジ付近に発生してしまう問題や平坦領域の平滑化が従来の平滑化フィルタに比べて劣ってしまうなどの課題がある．これらの解決のために，様々な研究者によってバイラテラルフィルタの改良やバイラテラルフィルタをベースにした新しいエッジ保存平滑化フィルタアルゴリズムの提案がされてきているものの，処理量の大幅な増加により処理速度の低下を招いてしまっており，高度なエッジ保存性，強力な平滑化能力，高速な実行の並立が困難という課題がある．

本論文では，高度なエッジ保存性，強力な平滑化能力，高速な実行の並立を目指し，新しいエッジ保存型フィルタアルゴリズムである適応ラテラル型のフィルタを提案した．提案フィルタは，画素毎に求めた勾配情報を利用して領域検出を行い，その結果に基づきエッジ部とそれ以外の領域で，複雑で高度なカーネルと単純で高速なカーネルを切り替えることでバイラテラルフィルタの課題を解決している．また，平滑化とエッジ保存の両立が不可欠な特徴量抽出アルゴリズム SIFT に組み込むことで性能評価をし，その結果シンプルなバイラテラルフィルタと同程度のフィルタ処理速度を実現しながら，高度なエッジ保存平滑化能力を有す

るフィルタアルゴリズムのトリラテラルフィルタと同程度かそれ以上の特徴点対応精度を有していることを明らかにした.

Abstract

Recently computer vision and computer graphics for example video image algorithms is studied and developed actively all over the world. Image smoothing has been used in most of these algorithms, and the importance of smoothing filter is increasing because of smoothing processing affects these algorithms result. However, traditional smoothing filter represented by gaussian filter has a problem which smoothing also the important edges not only noise content.

Then, edge-preserving filtering has attracted a lot of attention. Bilateral filter is the most general edge-preserving filter, but Bilateral filter has two problems that can not be strongly smoothing flat region and create an unnatural noise. Some researchers proposed enhanced bilateral filter algorithm to resolve these problems, but the improvement causes lower execution speed due to its computational complexity.

In this paper, we proposed Adaptive-lateral filter which is new edge-preserving filter algorithm for both of advanced edge-preserving, strong smoothing and fast execution. Proposed filter solve problems of bilateral filter with switching between high speed strong non-edge-preserving filter and advanced edge-preserving filter. In addition, We imbedded the proposed filter in a SIFT algorithm in order to show its effectiveness. Experimental results show that the equivalent proposed Adaptive-lateral filter provide processing speed to that of basic Bilateral Filter and equivalent performance to that of advanced Trilateral Filter.

目 次

図 目 次

表 目 次

1　はじめに

近年，平滑化フィルタはダイナミックレンジ圧縮 [1]，メッシュノイズ除去 [2]，特徴量抽出での差分画像生成 [3][4][5] 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の分野ではノイズ除去以外の目的でも利用しており、平滑化フィルタの性能向上は重要な課題である。とりわけ、特徴抽出アルゴリズム SIFT[6] 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をする事で階層画像 (Image-pyramid) の生成を行っているために平滑化の特性が最終的な認識精度に大きく影響することが知られている．なかでも、エッジ成分を保持しながら信号を平滑化することが可能なエッジ保存フィルタを利用する手法が注目されており、平滑化フィルタをガウシアンフィルタからエッジ保存フィルタのバイラテラルフィルタ [7] に置き換える試みがされ、筆者らも，トリラテラルフィルタ [8] への置き換えにより特徴点対応精度がかなり向上することを確認している [9]．これはエッジ境界付近の特徴点は誤対応を起こしやすく (開口問題)，エッジ保存の性能が高いほど開口問題の影響を受けにくくなるからである．しかし，単純なガウシアンフィルタに比べ、高度な

エッジ保存フィルタは平滑化時間が著しく大きい。また、エッジ保存とは画素勾配の大きな領域を平滑化しないことで成り立っているためにノイズも保存してしまいがちで、ガウシアンフィルタをはじめとする従来の平滑化フィルタよりもノイズ除去能力が劣っている。

そこで，本研究では，エッジ保存性能、高速性、高いノイズ除去能力の並立を目指し，領域検出結果に基づきエッジ領域と平坦領域で複雑で高度なカーネルと単純で高速なカーネル切り替え、孤立点でのみメディアンフィルタを使用する適応ラテラル型のフィルタを提案する．また，既存のエッジ保存フィルタと共にSIFT組み込み，特徴点対応精度を比較することでエッジ保存平滑化性能の評価・検証を行い，高いエッジ保存平滑化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　エッジ保存フィルタとその問題点

バイラテラルフィルタは最も代表的な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 [10] や，バイラテラルフィルタをベースとした高性能なエッジ保存フィルタの研究 [8][11] が行われてきた．この章では一例としてバイラテラルフィルタと高性能ながら複雑なトリラテラルフィルタについてそれぞれの特徴と問題点について述べる．

2.1　バイラテラルフィルタ (Bilateral 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，

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

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．

この欠点を改善すべく 1998 年に Tomasi ら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 [7] である．

バイラテラルフィルタは，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，各フィルタの出力結果である図 2.1(a)(b) の画像の一部を拡大した図 2.1(d)(e) から明らかなように平坦な領域においてはガウシアンフィルタよりも平滑化性能が低くなってしまっている．

2.2 トリラテラルフィルタ (Trilateral Filter)

トリラテラルフィルタ [8] は 2003 年に Choudhury ら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．図 2.1(c) の拡大である図 2.1(f) からわかるように，エッジ間近までバイラテラルフィルタよりも高度に平滑化されていることがわかる．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．このため各フィルタを 3 種類の異なる大きさの画像に掛けたときの実行時間は，表 4.2 に示されるように，他のフィルタと比較してトリラテラルフィルタが非常に低速であることがわかる．

図 2.1: 各フィルタの出力結果及びその拡大図 $*^{1}$]

$*^{1}$](a)$\sigma = 3$，ウィンドウサイズ $= 5$．(b)$\sigma_1 = 3$，$\sigma_2 = 3$，ウィンドウサイズ$= 5$．(c)$\sigma = 3$ をそれぞれパラメータとして与えている．

3　エッジ保存型適応ラテラルフィルタの提案

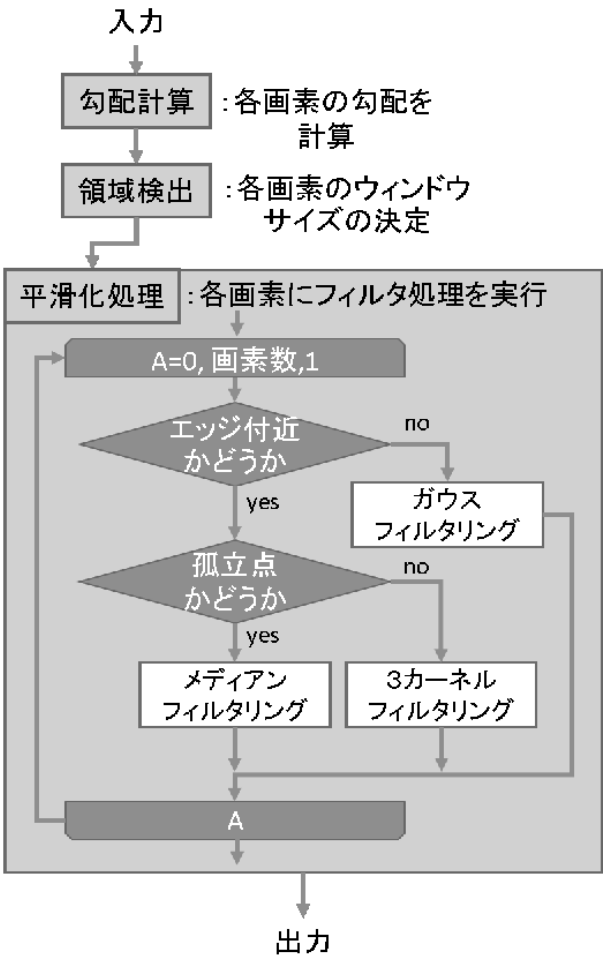


図 3.2: 提案フィルタフローチャート

本研究では高性能かつ高速な新しいエッジ保存フィルタである適応ラテラルフィルタ (Adaptive-lateral filter) を提案する．提案フィルタの主要な特徴は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な領域を 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るところにある．また、平坦な領域をガウシアンフィルタで平滑化していること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。このフィルタの処理は図3.2に示されるように大きく三つに分かれている．以下で，それぞれについて説明する．

3.1 勾配計算

本フィルタでははじめに入力画像から各画素毎の勾配計算を行っている．ここで計算した勾配情報は以下で説明する領域検出で利用するだけでなく，平滑化の際にもカーネルの重みとして使用する．勾配計算方法にはエッジ検出においてもっとも有効であると言われているキャニー法[12]の勾配抽出処理に基づいている．勾配計算の手順は以下である：

1. ガウシアンフィルタによる入力画像の平滑化
2. ソーベルフィルタ (図3.3) を使用して縦・横方向それぞれの勾配を算出
3. 求めた勾配を合成し勾配ベクトルを生成

ここで求めた縦・横方向の勾配はフィルタリングの際に，それらから計算した勾配ベクトルは領域拡張法で使用している．

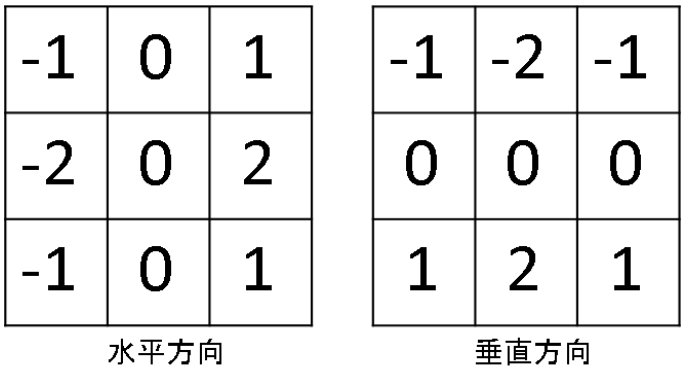


図 3.3: 各方向のソーベルフィルタカーネル

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てい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．同一領域かどうかの基準には、先に計算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入力として受け取った勾配判定精度調整用の変数 R を掛け合わせたものを使っている。注目画素から徐々に参照範囲を広げていき，別領域と思われる点に当たるまでを同一領域と判定し，それまでに広げた距離をその画素のウィンドウサイズとしている (図 3.4).

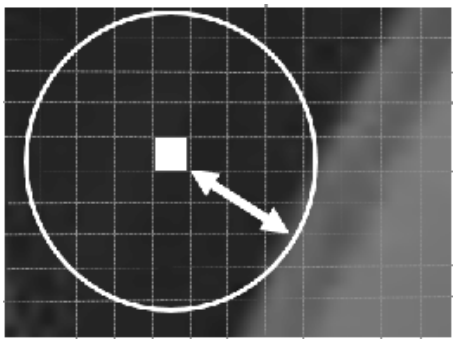


図 3.4: 領域拡張法

3.3 平滑化処理

平滑化は領域毎に 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スフィルタ，孤立点除去を目的としたメディアンフィルタに加えて，エッジ付近のエッジ保存平滑化を目的としたフィルタを使用している．このうち 3 つめのフィルタは，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：

$$
\begin{aligned}
C &= \exp(-\frac{\sqrt{D+E}}{2\sigma_3^2}), \\
D &= (x(i,j) - x(i+m, j+n))^2, \\
E &= (y(i,j) - y(i+m, j+n))^2.
\end{aligned}
$$

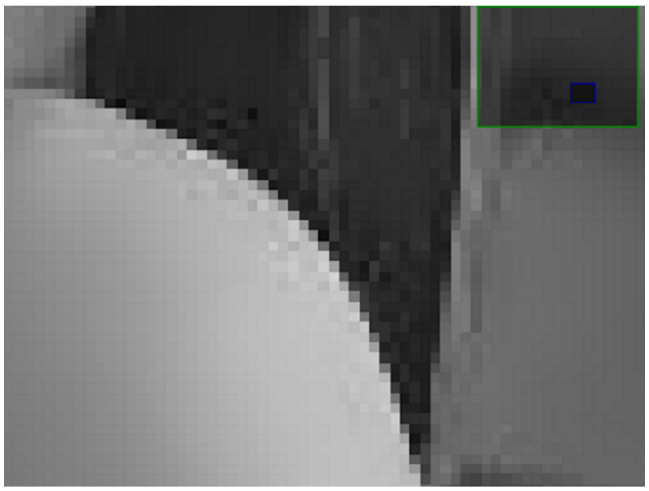
これら 3 つの重みを掛け合わせたエッジ付近のエッジ保存平滑化を目的とした 3 カーネルのフィルタは以下の式で表される：

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

LUT はフィルタの高速化にも効果的である．しかし，3 種類のカーネルを使っている従来のトリラテラルフィルタでは，LUT を適用できるのが基本的に事前計算の可能な空間的距離を用いるガウスカーネルのみなので，最大でも 3 割程度の処理時間低減が限度であると考えられる．一方，提案フィルタでは複雑なカーネルの使用を急勾配領域に限定しているため，大部分がガウスカーネルを用いて平滑化することになり，LUT による高速化の恩恵を最大限に受けられる．ただし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．この孤立点の検出は急勾配領域でのみ行うようにしているので全体の処理量増加を最低限に抑えられる．これら

により高性能かつ高速な平滑化が可能になる．提案フィルタの出力結果の図 3.5(a) の一部を拡大した図 3.5(b) が示すとおり，エッジを残しつつ平坦な領域はガウシアンフィルタの出力である図 2.1(d) のようになめらかに平滑化できる．ノイズ除去能力に関しても，バイラテラルフィルタ (図 3.6(a)) はノイズを平滑化した際にシミのようになってしまい，トリラテラルフィルタ (図 3.6(b)) はうまくノイズ除去を行えていないが，提案フィルタ (図 3.6(c)) では効率的にノイズ除去をしつつ平滑化ができている．

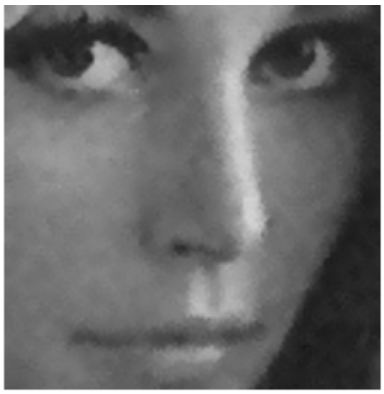
(a)提案フィルタ



(b)提案フィルタ(拡大)

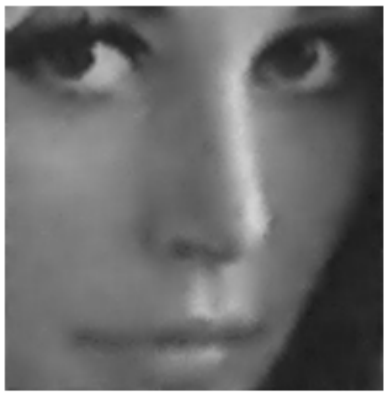
図 3.5: 提案フィルタの出力結果とその拡大 $*^2$

$*^2$] 入力パラメータは以下に設定した．$\sigma_1 = 3$，$\sigma_2 = 5$，$\sigma_3 = 21$



(a)バイラテラルフィルタ

(b)トリラテラルフィルタ



(c)提案フィルタ

図 3.6: ノイズ付加画像に対する各フィルタの出力 $*^{3}$
$*^{3}$] 入力パラメータは以下に設定した．(a)$\sigma_1 = 20$，$\sigma_2 = 20$，ウィンドウサイズ $= 5$ (b)$\sigma = 20$ (c)$\sigma_1 = 34$，$\sigma_2 = 5$，$\sigma_3 = 141$

4　エッジ保存フィルタの性能評価

現在，フィルタに関する研究は盛んにされてきているものの，フィルタを評価する明確な指標は存在しておらず，評価のほとんどを未だに人間の視覚に頼っているのが実情である．本研究ではエッジ保存性能と平滑化性能が最終的なマッチング精度に大きく影響する局所特徴量を用いた物体認識アルゴリズムでフィルタの性能評価を行っている (4.2 章)．AKAZE[4] のような高速で高精度な特定物体認識アルゴリズムや，高い認識精度で注目されているディープラーニングをはじめとする一般物体認識アルゴリズムなど様々な物体認識アルゴリズムが近年注目され，盛んに研究されているが，本研究ではその中でも特定物体認識アルゴリズムにおいて草分けとなっているベーシックな SIFT アルゴリズム [6] を評価用アルゴリズムとして採用する．

4.1　物体認識アルゴリズム SIFT

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 の処理は主に特徴点 (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：

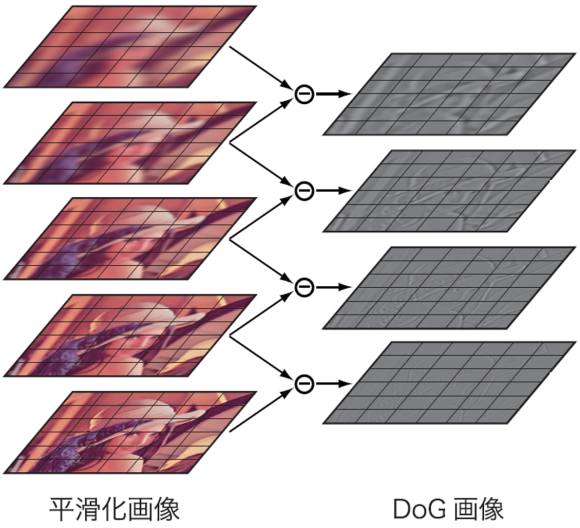


図 4.7: DoG 画像生成 [4]

1. 特徴点抽出

   - 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

特徴量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．図 4.7(出典:[13]，藤吉弘亘，Gradient ベースの特徴抽出，図 5) に示すように 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 [5][9]．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．

4.2 性能評価

提案フィルタの性能評価は，単純で速いバイラテラルフィルタと低速だが高品質なトリラテラルフィルタを SIFT に組み込んだ際の対応点検出精度を比較することで行う．DoG 画像の生成方法でそれぞれのフィルタについて DoB(Difference of Bilateral)，DoT(Difference of Trilateral)，DoA(Difference of Adaptive-lateral) 画像を作り，SIFT アルゴリズムの DoG 画像の代わりとして使用し，マッチングの正しい対応率を求めて比較する．ただし，トリラテラルフィルタに関しては著者らが示した [9] ように対応点検出精度への影響が出にくいことから，高速化のためにウィンドウサイズサイズに制限をかける．

SIFT アルゴリズムを用いて評価した対応点検出精度を表 4.1 にまとめる．対応点検出精度は実際に対応した点の内，座標を用いて正しく対応できていると判定できた点の割合で求めた．評価にはテスト画像及び，テスト画像の一部を切り出し 2 倍に拡大した後，加工したペアを用いる．加工方法はノイズ付加 (_n)，コントラストの変更 (_c)，jpeg 圧縮による劣化 (_d) で，テスト画像は 2 種類用意した．また，実行速度は SIFT アルゴリズムとは別で 3 種類の解像度の画像で測定したものを表 4.2 に示す．トリ

[4](藤吉弘亘，Gradient ベースの特徴抽出-SIFT と HOG-，情処学会研報，CVIM, 160，pp.213，図 5 (参考文献 [13]) より引用

ラテラルフィルタについてはウィンドウサイズ制限したものとしていないものの両方を記載する．

表 4.1，4.2 が示すとおり，エッジ保存フィルタ使った場合，通常のガウシアンフィルタを使ったものより軒並み対応点検出精度が高い．提案フィルタは対応点検出精度が最も高いだけでなく，実行速度も最も速く，優れたエッジ保存平滑化性能と高速性が両立されていることがわかる．

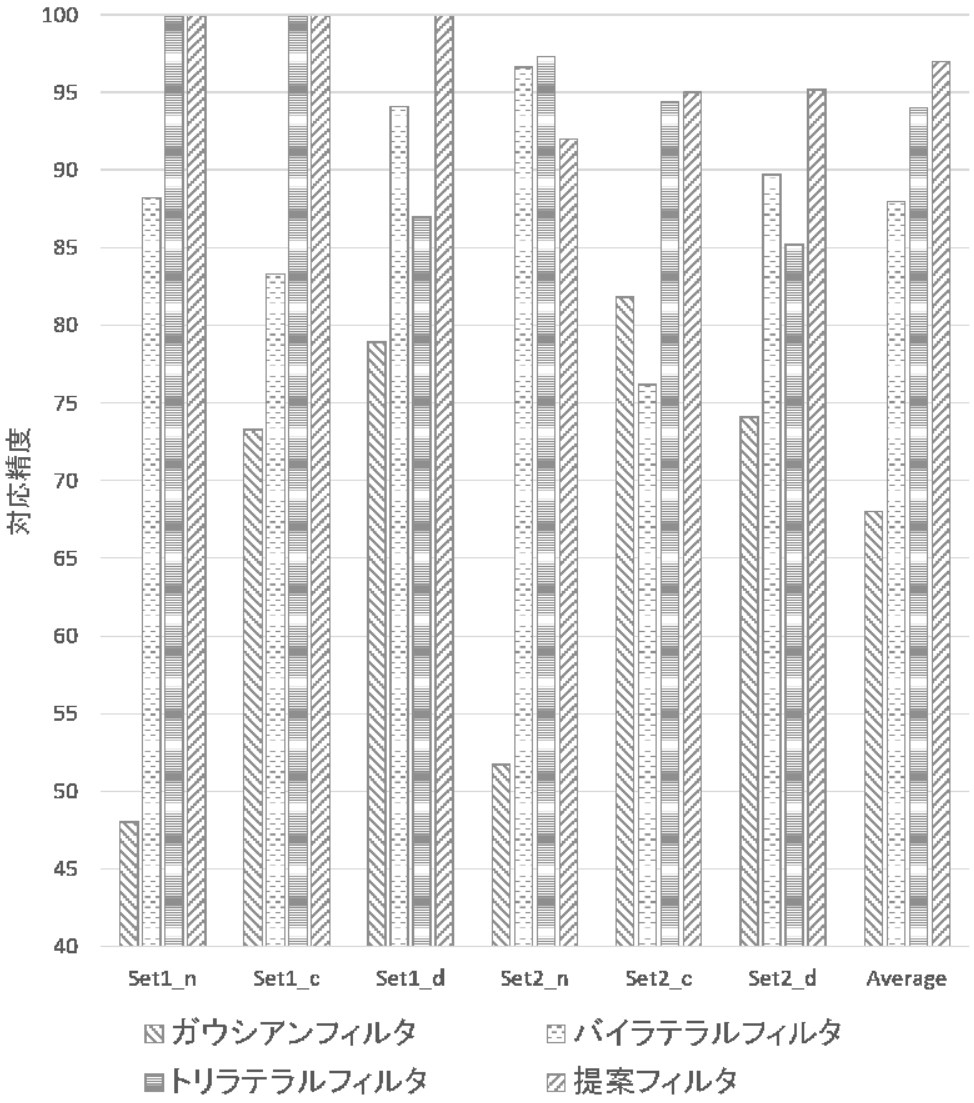


図 4.8: 精度評価グラフ [5]

表 4.1: 精度評価表 $*^{5}$

	Set1_n	Set1_c	Set1_d	Set2_n	Set2_c	Set2_d	Average
ガウシアンフィルタ	12/25	11/15	15/19	15/29	9/11	20/27	68.0
バイラテラルフィルタ	15/17	15/18	16/17	28/29	16/21	26/29	88.0
トリラテラルフィルタ	4/4	25/25	40/46	71/73	51/54	52/61	94.0
提案フィルタ	46/46	37/37	22/22	23/25	19/20	20/21	97.0

$*^{5}$] 基本的にパラメーターはそれぞれ SIFT アルゴリズムで用いられているデフォルトのものを使用している．トリラテラルフィルタと提案フィルタについては内部で $\sigma* = 3$ の値を設定している．

表 4.2: フィルタ毎の実行時間 (秒)[6]

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 (制限なし)	トリラテラルフィルタ (制限あり)	提案フィルタ
126*128	0.03	0.05	4.45	0.11	0.04
400*300	0.18	0.38	423.57	0.82	0.30
864*430	0.57	1.01	1399.23	2.14	0.82

$*^{6}$] 入力パラメータは以下に設定した．ガウシアンフィルタ)$\sigma = 3$，ウィンドウサイズ $= 5$ バイラテラルフィルタ)$\sigma_1 = 3$，$\sigma_2 = 3$，ウィンドウサイズ $= 5$ トリラテラルフィルタ (制限あり・なし))$\sigma = 3$，提案フィルタ)$\sigma_1 = 3$，$\sigma_2 = 5$，$\sigma_3 = 21$

5　おわりに

領域検出を行うことエッジ領域と平坦領域でカーネルを切り替える高速で高性能な適応ラテラル型のフィルタを提案した．また，既存のフィルタと実行速度を比較すると共に，特徴量検出アルゴリズム 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平滑化性能を評価した．その結果として，トリラテラルフィルタよりもエッジ保存平滑化性能が高く，高速実行可能であることを示した．

しかし，エッジ付近の平滑化に関しては図 5.9 に示すとおり，トリラテラルフィルタの方がうまく平滑化できている部分もある．このため，一層の平滑化の高度化には，エッジ領域用のフィルタカーネルの改良が有効と考えられる．

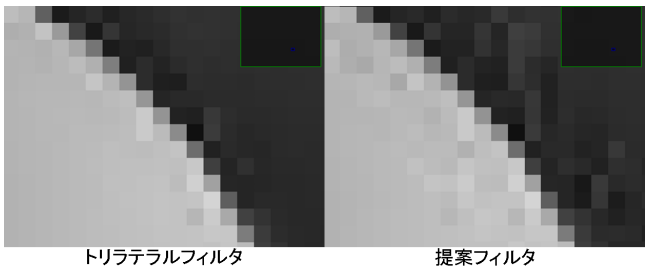


図 5.9: 各フィルタを適応後のエッジ領域拡大画像

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] Durand, F., and Dorsey, J. (2002),“Fast bilateral filtering for the display of high-dynamic-range images, ”ACM transactions on graphics (TOG), 21(3), pp.257-266.

[2] Fleishman, S., Drori, I., and Cohen-Or, D, (2003, July), “Bilateral mesh denoising, ”In ACM Transactions on Graphics (TOG) (Vol. 22, No. 3, pp. 950-953),ACM.

[3] Alcantarilla, P. F., Bartoli, A., and Davison, A. J. (2012), “KAZE features, ”In Computer Vision ECCV 2012 (pp. 214-227), Springer Berlin Heidelberg,.

[4] Alcantarilla, P. F., and Solutions, T, (2011), “Fast explicit diffusion for accelerated features in nonlinear scale spaces, ”IEEE Trans, Patt, Anal, Mach, Intell, 34(7), pp.1281-1298.

[5] 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.

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

[7] 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.

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

[9] 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)

[10] Weiss, B, (2006, July), “Fast median and bilateral filtering, ” In Acm Transactions on Graphics (TOG) (Vol. 25, No. 3, pp. 519-526), ACM.

[11] Zhang, B., and Allebach, J, P, (2008), “Adaptive bilateral filter for sharpness enhancement and noise removal, ”Image Processing, IEEE Transactions on, 17(5), pp.664-678.

[12] Canny, J, (1986), “A computational approach to edge detection, ” Pattern Analysis and Machine Intelligence, IEEE Transactions on, (6), pp.679-698.

[13] 藤吉弘亘, (2007), “Gradient ベースの特徴抽出-SIFT と HOG-, ” 情処学会研報, CVIM, 160, pp.211-224.

A フィルタプログラムコード

```
#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"

using namespace std;
using namespace cv;

Mat adaptivelateralFilter(Mat input, float sigmaC1, float sigmaC2, float sigmaC3, float Rp)
{
  Raw2D destImg;
  Raw2D fTheta;
  Raw3D minGradientStack, maxGradientStack;
  int level;
  float R;
  Raw2D* pSrcImg = new Raw2D;
  Gradients imgrad;
  Raw2D blur;

  //Mat to Raw
  pSrcImg->sizer(input.cols,input.rows);
  for(int y=0;y<input.rows+MAX_LEVEL*2;y++){
    for(int x=0;x<input.cols+MAX_LEVEL*2;x++){
  pSrcImg->put(x-MAX_LEVEL,y-MAX_LEVEL,input.data[(y-MAX_LEVEL)*input.step+(x-MAX_LEVEL)*input.elemSize()]);
    }
  }

  //set levels
  if(pSrcImg->getXsize()*pSrcImg->getYsize() < 400000) level = 4;
  else level = MAX_LEVEL;

  //allocate memory
  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);

  //compute image gradients
  imgrad.computeFullGradients(pSrcImg,&blur);
  R = pSrcImg->buildImageStack(&imgrad,&minGradientStack,&maxGradientStack,level);

  R *= Rp; //adjust find region parameter

  //find edge region
  fTheta.findAdaptiveRegion(&minGradientStack, &maxGradientStack, R, level);

  //filtering
```

```
55    destImg.Filtering(pSrcImg, &fTheta, sigmaC1, sigmaC2, sigmaC3,&imgrad)
          ;
56
57    //Raw to Mat
58    for(int y=0;y<input.rows;y++){
59      for(int x=0;x<input.cols;x++){
60        input.data[y*input.step+x*input.elemSize()] = (uchar)destImg.get(x
              ,y);
61      }
62    }
63
64    return input;
65  }
66
67
68  void Gradients::computeFullGradients(Raw2D *src,Raw2D *blur){
69    int xmax, ymax;
70    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}};
71    float sobel_x[3][3] = {{-1,0,1},{-2,0,2},{-1,0,1}};
72    float sobel_y[3][3] = {{-1,-2,-1},{0,0,0},{1,2,1}};
73
74    xmax=getXsize();
75    ymax=getYsize();
76
77    /***********************gauss_blur*********************************
          */
78    for(int y =0;y<ymax;y++){
79      for(int x=0;x<xmax;x++){
80        float tmp = 0;
81        float normFactor = 0;
82        for(int i=-2;i<3;i++){
83    for(int j=-2;j<3;j++){
84      if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
85        tmp += gauss_kernel[i+2][j+2]*src->y[xmax*(y+i)+(x+j)];
86        normFactor += gauss_kernel[i+2][j+2];
87      }
88    }
89        }
90        tmp /= normFactor;
91        blur->y[xmax*y+x] = tmp;
92      }
93    }
94    /*
          ***********************************************************************
          */
95
96
97    for(int y =0;y<ymax;y++){
98      for(int x=0;x<xmax;x++){
99        /***********************sobel*********************************
              */
100       float fx=0,fy=0;
101       for(int i=-1;i<2;i++){
102   for(int j=-1;j<2;j++){
103     if((y+i)>=0 && (y+i)<ymax && (x+j) >= 0 && (x+j) < xmax){
104       fx += sobel_x[i+1][j+1]*src->y[xmax*(y+i)+(x+j)];
105       fy += sobel_y[i+1][j+1]*src->y[xmax*(y+i)+(x+j)];
106     }
107   }
108       }
```

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

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

```
220   bool f = false;
221   int co=0;
222
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)] <
              MEDIAN_THRESHOLD && !(m==0 && n==0)){
228     f = true;
229     break;
230         }
231         else{
232     medi[co] = srcImg->y[i+m+imax*(j+n)];
233         }
234         co++;
235       }
236     }
237   }
238
239   //no noised pixel
240   if(f){
241     for(m = -DSFHS; m<=DSFHS; m++) {
242       for (n = -DSFHS; n<=DSFHS; n++) {
243         if( (i+m) >= 0 && (i+m) <imax && (j+n) >=0 && (j+n) < jmax) {
244     //distance
245     domainWeight = distW[m+MAX_LEVEL][n+MAX_LEVEL];
246
247     //brightness
248     brightWeight =  srcImg->y[i+imax*j] - srcImg->y[i+m+imax*(j+n)];
249     brightWeight = (float) pow(M_EXP, (double) (-(brightWeight*
            brightWeight)/(bufB)));
250     //gradient
251     float dx,dy,gradDiff;
252     dx = imgrad->gx[i+imax*j] - imgrad->gx[i+m+imax*(j+n)];
253     dy = imgrad->gy[i+imax*j] - imgrad->gy[i+m+imax*(j+n)];
254     gradDiff = sqrt(dx*dx+dy*dy);
255     gradWeight = (float) pow(M_EXP, (double) (-(gradDiff*gradDiff)/bufG)
            );
256
257     //convolution
258     tmp += srcImg->y[i+m+imax*(j+n)] * domainWeight * brightWeight *
            gradWeight;
259     normFactor +=  domainWeight * brightWeight * gradWeight;
260         }
261       }
262     }
263   }
264   //noised pixel
265   else{
266     sort(medi,medi+co);
267     normFactor = 1;
268     tmp = medi[4]; //center factor;
269   }
270       }
271       //easy smoothing
272       else{
273   for(m = -halfSize; m<=halfSize; m++) {
274     for (n = -halfSize; n<=halfSize; n++) {
275       if( (i+m) >= 0 && (i+m) <imax && (j+n) >=0 && (j+n) < jmax) {
276         //distance
```

```
277           domainWeight = distW[m+MAX_LEVEL][n+MAX_LEVEL];
278           tmp += srcImg->y[i+m+imax*(j+n)]* domainWeight;
279           normFactor +=  domainWeight;
280         }
281       }
282     }
283         }
284         if(normFactor != 0){
285     tmp /= normFactor;
286         }
287         else{
288     tmp = srcImg->y[i+imax*j];
289         }
290         put(i,j,tmp);
291       }
292     }
293   }
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
304     y = new float[ixsize*iysize]; // & allocate memory.
305     xsize = ixsize;       // set new image size,
306     ysize = iysize;
307   }
308
309   void Raw2D::sizer(Raw2D* src) {
310     int ix, iy;
311
312     ix = src->getXsize();
313     iy = src->getYsize();
314     sizer(ix,iy);
315   }
316
317   void Raw3D::sizer(int ixsize, int iysize, int izsize) {
318     int i;
319
320     z = new Raw2D[izsize];      // make room for the 2D objects,
321     zsize = izsize;
322     for(i=0; i< zsize; i++)
323       z[i].sizer(ixsize,iysize);  // use the Raw2D sizer.
324   }
325
326
327   void Gradients::sizer(int ixsize, int iysize) {
328
329     grad = new float[ixsize*iysize];  // & allocate memory.
330     gx = new float[ixsize*iysize];  // & allocate memory.
331     gy = new float[ixsize*iysize];  // & allocate memory.
332     xsize = ixsize;       // set new image size,
333     ysize = iysize;
```

```
}

void Gradients::sizer(Raw2D* src) {
  int ix, iy;

  ix = src->getXsize();
  iy = src->getYsize();
  sizer(ix,iy);
}
```

B フィルタヘッダーコード

```
#define M_EXP 2.71828182845904523536028747135 27
#define MAX_LEVEL 5 //maximam filter size
#define MEDIAN_THRESHOLD 10 //medianfilter's threshold
#define DSFHS 2 //Difficult Smoothing Filter Half Size
//=====================
// Forward Declarations
//=====================
class Raw2D;
class Raw3D;

class Gradients;

class Raw2D /*: public CObject*/ {
 public:        //-----------------DATA-----------------
  int xsize;    // # of pixels per scanline,
  int ysize;    // # of scanlines in this Raw2D.
  float *y;   // 1D array of float that are accessed as a 2D array.

 public:        //---------------init fcns-------------
      Raw2D(void){}   // constructor for 'empty' Raw2Ds
      ~Raw2D(void){}    // destructor; releases memory
    void sizer(int ixsize, int iysize);   // get mem for rectangle of
        pixels
    void sizer(Raw2D* src);          // get same amt. of mem as 'src'
  int getXsize(void) {return xsize;};   // get # pixels per scanline
  int getYsize(void) {return ysize;};   // get # of scanlines.
    void put(int ix, int iy, float val) { // write 'val' at location ix,
        iy.
  y[ix + xsize*iy] = val;
  };
    float get(int ix, int iy) { // read the value at ix,iy.
    return(y[ix + xsize*iy]);
  };
  float getXY(int ixy){   // read value at 1D address ixy
    return y[ixy];
  };
  void putXY(int ixy,float val){// write value at 1D address ixy
    y[ixy] = val;
  };

    void ComputeGradients(Raw2D* pX, Raw2D* pY);

    void findAdaptiveRegion(Raw3D* pMinStack, Raw3D* pMaxStack, float R,
         int levelMax);
```

```
42
43    void Filtering(Raw2D* srcImg,Raw2D* fTheta, float sigmaCTheta1, float
          sigmaCTheta2, float sigmaCTheta3,Gradients *imgrad);
44
45    float buildImageStack(Gradients *imgrad, Raw3D* pMinStack,
46                  Raw3D* pMaxStack , int level);
47
48  };
49
50  class Raw3D /*: public CObject*/ {
51  public:
52        Raw2D *z; // dynam. allocated space for a set of Raw2D objects.
53        int zsize;  // # of Raw2D objects stored.
54
55  public:
56        Raw3D(void){} // 'empty' Raw3D constructor.
57        ~Raw3D(void){}  // destructor.
58        void sizer(int ixsize, int iysize, int izsize); // reserve memory
59        void sizer(Raw3D* src);   // get same amt. of mem as 'src
60    int getZsize(void) {      // return # of Raw2D's we hold;
61      return(zsize);
62    };
63    int getYsize() {      // # of Raw1D's in zval-th Raw2D;
64      return(z[0].getYsize());
65    };
66    int getXsize(){       // # of pixels on yval,zval-th line
67      return(z[0].getXsize());
68    };
69        float get(int ix, int iy, int iz) {
70          return(z[iz].get(ix,iy)); // write 'val' at location ix,iy,iz.
71        };
72        void put(int ix, int iy, int iz, float val) {
73          z[iz].put(ix,iy,val); //write 'val' at location ix,iy,iz.
74        };
75    void wipecopy(Raw3D& src);  // copy, resize as needed.
76  };
77
78  class Gradients{
79   public:
80    int xsize;
81    int ysize;
82    float *gx;
83    float *gy;
84    float *grad;
85
86   public:
87        Gradients(void){}
88        ~Gradients(void){}
89    int getXsize(void) {return xsize;};   // get # pixels per scanline
90    int getYsize(void) {return ysize;};   // get # of scanlines.
91      void sizer(int ixsize, int iysize);   // get mem for rectangle of
            pixels
92      void sizer(Raw2D* src);         // get same amt. of mem as 'src'
93    void computeFullGradients(Raw2D *src,Raw2D *blur);
94
95  };
```