2008－CVIM－165 (34)
2008／11／28

近似LoGフィルタを用いた局所不変特徴量の抽出*
－ GPUによる実装 －

市村 直幸†

あらまし 局所不変特徴量を抽出する際，正規化LoGフィルタを用いて特徴点の固有スケールを求める．この処理には，数多くの畳み込み演算が必要なスケールスペースの生成を伴うため，計算時間に制約がある応用では，計算量の削減が必要となる．本論文では，この計算量の削減を目的とした近似LoGフィルタの，GPUによる実装について述べる．近似LoGフィルタでは，正規化LoG関数の極値点に位置する画素のみを参照することにより，計算量を削減する．このフィルタをCUDAを用いて実装し，計算速度の向上を図る．処理の一例として，720×480画素の画像から，初期スケールを1.6とし，各オクターブ5枚の画像をもつ5オクターブのスケールスペースを生成した．その結果，正規化LoGフィルタと比較して約2倍高速となり，計算時間は約14[ms]となった．スケールが大きくなるほど，近似LoGフィルタの優位性が高まることも確認した．局所不変特徴量の抽出に必要な他の処理もGPUにより実装したので，その結果も合わせて示す．

Extracting Local Invariant Features Using the Approximated LoG Filter
– A GPU-based Implementation –

Naoyuki ICHIMURA†

Abstract Detecting characteristic scales of feature points by the normalized LoG filter is used to extract local invariant features. Since large amount of computations for convolutions to create scale spaces are required, the computational cost in detecting characteristic scales has to be reduced for applications with time constraints. This paper presents a GPU-based implementation of an approximated LoG filter (ALoG filter). The response of the ALoG filter is calculated from the pixels corresponding to the extrema of the normalized LoG function to reduce the computational cost. We implement the filter using the CUDA for fast computation and create the scale space with 5 octaves, 5 images within an octave and the initial scale 1.6 from a 720×480 pixel image. For the scale space, the ALoG filter is about 2 times faster than the normalized LoG filter and the computational time is around 14[ms]. The ALoG filter has much more advantage in its suitability for computing with large scales. We implement other functions for extracting local invariant features based on the GPU and show their performance.

1 まえがき

画像の対応付けは，複数の画像間で共通部分を見出す処理であり，多視点画像処理や物体認識等の基本を成す．近年，対応付けのための画像特徴量として，局所不変特徴量が幅広く用いられている.[1, 2, 3, 4, 5, 6]．局所不変特徴量は，(1) 局所領域の設定，(2) 局所領域の画像特徴を表す記述子 (descriptor) の計算，の2段階の処理を通じて抽出される．これらの処理を，抽出される特徴量が，画像の幾何学的変換や輝度変化に対し不変になるように構成する．図1に局所領域の例を示す [7]．図中の円が，記述子を計算する局所領域を表す．このような局所領域内で計算された記述子の比較により，画像の対応付けを行う．

局所領域の設定において重要なことは，複数の画像間にスケール変化が生じたとしても，共通部分において物理的に同じ範囲をカバーする局所領域を設定すること，すなわち，スケール不変性を有した設定が可能なことである．スケール不変性を有した局所領域の設定を実現する1つの方法として，正規化LoG(Laplacian of Gaussian) 関数によるスケー

*本研究の一部は，科学研究費補助金 (基盤研究 (c))，課題番号 18500145 の助成の下で行われた．

†産業技術総合研究所 脳神経情報研究部門，Neuroscience Research Institute, National Institute of Advanced Industrial Science and Technology (AIST), nic@ni.aist.go.jp, http://staff.aist.go.jp/naoyuki.ichimura/

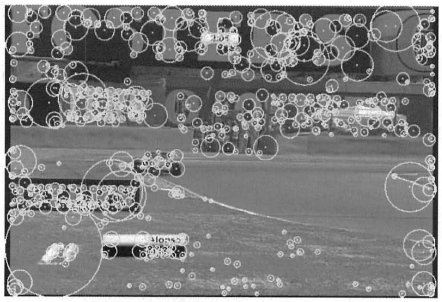


図 1: 局所領域の例．図中の円が，記述子を計算する局所領域を表す．局所領域の大きさは，中心に位置する特徴点の固有スケールにより決定される．

ルスペースの利用が挙げられる [8]．図 2に，スケールスペースの例を示す．この図では，5 つのスケールを有する正規化 LoG フィルタと，画像のダウンサンプリングを組み合わせ，スケールスペースを生成している．このスケールスペース内で極値検出を行い，極値点に特徴点を置く．極値点のスケールの値が，特徴点の固有スケール (characteristic scale) となる．その固有スケールに比例した大きさをもつ局所領域を，特徴点を中心として設定する．このように，固有スケールを局所領域の大きさの決定に利用することにより，スケール不変性が得られる．

上記のスケールスペースの生成には，入力画像とそのダウンサンプル結果に対して数多くの畳み込み演算を行う必要がある．よって，計算時間に制約がある応用では，計算量が問題になる場合がある．計算量を削減する 1 つの方法として，正規化 LoG 関数の原点に位置する画素の値，および，原点周辺の画素の値，この 2 つの値の差を用いる方法が Lepetit ら，Rosten らにより提案されている [9, 10, 11, 12]．この方法では，参照する画素数を減少させ，計算量を削減する．そのため，どの場所の画素を参照するか，つまり，フィルタサイズの決定が重要となる．しかし，Rosten らの研究 [10, 11] では，フィルタサイズは固定されており，スケールスペース生成への適用には言及されていない．また，Lepetit らは多重スケールの特徴点抽出を行っているが，フィルタサイズの決定方法が明確でない [9, 12]．また，これらの研究では，原点とその周辺の画素値の重み付けに関する考察がないという問題点もあった．

その問題点に対し，著者は，正規化 LoG 関数の極値点に位置する画素のみを参照する近似 LoG フィルタの提案を行っている [13]．近似 LoG フィルタでは，フィルタサイズは正規化 LoG 関数の極値点より明確に定まる．また，画素値の重み付けに関し

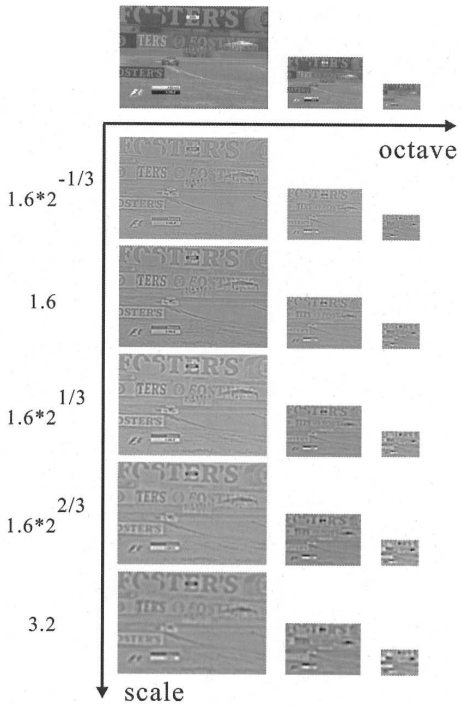


図 2: スケールスペースの例．5 つのスケールを有する正規化 LoG フィルタと，画像のダウンサンプリングを組み合わせ，スケールスペースを生成する．

ても，フィルタ応答を画素値の多項式で表現することにより実現されている．対応付けのための局所領域の設定において，近似 LoG フィルタが従来の方法と同等以上の性能を有していることを，実験により確認している．

本論文では，近似 LoG フィルタの GPU(Graphics Processing Unit) による実装について述べる．GPU はグラフィックスパイプラインを並列処理するために，多数のコアを有している．近年，それらのコアを，グラフィックスパイプライン以外の処理にも利用することが活発に行われている (例えば [14] 参照)．画像処理フィルタは，注目画素毎の並列化が可能であるため，複数のコアを用いた並列処理により高速化が期待できる典型的な例である．このことから，GPU による近似 LoG フィルタの実装は，計算時間短縮のために検討すべき手段の 1 つと言える．

開発環境として NVIDIA 社の CUDA[14] を用いて実装を行い，処理時間，GPU へのデータ転送の影響，CPU での処理時間との比較，正規化 LoG フィルタの処理時間との比較について述べる．具体的な処理の一例として，720×480 画素の画像から，初期スケールを 1.6 とし，各オクターブ 5 枚の画像をも

つ5オクターブのスケールスペースを生成した．その結果，正規化 LoG フィルタと比較して約 2 倍高速となり，計算時間は約 14[ms] となった．スケールが大きくなるほど，近似 LoG フィルタの優位性が高まることも確認している．局所不変特徴量の抽出に必要な他の処理も GPU により実装したので，その結果も合わせて示す．

2 近似 LoG フィルタ

本節では，まず，正規化 LoG 関数について述べる．そして，スケールスペースの生成に用いる近似 LoG フィルタの構成と，そのコーナー検出フィルタとの併用について述べる．

2.1 正規化 LoG 関数

正規化 LoG 関数は，次式で与えられる．

$$\sigma^2\nabla^2 g(u,v;\sigma) = -\left\{\left(1-\frac{u^2}{\sigma^2}\right)+\left(1-\frac{v^2}{\sigma^2}\right)\right\} g(u,v;\sigma) \quad (1)$$

ここで，関数 $g(u,v;\sigma)$ は，画像空間の変数 (u,v) とスケール σ を有する 2 変数の等方ガウス関数である．正規化 LoG 関数の極値と極値点は，次式で与えられる．

$$\text{極小値：} -\frac{1}{\pi\sigma^2},\ u=v=0 \quad (2)$$

$$\text{極大値：} \frac{e^{-2}}{\pi\sigma^2},\ u^2+v^2=4\sigma^2 \quad (3)$$

上式より，正規化 LoG 関数は，原点とその周囲の円上に極値を有することがわかる．図 3の濃淡は，フィルタリングに使用するオペレータとして正規化 LoG 関数を表現したものである．このようにコントラストの明確なオペレータを入力画像に畳み込んだ応答を用い，対応付けに必要な輝度変化が存在する部分に局所領域を設定する．

2.2 近似 LoG フィルタの構成

本論文で用いる近似 LoG フィルタ [13] では，正規化 LoG 関数の極値点に位置する画素のみを参照する．図 3にその概要を示す．式 (3) より，極大値は半径 2σ の円上に存在する．よって，近似 LoG フィルタの半径は 2σ となる．画素位置の量子化を考慮し，半径 2σ のフィルタの応答を，$n \le 2\sigma < n+1$ を満たす整数 n，$n+1$ を半径とする 2 つのフィルタを用いて計算する．中心画素の値を c とする．また，半径 n の円上における画素値の平均を $\bar{p}$ とする．半径 n の近似 LoG フィルタの応答 $ALoG(u,v;n)$ を，

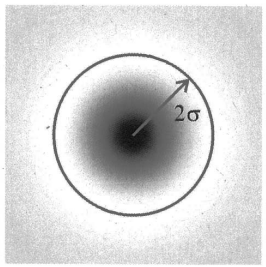


図 3: 近似 LoG フィルタ．極小値をとる中心画素と，極大値をとる円上の画素により，正規化 LoG フィルタの応答を近似する．フィルタの半径は 2σ となる．画素値とフィルタ応答の関係は，1 次多項式によりモデル化する．多項式の係数は，正規化 LoG フィルタの応答との誤差を評価関数とし，大量の事例画像を用いて決定する．

c, $\bar{p}$ の 1 次多項式を用いて計算する．

$$ALoG(u,v;n) = f_p(c,\bar{p};\boldsymbol{a}) \quad (4)$$

$$f_p(c,\bar{p};\boldsymbol{a}) = a_1 c + a_2\bar{p} + a_3 \quad (5)$$

$$\boldsymbol{a} = (a_1,a_2,a_3) \quad (6)$$

式 (4) を用いて，スケール σ を有する近似 LoG フィルタの応答 $ALoG(u,v;\sigma)$ を次式より求める．

$$\sigma^2\nabla^2 g(u,v;\sigma) \approx ALoG(u,v;\sigma) \quad (7)$$

$$ALoG(u,v;\sigma) = w_1 ALoG(u,v;n) + w_2 ALoG(u,v;n+1) \quad (8)$$

$$w_1 = (n+1)-2\sigma,\ w_2 = 2\sigma - n \quad (9)$$

この近似 LoG フィルタの計算量のオーダーは $O(n)$ であり，計算量のオーダーが $O(n^2)$ である 2 次元の畳み込み演算に比べ計算量が削減される．

式 (7) の近似式が成り立つように，式 (4) の係数 $\boldsymbol{a}$ を次式の評価関数に基づき決定する．

$$J = \sum_{\sigma,u,v} \left\| \sigma^2\nabla^2 g(u,v;\sigma) - ALoG(u,v;\sigma) \right\|^2 \quad (10)$$

この評価関数を計算するために，まず，入力画像に対しガウスフィルタを適用し，スケールスペースを生成する．スケールスペースの生成は，画像のダウンサンプリングとスケールの変化を組み合わせて行う (図 2 参照)[3]．そして，全てのスケールの全ての画素において，c, $\bar{p}$ を求める．一方で，正規化 LoG フィルタによりスケールスペースを生成する．そして，c, $\bar{p}$ および正規化 LoG フィルタの応答を用いて，式 (10) を最小にする係数 $\boldsymbol{a}$ を，線形方程式を解くことによって求める．

上記の係数計算を大量の事例画像に対し行い，近似 LoG フィルタを構成した [13]．係数計算に用いた

画像は，Caltech-256[15] より得た．合計 30607 枚の画像より得られた係数の平均値 $\bar{a}$ は，

$$\bar{a} = (-1.39, 1.41, -2.54) \tag{11}$$

であった．この値を使用した近似 LoG フィルタにより，スケールスペースを生成する．

2.3 コーナー検出フィルタとの併用

近似 LoG フィルタでスケールスペースを生成すると，シーンや画像解像度に依存するが，極値点数は数百から数万になることが多い．対応付けの処理量を減少させるため，それらの極値点の選択を必要とする場合がある．この選択のために，Trajkovic らのコーナー検出フィルタ [16] を適用する．中心画素 c の cornerness R_c を，次式で定義する．

$$R_c = \min_{p,p'} \left((p-c)^2 + (p'-c)^2 \right) \tag{12}$$

ここで，p, p' は，正規化 LoG 関数の極大値の円上で，中心画素 c に対して対称な位置にある 2 つの画素の値を表す．上式からわかるように，R_c は近似 LoG フィルタで参照する画素のみから計算できる．よって，近似 LoG フィルタは，コーナー検出フィルタと容易に併用可能である．この性質は，直線エッジに対する不要な応答を抑制するために有用である．極値点の選択は，式 (8) および式 (12) の値に対するしきい値処理により行う．

上記のようにして構成した近似 LoG フィルタは，同様に正規化 LoG フィルタ (Laplacian) とコーナー検出フィルタを併用している Harris-Laplacian フィルタおよび Hessian-Laplacian フィルタ [5] よりも，repeatability[17] に優れており，対応付けに適した局所領域の設定が可能なことを確認している [13]．次節では，近似 LoG フィルタの GPU による実装について述べる．

3 近似LoGフィルタのGPUによる実装

本節では，近似 LoG フィルタの GPU による実装について述べる．まず，execution configuration について説明を行う．その後に計算時間，GPU へのデータ転送の影響，CPU での計算時間との比較，正規化 LoG フィルタの計算時間との比較について述べる．

3.1 Execution Configuration

実装は NVIDIA 社の CUDA[14] を用いて行った．CUDA では，データをいくつかのブロックに分割する．そして，各ブロック内のデータに対し，いくつかのスレッドを付随させる．ブロックは，GPU 内に複数あるマルチプロセッサに割り当てられる．各マルチプロセッサは複数のコアを有しており，それらのコアでスレッドを並列実行する．

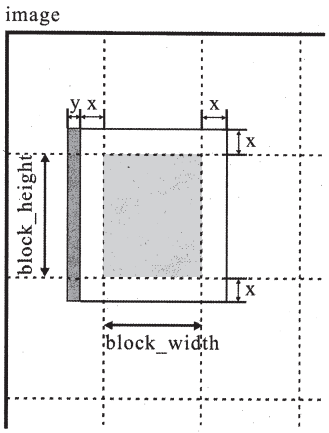


図 4: 画像処理フィルタの実装における execution configuration．まず，画像をいくつかのブロックに分割する．それらのブロックは，GPU 内に複数あるマルチプロセッサに割り当てられる．各マルチプロセッサは複数のコアを有しており，それらのコアで各ブロック内の画素に付随するスレッドを並列実行する．近似 LoG フィルタでは，block_width と block_height は共に 16 画素とした．付加するデータの幅 x と y は，フィルタのスケールにより定まる．よって，スケールによりスレッド数が変化する．

ブロックとスレッドの数は execution configuration と呼ばれ，GPU で実行する関数を呼ぶ場合に指定する．図 4に，画像処理フィルタにおける execution configuration の例を示す．指定すべき値は，ブロックの幅と高さ，および，各ブロック内でのスレッドの数である．

GPU で画像処理を行う場合，ホストコンピュータのメインメモリにある画像データを，GPU の global メモリに転送する必要がある．この global メモリのデータを，ブロックに分割する．そして，フィルタ処理を行う前に，各ブロックのデータを GPU の shared メモリにコピーする．これは，shared メモリがコアから高速アクセス可能であり，処理速度が向上するためである．このコピーの際，次の 2 点に注意した．1 点は，ブロック境界部分の画像処理を正しく行うため，フィルタサイズの半分の画素数だけ，ブロックの外側のデータもコピーすることである．図 4では，x がフィルタサイズの半分を示している．もう 1 点は，転送を行う領域の幅を，half warp size と呼ばれる数の倍数となるように調整することである．連続するメモリ領域の幅を調整する

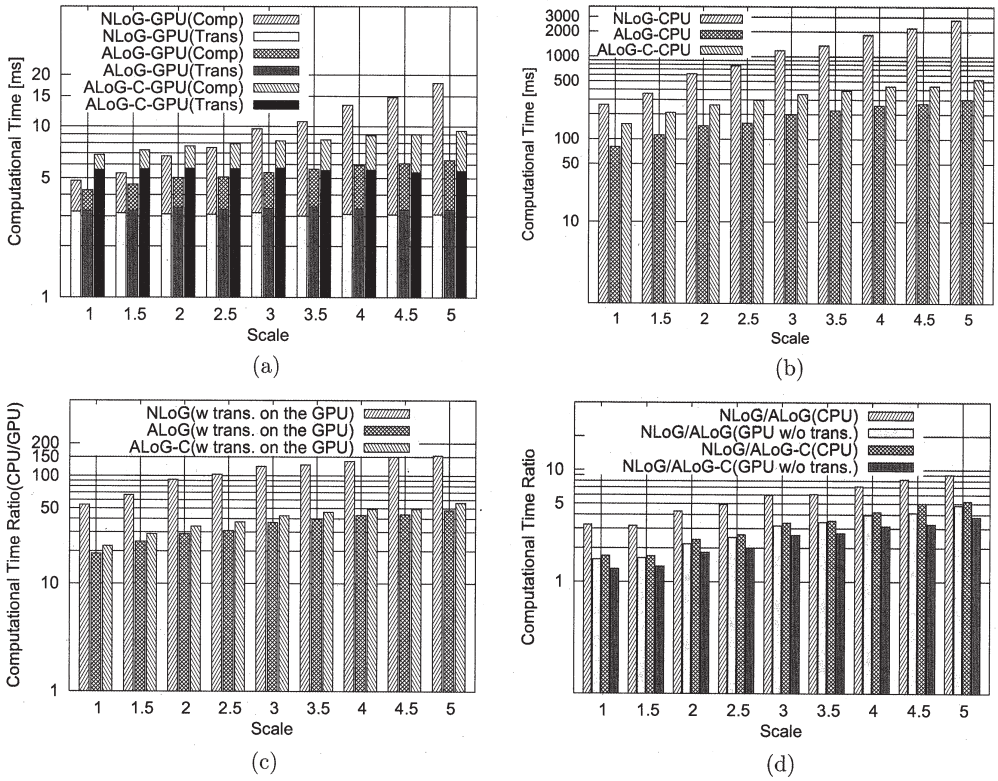


図 5: 近似 LoG フィルタの計算時間．処理した画像の解像度は 720 × 480 画素である．各グラフの横軸は，スケール σ を表す．(a) GPU による計算時間．この図で，NLoG は正規化 LoG フィルタを，ALoG は近似 LoG フィルタを，ALoG-C はコーナー検出フィルタを併用した近似 LoG フィルタを表す．また，Comp はフィルタの計算時間を，Trans はホストから GPU への画像データの転送時間と GPU からホストへの処理結果の転送時間の和を表す．(b) CPU による計算時間．(c) CPU と GPU の計算時間の比．データの転送時間を含む．(d) 正規化 LoG フィルタと近似 LoG フィルタの計算時間の比．データの転送時間は含まない．

ことにより，global メモリへのアクセスを効率良く行うこと (coalesced memory access を用いること) が，この調整を行う理由である [18]．このために，必要ならば，幅の調整のためのデータを加える．そのデータの幅は，図 4では y と示されている．このようにして，shared メモリにコピーされたデータに対し，スレッドを割り当てる．

近似 LoG フィルタの実装では，データが付加される前のブロックの幅と高さ，つまり図 4の block_width と block_height は，共に 16 画素とした．フィルタサイズはスケールにより変化するため，図 4の x と y は共にスケールにより変化する．よって，各ブロックの画素数も，スケールにより変化する．ここで，

$$\text{tile\_w} = \text{block\_width} + 2x + y \quad (13)$$

$$\text{tile\_h} = \text{block\_height} + 2x \quad (14)$$

とすると，各画素にスレッドを割り当てた場合，その数は tile_w × tile_h となる．しかし，1 ブロックでのスレッド数には上限があるため (例えば 512)，スケールによってはそれを越える場合もある．この点を考慮し，ブロック内部のデータを tile_w × 8 の大きさのサブブロックに分割した．そして，サブブロックの各画素にスレッドを割り当て，各スレッドで全サブブロックの同一位置にある画素に対し，逐次的に処理を行った．

3.2 処理結果

実装を行った計算機環境は，以下の通りである．

ホストコンピュータ: HP xw8600 Workstation
OS: Red Hat Enterprise Linux WS4
CPU: Intel Quadcore Xeon (3.16GHz/12MBL2)
メモリ: 8GB DDR2 FBD RAM
グラフィックスカード: NVIDIA Quadro FX4600,

および，GeForce GTX 280
ディスプレイへの表示はプライマリカードのQuadroで行い，セカンダリカードのGeForceは演算専用とした．このカードは240個のコアを有する．浮動小数点演算は，全て単精度で行った．

図5(a)にGPUによる計算時間を示す．処理した画像の解像度は720×480画素である．正規化LoGフィルタ(NLoG)，近似LoGフィルタ(ALoG)，および，コーナー検出フィルタを併用する近似LoGフィルタ(ALoG-C)の計算時間を示している．また，GPUでの計算時間(Comp)と，ホストとGPU間での画像データと処理結果の転送時間(Trans)を分けて表示している．横軸はフィルタのスケールσである．この図より，フィルタリングにおいて参照する画素数が少ないALoGおよびALoG-Cの計算時間は，NLoGの計算時間に比べて短く，かつ，スケールに対してより緩やかに増加することがわかる．ALoG-Cに関しては，処理結果として，式(8)の応答と式(12)のcornernessの2つを出力するので，GPUからホストへの処理結果の転送時間が他のフィルタより長くなる．しかし，計算時間はスケールが小さい場合でもNLoGより短くなっている．

図5(b),(c)に，各フィルタのCPUでの計算時間，および，CPUとGPUの計算時間の比を示す．スケールに対する計算時間の推移は，CPUでもGPUと同様の傾向を示し，ALoGおよびALoG-CがNLoGより優位と言える．しかし，GPUによる計算時間の短縮の度合に関してはNLoGが優れている．GPUの使用により，NLoGは対CPU比で50倍から150倍程度の速度向上がみられるのに対し，ALoGおよびALoG-Cでは20倍から50倍程度に留まる．これはALoGおよびALoG-Cでは，円上にある画素を参照するため，連続なメモリ領域へのアクセスが生じず，計算効率が低下することに起因していると考えられる．このため，図5(a)に示したように，スケールが小さい場合には，転送時間を含めて考えると，ALoG-CがNLoGより遅くなる場合が出てくる．しかし，図5(d)のNLoGとALoGおよびALoG-Cの計算時間の比に示すように，転送時間を含まない場合には，コーナー検出を行ったとしても，近似LoGフィルタは正規化LoGフィルタよりも全スケールにおいて計算時間で優位となる．

局所不変特徴量の抽出に必要な他の処理もGPUにより実装すれば，処理結果の転送は不要となる．よって，スケールが小さい場合にも，近似LoGフィルタの優位性を活かすことができる．そのような実装について，次節で述べる．

4 GPUを用いた局所不変特徴量の抽出

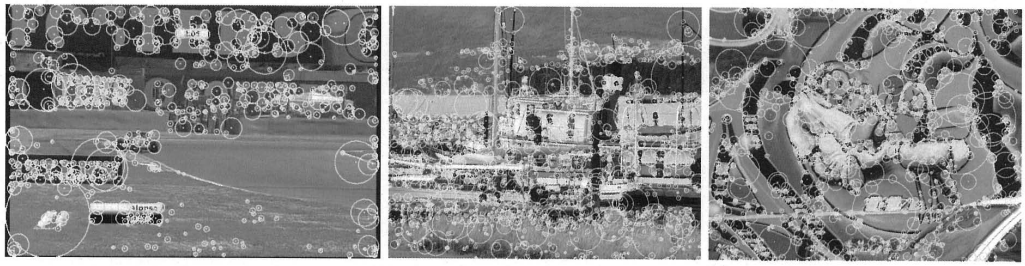
本節では，GPUを用いた局所不変特徴量の抽出について述べる．図6に，抽出アルゴリズムの擬似コードを示す．まず，入力画像imgをホストからGPUへ転送する．その後，カラー画像から輝度画像d_yiを求める．その画像から，スケールスペースを生成する．まず，オクターブ番号jに応じたダウンサンプルを行う．そして，その結果に，画像番号kに応じたスケールを有するガウスフィルタを適用する．ガウスフィルタの処理結果d_giに対して，画像番号kに応じたスケールを有する近似LoGフィルタとコーナー検出フィルタ，および，5×5の微分フィルタ[20]を適用する．近似LoGフィルタとコーナー検出フィルタのスケールスペースd_aiとd_ciを用い，その極値を各オクターブ毎に検出する．この検出結果である極値の位置を表す2次元のマップd_emapから，特徴点の位置と固有スケールをもつ特徴点リストd_fp_listを生成する．GPUでは，共通の変数である特徴点数に対して，全てのスレッド間で排他制御を行いつつ処理することができないため，この処理は逐次的に行う必要がある．よって，特徴点リストの生成はCPUで行う．このために，ホストとGPU間でのデータ転送が必要となり，計算時間を消費する．局所領域内の画像特徴を表す記述子として，SIFT記述子[3]を用いる．この記述子の計算には輝度勾配が必要なため，微分フィルタのスケールスペースd_iiを用い，dominant orientation d_dorと記述子d_desを計算する．

計算時間の計測に用いたのは，図7に示す3枚の画像である．スケールスペースの生成は，図2に示すように，初期スケールを1.6とする5つのスケールをもつフィルタとダウンサンプリングを組み合わせて行った．オクターブ数は，画像の長辺が32画素未満の長さになるまでダウンサンプルして決定した．ガウスフィルタのサイズは，ガウス関数の値が10^{-3}未満になる点で定義域を打ち切って決定した．また，計算時間の比較を行った正規化LoGフィルタでは，そのフィルタサイズをスケールの3倍とした．近似LoGフィルタの応答の絶対値，および，cornernessに対するしきい値は，それぞれ，10および100とした．特徴点リストを用いる処理では，リストを長さ16のブロックに分割し，並列処理を行った．dominant orientationおよび記述子を計算する際の局所領域の半径は，それぞれ，固有スケー

```
Algorithm : LocalInvariantFeatures(img, opt)
ImageTransfer(img, d_i)
YComponentGPU(d_i, d_yi)
for j ← 0 to NO_OCTAVE − 1
    do { DownSamplingGPU(d_yi, d_dyi[j])
         for k ← 0 to NO_IMAGE_OCTAVE − 1
             do { GaussFilterGPU(d_dyi[j], opt, d_gi[j][k])
                  ALoGFilterGPU(d_gi[j][k], opt, d_ai[j][k], d_ci[j][k])
                  ImageDerivativeFilterGPU(d_gi[j][k], d_ii[j][k])
ExtremaDetectionGPU(d_ai, d_ci, opt, d_emap)
FeaturePointList(d_emap, d_fp_list)
DominantOrientationGPU(d_fp_list, d_ii, opt, d_dor)
DescriptorGPU(d_fp_list, d_dor, d_ii, opt, d_des)
return (d_fp_list, d_dor, d_des)
```

[Notation]
- img: input image
- opt: options (e.g.,initial scale)
- d_i: input image on a device
- d_yi: grey scale image
- d_dyi: image pyramid
- d_gi: Gaussian scale space(SS)
- d_ai: ALoG SS
- d_ci: cornerness SS
- d_ii: image derivative SS
- d_emap: extrema map
- d_fp_list: feature point list
- d_dor: dominant orientations
- d_des: descriptors

図 6: 局所不変特徴量抽出アルゴリズムの擬似コード．変数のプレフィックス “d_” は，変数が GPU 上でアロケートされていることを示す．また，関数のサフィックス “GPU” は，関数が GPU で実行されることを表す．



(a) (b) (c)

図 7: 計算時間の計測に用いた画像．図中の円は，抽出された局所領域である．(a) F1 の放送映像 [7], F1_33. (b),(c) Interest Point Test Sequences の boat と graffiti[19].

ルの 2.5 倍と 7.5 倍とした．SIFT 記述子の次元数は 128 である [3].

表 1に，各処理に要する計算時間を示す．近似 LoG フィルタは，コーナー検出フィルタと併用した．この結果より，コーナー検出を行ったとしても，近似 LoG フィルタは正規化 LoG フィルタの約 2 倍の速度で処理が可能なことがわかる．短縮した計算時間は，約 3〜17[ms] である．GPU では，これらの時間は他のいくつかの処理を実行可能な長さであり，近似 LoG フィルタは処理全体の速度向上に寄与している．初期スケールを大きくした場合には，図 5の結果から，計算時間の短縮の度合はより大きくなる．よって，これらの結果は，近似 LoG フィルタの計算速度における優位性を明確にするものと言える．

画像 graffiti に対しては，解像度を変化させた場合の結果を示している．解像度の減少に伴い，処理全体に渡る計算時間の大きな短縮がみられる．短縮の度合が最も小さいのは，記述子の計算部分である．これは記述子の計算量は解像度ではなく，特徴点数と特徴点の固有スケールに依存するためである．解像度が低くても，特徴点が数多く検出されれば，記述子の計算量は大きくなる．また，特徴点数が少なくても，それらの固有スケールが大きい場合には，局所領域が大きくなり，処理すべき画素数が増加し，結果として計算量も増加する．このことから，処理全体の計算速度の向上を図るためには，記述子の計算を高速化する必要性が高いと考えられる．この高速化は，今後の課題の 1 つである．

5 むすび

本論文では，近似 LoG フィルタの GPU による実装について検討を行った．局所不変特徴量の抽出処理全体を GPU により実装した結果，近似 LoG フィルタの計算時間における優位性が確認できた．本論文と文献 [13] の結果から，近似 LoG フィルタ

表 1: 局所不変特徴量の抽出に必要な計算時間．単位は [ms]．100 回の処理の平均値を示す．総計算時間には，ここに示したタスク以外の処理，例えばメモリアロケーション等も含まれる．

image (#octave)	F1_33 (5)	boat (5)	graffiti (5)	graffiti (4)
Task\image size	720×480	800×640	800×640	320×240
image transfer	2.588	3.877	3.864	0.623
Y component	0.120	0.160	0.156	0.071
Down sampling	0.163	0.192	0.187	0.104
Gaussian filter	7.882	9.928	9.679	4.423
ALoG-C(NLoG) filter	13.625 (26.112)	19.302 (37.769)	19.261 (37.738)	3.959 (6.785)
Image derivative filter	4.332	5.984	5.963	1.444
Extrema detection	0.996	1.418	1.571	0.442
Feature point list	11.675	19.499	17.774	2.682
Dominant orientation	3.836	4.399	4.371	2.867
Descriptor	19.821	24.803	23.314	18.381
Total	68.136	93.334	89.715	37.134
#feature point	788	1914	1634	453

は，従来の特徴点抽出フィルタよりも，計算時間および repeatability において優位性を有するものと考えられる．今後も，局所不変特徴量の抽出とその対応付け全体に渡る処理の高速化に関して検討を行う．また，特徴量の応用に関しても，研究開発を進める予定である [21]．

参考文献

[1] C. Schmid and R. Mohr. Local greyvalue invariants for image retrieval. *IEEE Trans. PAMI*, Vol. 19, No. 5, pp. 530–535, 1997.

[2] M. Urban J. Matas, O. Chum and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *Proc. British Machine Vis. Conf.*, pp. 384–393, 2002.

[3] D. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comp. Vis.*, Vol. 60, No. 2, pp. 91–110, 2004.

[4] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *Int. J. Comp. Vis.*, Vol. 60, No. 1, pp. 63–86, 2004.

[5] C. Schmid A. Zisserman J. Matas F. Schaffalitzky T. Kadir K. Mikolajczyk, T. Tuytelaars and L. Van Gool. A comparison of affine region detectors. *Int. J. Comp. Vis.*, Vol. 65, No. 1/2, pp. 43–72, 2005.

[6] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Trans. PAMI*, Vol. 27, No. 10, pp. 1615–1630, 2005.

[7] 本研究では，SKY PerfecTV フジテレビ 721 において放送された映像を使用している．

[8] T. Lindeberg. Feature detection with automatic scale selection. *Int. J. Comp. Vis.*, Vol. 30, No. 2, pp. 79–116, 1998.

[9] V. Lepetit and P. Fua. Towards recognizing feature points using classification trees. Technical Report IC/2004/74, EPFL, 2004.

[10] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. In *Proc. Int. Conf. Comp. Vis.*, Vol. 2, pp. 1508–1515, 2005.

[11] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. European Conf. Comp. Vis.*, Vol. 1, pp. 430–443, 2006.

[12] V. Lepetit and P. Fua. Keypoint recognition using randomized tree. *IEEE Trans. PAMI*, Vol. 28, No. 9, pp. 1465–1479, 2006.

[13] 市村直幸. 正規化 LoG 関数の近似に基づく局所不変特徴量の抽出. 信学技報, No. PRMU2007-314, pp. 449–456, 2008.

[14] CUDA Zone: http://www.nvidia.co.jp/object/cuda_home_jp.html.

[15] Caltech-256: http://www.vision.caltech.edu/Image_Datasets/Caltech256/.

[16] M. Trajkovic and M. Hedley. Fast corner detection. *Image and Vision Computing*, Vol. 16, pp. 75–87, 1998.

[17] C. Schmid. Evaluation of interest point detectors. *Int. J. Comp. Vis.*, Vol. 37, No. 2, pp. 151–172, 2000.

[18] *NVIDIA CUDA Programming Guide, Version 2.0, Sec.5.1.2*, 2008.

[19] Interest Point Test Sequences: http://lear.inrialpes.fr/people/mikolajczyk/Database/.

[20] S. Ando. Consistent gradient operators. *IEEE Trans. PAMI*, Vol. 22, No. 3, pp. 252–265, 2000.

[21] N. Ichimura. Recognizing multiple billboard advertisements in videos. In *Proc. Pacific-Rim Symp. on Image and Video Technology*, pp. 463–473, 2006.