Fast Computation of Local Correlation Coefficients on Graphics Processing Units

Georgios Papamakarios[a], Georgios Rizos[a], Nikos P. Pitsianis[ab] and Xiaobai Sun[b]

[a]Dept. of Electrical and Computer Eng., Aristotle University, Thessaloniki, 54124, Greece
[b]Department of Computer Science, Duke University, Durham, NC, U.S.A.

ABSTRACT

This paper presents an acceleration method, using both algorithmic and architectural means, for fast calculation of local correlation coefficients, which is a basic image-based information processing step for template or pattern matching, image registration, motion or change detection and estimation, compensation of changes, or compression of representations, among other information processing objectives. For real-time applications, the complexity in arithmetic operations as well as in programming and memory access latency had been a divisive issue between the so-called correction-based methods and the Fourier domain methods. In the presented method, the complexity in calculating local correlation coefficients is reduced via equivalent reformulation that leads to efficient array operations or enables the use of multi-dimensional fast Fourier transforms, without losing or sacrificing local and non-linear changes or characteristics.

Keywords: fast convolution, normalized correlation coefficients, multi-core processors

1. INTRODUCTION

We present a fast computation method for two-dimensional convolutions and correlation coefficients (in the statistical sense) via exploiting both mathematical properties and modern computer architectures. Two-dimensional convolution has been a basic operation in image processing.[1] In recent years the convolution of a 2D image with a spatially-local filter is found at the central computation block in feature analysis, detection and matching, using algorithms based on the scale-invariant feature transform (SIFT).[2] The computation of the correlation coefficients between a 2D object template against a 2D image scene is frequently needed in object recognition or detection in cluttered scenes[3–5] or automated image registration and segmentation.[6] Rapid computation of such operations is demanded in numerous real-time applications across many scientific and engineering domains.[7–9]

We may describe the operation objects in the same way for the two basic operations. While mathematically symmetric, the two operand objects in a 2D convolution differ in size in the applications of our interest. We make the following convention. The first object is an $m \times n$ array of pixels with numerical values representing an image scene. We call it a *frame*. The second is a relatively smaller $p \times q$ array of numerical pixels representing a filter as in a convolution or a target object as in a correlation. We refer to it as a *template*. Each of the operations renders an array of numerical values representing a filtered image or a similarity score map, respectively. The output array size may range from $m \times n$ up to $(m + 2p) \times (n + 2q)$. All the array values are assumed real.

Rapid computation of the convolution is relatively better understood. The number of arithmetic operations for a convolution can be reduced when the template is sufficiently large, based on the convolution theorem and the use of the fast Fourier transform (FFT). Specifically, the number of the real arithmetic operations is the smaller one between $O(pq \times mn)$ by the direct evaluation method and $O((m + p)(n + q) \log((m + p)(n + q)))$ via the use of FFTs. Similar results in arithmetic complexity can be obtained easily for the computation of the correlation coefficients without local normalization, which is limited to applications where the changes across the frame are homogeneously linear. Recently, these results are successfully extended to the general case for computing correlation coefficients with local centering and normalization (LCCs), by a distinct reformation in the LCC expression[10] and a further algorithmic interpretation. The reformulation describes the LCCs in

Further author information: (Send correspondence to N. Pitsianis, X. Sun: E-mail: {nikos, xiaobai}@cs.duke.edu

Mathematics for Signal and Information Processing, edited by Mark S. Schmalz, Gerhard X. Ritter, Junior Barrera, Jaakko T. Astola, Franklin T. Luk, Proc. of SPIE Vol. 7444, 744412 · © 2009 SPIE
CCC code: 0277-786X/09/$18 · doi: 10.1117/12.834180

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

terms of a few phase correlations and element-wise array operations and therefore leads to a Fourier-domain algorithm for the LCC computation. Since the phase correlation is the convolution with a flip permutation of the template (or the frame), we may mention only one or the other at one place or another in the rest of the paper without redundant discussion on both.

Once the arithmetic complexity is reduced to the minimal, one resorts to available computer architecture means for further acceleration in order to meet real-time application demands. However, the algorithmic means and architectural means for acceleration are not independent of each other. We investigate their mutual impact on one another and their integral impact on the computational efficiency. As various multi-core processors exist and emerge, we experiment in particular with the graphics processing units (GPUs)[11–13] because they are widely available, affordable and offering relatively developed programming environment and easy integration with many kinds of host machines.

In the subsequent sections, we introduce the transformation of LCC expressions, the Fourier-domain algorithm for computing LCCs, and the implementation on GPUs. We present experimental results and conclude the paper with additional remarks.

2. LCC COMPUTATION VIA THE FFT

In this section, we review the correlation coefficients with local normalization and the reformulation of the LCC expression introduced originally in.[10] We then present the algorithm FLCC, in specific algorithmic arrangements for the first time, for computing the LCC in the Fourier domain and provide the analysis on its arithmetic complexity.

2.1 Local correlation coefficients

We introduce some notation and basic assumptions. Denote by T the target template with N_T pixels. The target template may represent an image feature drawn from an image feature bank or from a previous image in a temporal or spectral image series. Denote by S the frame, with N_S pixels, representing an image scene that is to be analyzed against the feature template. By assumption, both T and S are real-valued, the template is smaller than the frame, and hence $N_S > N_T$. Denote by P a typical sub-image (a panel) of S that is the same size as the template T.

We start with the correlation coefficient $c(P,T)$ between the template T and a single panel P. Assume momentarily that neither T nor P is constant-valued. We describe $c(P,T)$ in the following symmetric format:

$$c(P,T) = \frac{\mathrm{cov}(T,P)}{\sigma(T)\sigma(P)} = \frac{1}{N_T} \frac{\sum_{(x,y)}[P(x,y) - \mu(P)] \cdot [T(x,y) - \mu(T)]}{\sigma(T)\,\sigma(P)},$$

where $T(x,y)$ is the numerical value at (x,y) pixel of T, $\mu(T)$ and $\sigma(T)$ are the mean and standard deviation of T, respectively,

$$\mu(T) = \frac{1}{N_T}\sum_{x,y} T(x,y), \quad \sigma^2(T) = \frac{1}{N_T}\sum_{x,y}(T(x,y) - \mu(T))^2,$$

similarly with P, and $\mathrm{cov}(T,P)$ is the covariance between T and P. By the Cauchy-Schwartz inequality, $c(P,T) \in [-1,1]$. The coefficient is often interpreted as the cosine of the angle between P and T in the space of zero-mean and normalized panel vectors. When $c(P,T) = 0$, we may say the panel P is orthogonal to the template T. When $c(P,T) = 1$, P is a perfect match to T, pixel by pixel.

Consider the degenerate case that $\sigma(T)\sigma(P) = 0$, i.e., either the template or the panel is constant valued, or both. When the template T is constant valued, the matching problem becomes whether or not the panel is constant valued. This test can be done efficiently in some other way. We may therefore assume that T is not constant-valued and assume that T has zero mean and unit standard deviation (by appropriate translation in the mean value and normalization in the standard deviation),

$$\mu(T) = 0, \quad \sigma(T) = 1.$$

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

Unlike the template, a panel P from the scene S may be constant-valued. We express the correlation coefficient in the seemingly asymmetric form instead,

$$\bar{\sigma}(P)\, c(P,T) = \sum_{x,y} [P(x,y) - \mu(P)]\, T(x,y), \qquad \bar{\sigma}(P) = \frac{1}{\sqrt{N_T}} \sigma(P). \tag{1}$$

When the left-hand side vanishes, either the panel is found constant by $\sigma(P) = 0$, or else the correlation coefficient is zero, i.e., the panel is not a match to the template.

We are in a position to describe the field of the correlation coefficients of the template T with all the panels in the image scene S. With the reformulation in mind, we describe the LCC field as a whole entity in the form of a 2D array. We describe the traversal of the template T across the frame S in terms of its spatial translation, $T(x-u, y-v)$, where (u,v) is the location of a designated marker pixel of T in the pixel indices of S, $(x-u, y-v)$ specify the spatial positions of the template pixels relative to the marker pixel. The marker point can be arbitrarily chosen, such as the first pixel of the template at the north west corner. The panel P overlapping with the template at location (u,v) can be described as follows,

$$P(x,y;u,v) = S(x,y) B_T(x-u, y-v)$$

where B_T is the binary-valued characteristic function of the spatial support of the template. Substituting the spatially translated template and the overlapped panel into (1), we have the expression of the local correlation coefficients of T with all candidate panels

$$c(u,v)\, \bar{\sigma}(u,v) \;=\; \sum_{x,y} [P(x,y;u,v) - \mu(u,v)]\, T(x-u, y-v), \tag{2}$$

where (u,v) is a location in S, traversing across the entire frame of S, $\mu(u,v)$ and $\sigma(u,v) = \bar{\sigma}(u,v)/\sqrt{N_T}$ are the mean value and standard deviation of the (u,v) panel, and $c(u,v)$ is the correlation coefficient of the template with the associated panel. The LCC field $\{c(u,v)\}$ represents the collective information for subsequent location of best template matching, for example. It accounts for non-linear, non-rigid, and local changes in the scene.

The computation of the LCC field $c(u,v)$ with each and every panel translated in value to zero mean and normalized to 1 in standard deviation, as expressed in (2), is computationally expensive. It is not a phase correlation except when $\mu(u,v)$ and $\sigma(u,v)$ are constant across all the panels in the frame.[1] Various ad-hoc assumptions or heuristic approximations had been made on the properties of the image scene in an attempt to reduce the arithmetic complexity in evaluating the LCCs.

2.2 The reformulation

We describe the reformulation, as introduced in,[10] in expressing the LCCs in general, without distortion. The reformulation leads to an algorithm with most of the computation done in the Fourier domain.

We write the right-hand side of (2) in terms of three phase correlations,

$$\sum_{x,y} [P(x,y;u,v) - \mu(u,v)]\, T(x-u, y-v) \;=\; \{S * T\}(u,v) - \frac{1}{N_T} \{S * B_T\}(u,v) \cdot \{B_S * T\}(u,v), \tag{3}$$

with

$$\begin{aligned}
\{S * T\}(u,v) &= \sum_{x,y} S(x,y)\, T(x-u, y-v), \\
\{S * B_T\}(u,v) &= \sum_{x,y} S(x,y)\, B_T(x-u, y-v) = N_T\, \mu(u,v), \\
\{B_S * T\}(u,v) &= \sum_{x,y} B_S(x,y)\, T(x-u, y-v),
\end{aligned}$$

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

where B_X denotes the binary-valued characteristic function of the spatial support of X. We reformulate the expression of the standard deviation $\sigma^2(u,v)$ on the left side of (2) in terms of phase correlations as well,

$$N_T \cdot \sigma^2(u,v) = \{S^2 * B_T\}(u,v) - N_T \cdot \mu^2(u,v) = \sum_{x,y} S^2(x,y) B_T(x-u, y-v) - N_T \cdot \mu^2(u,v). \tag{4}$$

The reformulation in (3) and (4) describes the LCC computation in terms of four phase correlations and pixel-wise array operations among them, namely, pixel-wise scaling, shifting, division and square root extraction.

2.3 The FLCC algorithm

We explain how the reformulation in (3) and (4) leads to an algorithm with most of the computation in the Fourier domain. The evaluation of the phase correlations, dominating the LCC computation by the reformulation can be done in the Fourier domain as follows. Consider first the following five quantities in the Fourier domain,

$$F(B_T), \quad F(T), \quad F(B_S), \quad F(S), \quad F(S^2)$$

where $F(X)$ denotes the discrete Fourier transform of X. By the convolution/correlation theorem, the following four element-wise products in the Fourier domain

$$F(T) \cdot F(B_S), \qquad F(T) \cdot F(S), \qquad F(B_T) \cdot F(S), \qquad F(B_T) \cdot F(S^2)$$

correspond to the respective phase correlations via the inverse Fourier transforms.

The FLCC exploits the relationship between these quantities as well and utilizes further certain properties of the (discrete) Fourier transform. Denote by FFT and iFFT the fast forward and inverse Fourier transforms. We describe the special algorithmic arrangements in the use of FFTs in four stages,

A. FFTs of B_T, B_S and T and iFFT of $F(T) \cdot F(B_S)$;

B. FFT of the complex array $S + i \cdot S^2$ to obtain both $F(S)$ and $F(S^2)$;

C. iFFT of $F(B_T) \cdot F(S) + i \cdot F(B_T) \cdot F(S^2)$ to obtain both $S * B_T$ and $S^2 * B_T$;

D. iFFT of $F(S) \cdot F(T)$ to obtain $S * T$.

The data packing and unpacking in the last three stages are explained as follows. Stage B is based on the relationships between $\hat{X} = F(X)$, $\hat{Y} = F(Y)$ and $\hat{Z} = F(X + i \cdot Y)$ with real X and Y. Denote by I and J the identity operator and the reversal permutation except the leading element, respectively. Then, the relationships are as follows, in one dimensional expressions for convenience,

$$2\,\hat{X} = (I+J)\hat{Z}_{\mathrm{Re}} + i \cdot (I-J)\hat{Z}_{\mathrm{Im}}, \qquad 2\,\hat{Y} = (I+J)\hat{Z}_{\mathrm{Im}} - i \cdot (I-J)\hat{Z}_{\mathrm{Re}}.$$

The identity underlying stage C is

$$F^{-1}(X + i \cdot Y) = F^{-1}(X) + i \cdot F^{-1}(Y),$$

where $F^{-1}(X)$ and $F^{-1}(Y)$ are real. In Stage D, the number of operations in stage D can be reduced by a half if the real-valued property of $S * T$ is taken advantage of. Some existing software for FFTs provides such option.

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

2.4 The arithmetic complexity

We now provide an analysis on the arithmetic complexity of Algorithm FLCC in a common situation where a target template T is provided for object detection in a stream of image frames.

First of all, the number of arithmetic operations for the pixel-wise array operations, in and out of the Fourier domain, is linear in the frame size N_s. Next, the computation in the algorithm stage A is independent of any frame content and shall be done once for all the frames. Consequently, the total number of real arithmetic operations with the FLCC algorithm is about

$$12.5\, N \log(N) + \alpha\, N_S, \qquad N = N_T + N_S.$$

The scalar factor in the dominant term is based on the convention that the arithmetic complexity of the FFT in computational implementation is no more than $5N \log(N)$. The scalar α in the term linear in N_S is a modest constant, independent of data or data size, accounting for the scaling operations in the Fourier domain, the packing and unpacking operations before and after each fast Fourier transform, and the pixel-wise array operations in the image domain as described in Section 2.2.

The direct LCC evaluation by (1) takes $5N_S N_T + \beta\,(N_S + N_T)$ arithmetic operations, with a very modest scalar β. In comparison, the pairing of the template size N_T and the frame size N_S in the arithmetic complexity is multiplicative for the direct method and nearly additive for the FLCC algorithm.

3. CONVOLUTION AND FLCC ON GPUS

Further acceleration of convolutions and LCC computation for real-time image processing applications lies in advancing and utilizing computer technology and techniques. Many advanced computers today have multicore processors as the hardware architectures and support multi-threaded programming environments. We experiment with the graphics processing units (GPUs) in particular because they are widely available, affordable and integrate easily with desktop and laptop computers. GPUs by different vendors are equipped with different software environments to ease algorithm implementation and facilitate utilizing the parallel-processing hardware. We use the GPUs by NVIDIA for their relatively easy-to-use CUDA programming environment[14] and the CUFFT library providing the FFT functions.

3.1 Parallel implementation on GPUs

In the GPU implementation of the Fourier-domain algorithms for convolution and LCC computation we made extensive use of the library FFT routines. We are also facilitated in exploiting the data type at input or output for reduction in arithmetic operations. Specifically, if the data at the input or output of the FFT are real-valued, the number of arithmetic operations for the FFT is reduced by a half. We refer to these as half FFTs. For the LCC computation on GPUs, we use the FFTs as specified in the FLCC algorithm in Section 2.3. The element-wise array operations in the FLCC algorithm can be easily parallelized at any granularity level. We omit the detail in packing and unpacking the data arrays before and after the FFTs. The convolution via the FFT is much simpler. The convolution of a filter or template T with a scene frame S in a frame stream can be obtained by the use of a half FFT (from real to complex) and a half iFFT (from complex to real), assuming the Fourier transform of T is computed once for all or drawn from a template bank. In short, its takes two half FFTs per frame.

The parallelization of the direct methods is more involved. To ensure mutual exclusion in writing the results, the output data array is partitioned without overlapping. CUDA threads are grouped into CUDA blocks. Each thread can access data in three distinct memory spaces during execution: (i) a local memory private to the single thread, (ii) a shared memory visible to all threads in the same thread block, and (iii) the global memory to all threads. To utilize the memory structure, we partition the output array into sub-arrays. Each thread block is associated with a sub-array and responsible for the computation of the output sub-array. Every element in a sub-array is assigned to a designated thread. The template is copied to all blocks. At each thread block, a sub-array of the input frame that is needed by the output sub-array is copied from the global memory to the shared memory associated with the thread block. The input sub-array is larger than the output

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

sub-array along both dimensions. The margin in between is determined by the template size. The computation per thread for the LCC computation is of course different from that for the convolution.

The GPU implementation of the Fourier-domain methods is reported here first. The GPU implementation of the direct methods was by an earlier effort and detailed in.[15]

3.2 Experimental results

We present next experimental results on the NVIDIA Tesla C1060 GPU card, with 240 streaming processor cores running at 1.35GHz. The GPU has 4GB of dedicated memory and a 512-bit memory interface with a bandwidth of 102GB/sec. The host machine is a Dell T5400 workstation with two Quad Core Intel Xeon processors at 1.86GHz. We present two sets of experiments. The first demonstrates the architectural acceleration, by comparing the GPU computation to the CPU computation. The CPU-computation experiments were on the host machine with only one of its cores activated. The second set shows the algorithmic acceleration, by comparing the direct methods and the Fourier-methods on the GPU.

We illustrate the GPU acceleration in Table 1 for FFT-accelerated 2D convolution without assuming separability. The execution time is in milliseconds. On the CPU, the 2D convolution is calculated as two real-to-complex FFTs, followed by element-wise complex multiplication and a complex-to-real FFT using FFTW version 3.2.1. Experiments on the CPU in single-precision (`float`) and double-precision (`double`) floating point operations are reported.

One observes that, among the comparable cases, the computation on the GPU is 45-50 times faster that on the CPU, or 60-70 times faster if we take into account the difference in frequency. This is consistent with that for the algorithms addressed in this paper. We shall mention that the NVIDIA SDK provides indeed an FFT-accelerated 2D convolution as a demo. However, it is 50% slower for real-valued convolution.

Table 1. Execution time (msec) for FFT-accelerated 2D correlation

Data Size	Xeon 1 core @ 1.86GHz		C1060 @ 1.35 GHz
	`double`	`float`	`float`
$2K \times 2K$	931	674	15
$4K \times 4K$	4,223	3,210	63

The experiments of the second set are summarized in Figure 1. The plot on the left is for the 2D convolution on the GPU and the plot on the right is for the LCC computation. Each plot shows the comparison between the direct method and the FFT-acceleration as the template size varies. The experimental results with $2K \times 2K$ frame are plotted in blue lines; with $4K \times 4K$ in red. The arithmetic complexity and execution of the Fourier-domain method for the convolution, or the LCC computation, depends on the additive size of the template and the frame and therefore changes slowly with the increase in the template size, in comparison to that with the respective direct method. In the plot on the right for the LCC computation with the FLCC algorithm, for each of the two frame sizes, we give the time results in two different situations. In the first situation, the LCC computation includes the operations in stage A as described in Subsection 2.3. In total 4.5 FFTs are used. In the second, the same template is used on a stream of frames, the quantities in stage A are precomputed. Therefore, only 2.5 FFTs are used per frame. In each case, when the template size increases beyond a certain point, the Fourier-domain method is indeed faster. The second situation is more common in image processing applications, which motivated the design and development of the FLCC algorithm in the first place.

4. CONCLUDING REMARKS

We discuss first about the cross-over point in the comparison between the direct method and the Fourier-domain method, for the LCC computation as well as for the convolution. For each of the operations, the cross-over point on the GPU is higher than that in the arithmetic complexity. In fact, the up shift in the cross-over point for the convolution is mainly due to that in the discrete Fourier transform (DFT) between the FFT and the direct DFT. There are two primary factors contributing to this shift in the cross-over point. One is how the parallel processing architecture with complex memory hierarchy is utilized by the implementation in the

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

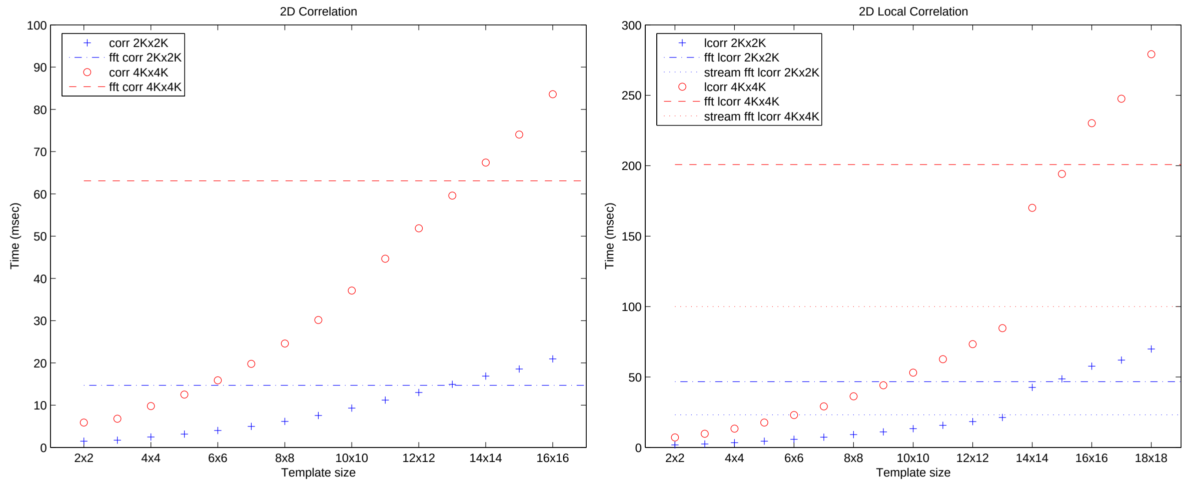

Figure 1. 2D Correlation on the left and LCC computation on the right; the direct and the Fourier-domain methods for each of the operations; data range with two frame sizes and multiple template sizes.

provided programming environment. The other is the effect of the architectural constraints on a particular algorithm. Such a shift may be reduced to some extent. Its existence, however, is a manifest of the mutual constraints between a parallel architecture like the GPU and an algorithm with a low ratio between arithmetic operations and memory accesses, such as the FFT.

We remark next on the ratio between the template size and the frame size and its impact. There are situations in image processing applications where the frames to be analyzed against the same template are sub-regions of a large image and the frame size is not much larger than the template size. For this case, the direct method for the LCC computation is not expected to change much in performance; the FLCC algorithm is expected to be more advantageous.

Finally we would like to comment on the connection and disconnection in algorithm design and development between the convolution and the LCC computation. The reformulation in Subsection 2.2 relates the LCC computation to the phase correlations. We take a further step in developing the FLCC algorithm in Subsection 2.3. Instead of using the phase correlation as the primitive operation for the LCC computation, we exploit the relationship among the phase correlations and utilize the properties of the Fourier transform. This leads to a substantial reduction in arithmetic complexity.

Acknowledgment

This work is supported in part by DARPA/MTO/DESA program.

REFERENCES

[1] Kuglin, C. D. and Hines, D. C., "The phase correlation image alignment method," in [*Proc. IEEE 1975 Int. Conf. Cybernet. Society*], 163–165 (1975).

[2] Lowe, D., "Object recognition from local scale-invariant features," in [*Proc. IEEE Int. Conf. Comp. Vision*], **2**, 1150–1157 (1999).

[3] Chen, J. Y. and Reed, I. S., "A detection algorithm for optical targets in clutter," *IEEE Trans. Aero Elec Sys* **AES-23**, 46–59 (1987).

[4] Johnson, A. E. and Hebert, M., "Efficient multiple model recognition in cluttered 3-d scenes," *Proc. IEEE Conf. Computer Vision and Pattern Recognition* , 671–677 (1998).

[5] Allney, S. and Morandi, C., "Digital image registration using projections," *IEEE Trans. Pattern Anal. Machine Intell.* **PAMI-8**, 222–233 (Mar 1986).

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms

[6] Johnson, G. A., Ali-Sharief, A., Badea, A., J. Brandenburg, G. C., Fubara, B., Gewalt, S., Hedlund, L. W., and Upchurch, L., "High-throughput morphologic phenotyping of the mouse brain with magnetic resonance histology," *Neuroimage* **37**(1), 82–89 (2007).

[7] Riesenhuber, M. and Poggio, T., "Hierarchical models of object recognition in cortex," *Nature Neuroscience* **2**(11), 1019–1025 (1999).

[8] Giese, M. A. and Poggio, T., "Neural mechanisms for the recognition of biological movements," *Nat Rev Neurosci* **4**(3), 179–192 (2003).

[9] Serre, T., Wolf, L., Bileschi, S., Riesenhuber, M., and Poggio, T., "Robust object recognition with cortex-like mechanisms," *IEEE Transactions on Pattern Analysis and Machine Intelligence* **29** (March 2007).

[10] Sun, X., Pitsianis, N. P., and Bientinesi, P., "Fast computation of local correlation coefficients," *Advanced Signal Processing Algorithms, Architectures, and Implementations XVIII* **7074**, 707405 (2008).

[11] Krüger, J. and Westermann, R., "Linear algebra operators for GPU implementation of numerical algorithms," in [*ACM SIGGRAPH*], 908–916 (2003).

[12] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P., "Brook for GPUs: stream computing on graphics hardware," in [*ACM SIGGRAPH*], 777–786 (2004).

[13] Nickolls, J., Buck, I., Garland, M., and Skadron, K., "Scalable parallel programming with CUDA," *ACM Queue* **6**(2), 40–53 (2008).

[14] *NVIDIA CUDA Programming Guide*, 2.2.1 ed. (May 2009).

[15] Karafyllias, I. and Manolis, I., *The computation of local correlation coefficients of images on graphics processing units*, dissertation for diploma (in Greek), Aristotle University of Thessaloniki (Apr. 2009).

Downloaded from SPIE Digital Library on 03 Jan 2010 to 152.3.136.133. Terms of Use: http://spiedl.org/terms