Fast Computation of Local Correlation Coefficients

Xiaobai Sun,[a] Nikos P. Pitsianis[ab] and Paolo Bientinesi[c]

[a] Department of Computer Science, Duke University, Durham, NC 27708 USA
[b] Electrical and Computer Engineering, Aristotle University, 54124 Thessaloniki, Greece
[c] AICES, RWTH-Aachen University, 52074 Aachen, Germany

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. The computation time is further reduced by utilizing modern multi-core architectures, such as the Sony-Toshiba-IBM Cell processor, with high processing speed and low power consumption.

Keyword list: local correlation coefficients, multi-dimensional fast Fourier transform, multi-core processors

1. INTRODUCTION

Computing local correlation coefficients is a basic step in various image-based data or information processing applications, including template or pattern matching, detection and estimation of motion or some other change in an image frame series, image registration from data collected at different times, projections, perspectives or with different acquisition modalities, and compression across multiple image frames.[1–8] Figure 1 depicts changes across image frames to be detected or estimated for correction, compensation or compression purposes. Local correlation coefficients (LCCs), with local normalization in standard deviation (or Euclidean length) after centering the local mean at zero, capture local, non-rigid, and non-linear changes to a greater extent than linear-transform based processing methods. However, the desirable real-time use of local correlation coefficients had been limited by the computation complexity in terms of arithmetic operation account, structures in software or/and hardware implementations, and memory latency.

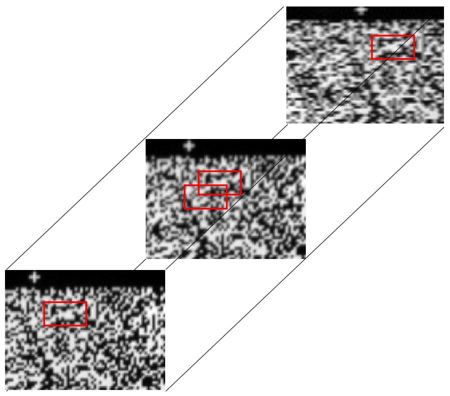

Figure 1. Changes across frames in an image stream.

Send correspondence to Nikos.P.Pitsianis@Duke.edu

This paper presents a method of accelerating the LCC computation, using both algorithmic and architectural means, for real-time applications or large-scale computational simulations. In the algorithmic aspect, we introduce an equivalent reformulation of local correlation coefficients that leads to array operations, which can be efficiently implemented in both software and hardware, and, more importantly, enables the use of multi-dimensional fast Fourier transforms (FFT), which was previously thought infeasible without sacrificing the local normalization. The expression of local correlation coefficients and its reformulation for reducing computation complexity are described in Section 2.

In the architectural aspect, we utilize the advanced features of emerging architectures in high processing speed, and low power consumption. We use particularly the Sony-Toshiba-IBM Cell broadband engine (the Cell) for a prototype study. In the context of image-based data or information processing, we implement distributed two-dimensional (2D) or three-dimensional (3D) FFTs in succession. We describe in Section 3 the Cell architecture, a distributed algorithm for multi-dimensional FFTs and a streaming strategy. By both model-based estimates and experimental results, the FFTs on the Cell can accelerate the LCC computation by two orders of magnitude in comparison to the performance of the same algorithm in MATLAB on a 3.4 GHz Pentium 4 workstation. Together, the algorithmic and architectural means make it practical to compute LCCs faithfully in real time. Section 4 concludes the paper.

2. LOCAL CORRELATIONS VIA THE FFT

We present in this section a method for rapid computation of the local correlation coefficients between a template and a scene image via the use of the fast Fourier transform (FFT). The computation efficiency had been a divisive issue between the so-called local correlation methods and the Fourier domain methods employed for template matching. We describe the method for the case with two-dimensional (2D) spatial translation, which can be extended to the case with rotational translation or scaling[9] and to higher-dimensional cases.

2.1 Local correlation coefficients

We review and re-write the formal expression of the local correlation coefficients, which provide the basic information, geometrically and statistically, for the subsequent location of best template matching under certain criteria. Denote by T the template or pattern, which may be drawn from an image library or from a previous image frame in a temporal or spectral series. Let N_T be the number of pixels in T. Denote by S a scene image, which may be a region of interest in the current frame or the entire frame. Assume that S has larger support than T in the sense that T can overlap with sub-panels of S. Let N_S be the number of pixels in S. We assume also that both T and S are real valued.

Consider first a typical correlation coefficient between T and S. Let P be a panel of S that the template T overlaps with. Assume for the moment that neither T nor P is constant-valued. We describe the correlation coefficient between the template T and the panel P in the following symmetric format,

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

where $\operatorname{cov}(T,P)$ is the covariance between T and P, $\mu(T)$ and $\mu(P)$ are the respective mean values of T and P, and $\sigma(T)>0$ and $\sigma(P)>0$ are the respective standard deviations,

$$\mu(T) = \frac{1}{N_T}\sum_{x,y} T(x,y), \qquad \mu(P) = \frac{1}{N_T}\sum_{x,y} P(x,y),$$

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

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 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. We may therefore keep the assumption that T is not constant-valued and assume further that T has zero mean and unit standard deviation (by appropriate translation in the mean value and normalization by the standard deviation),

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

Taking into account that a panel P of S may be constant-valued or a constant-valued panel may be searched for, we express the correlation coefficient in the seemingly asymmetric form instead,

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

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 correlation coefficients of the template with all the candidate panels of S. We specify the overlap location of the template within S in terms of spatial translation, $T(x - u, y - v)$, where (u, v) is the location of a specified marker pixel of T in the pixel indices of S. The marker point can be arbitrarily chosen, such as the first pixel of the template at the north west corner or the center pixel if it is not ambiguous. Accordingly, the overlapped panel 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 $\mu(u, v)$ 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 panel. The LCC distribution $c(u, v)$ represents the collective information for subsequent location of best template matching. It is linear in centered and normalized panels, but nonlinear in non-normalized panels. In other words, it accounts for non-linear, non-rigid, and local changes in the scene.

The computation of the LCC distribution $c(u, v)$ with each and every panel translated in value to zero mean and normalized to 1 in standard deviation, as expressed in (2), had been considered computationally expensive, especially, in comparison to the Fourier domain methods such as phase correlation method,[1] where the fast Fourier transform is used directly. Various changes have been made in an attempt to reduce the complexity in arithmetic operations. For example, the number of the local correlation coefficients is reduced by limiting or truncating the translation range. Or, the mean and standard deviation local to each panel are replaced with the mean and standard deviation of the scene S. In this case it is obvious that the computation can be accelerated via the use of the FFT. The reformulation and the resulting coefficients, however, no longer account for non-linear, non-rigid and local changes. A farther simplification or replacement of the LCC expression may be made for fast computation in the spatial domain directly, with perhaps an alternative interpretation.

2.2 Efficient computation with 2.5 FFTs

We describe in this section a method for efficient computation of the LCC distribution, without distortion or severe truncation, via the use of the FFT. In this method, with a given template T, only two and a half 2D FFTs are used for computing the LLC distribution, faithfully, for each scene image S.

We write the right-hand side of (2) into three cross-correlation components,

$$\begin{aligned}
\sum_{x,y}[P(x,y|u,v)-\mu(u,v)]\,T(x-u,y-v). &= \{S*T\}(u,v)-\frac{1}{N_T}\{S*B_T\}(u,v)\cdot\{B_S*T\}(u,v),\\
\{S*T\}(u,v) &= \sum_{x,y} S(x,y)\,T(x-u,y-v),\\
\{S*B_T\}(u,v) &= \sum_{x,y} S(x,y)\,B_T(x-u,y-v)=N_T\,\mu(u,v),\\
(B_S*T)(u,v) &= \sum_{x,y} B_S(x,y)\,T(x-u,y-v),
\end{aligned}$$

where B_S is the binary-valued characteristic function of the support of S. We re-write the expression of the standard deviation distribution $\sigma^2(u,v)$ on the left side of (2) into cross-correlation terms as well,

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

The reformation above leads to efficient array operations when the template is small and enables the use of multi-dimensional FFTs otherwise. We omit detailed discussion on the cross-over point.

We elaborate on how to use only two and a half FFTs in computing the LLC distribution per scene. By the cross-correlation theorem, the four cross-correlation components can be computed via the following five quantities in the Fourier domain

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

and the inverse FFT of the following four point-wise products,

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

For a sequence of scene images, the quantities associated with B_T, B_S and T only shall be computed once for all. Thus, for each scene image, we can use one FFT with $S+i\cdot S^2$ to obtain both $F(S)$ and $F(S^2)$ at the same time, then use one inverse FFT from the scaled quantities $F(B_T)\cdot F(S)$ and $F(B_T)\cdot F(S^2)$ to obtain two cross-correlation components $S*B_T+iS^2*B_T$, and use another inverse FFT to obtain $S*T$ from the scaled quantity $F(S)\cdot F(T)$ in the Fourier domain. Exploiting the real-valued property of $S*T$, the operations in the last inverse FFT can be reduced to a half.

In total, the number of real arithmetic operations in computing the LCC distribution per scene is about $12.5N\log(N)+\alpha N$ with $N=N_T+N_S-1$. The dominant term is based on the well known fact that the arithmetic complexity of the FFT is no more than $5N\log(N)$. The constant α in the linear term is a modest number, accounting for the scaling operations in the Fourier domain and the packing and unpacking operations before and after each FFT.

3. STREAMING MULTI-DIMENSIONAL FFTS ON THE CELL

We describe in this section the architectural approach for accelerating the computation of local correlation coefficients. In the prototype study, we use in particular the Sony-Toshiba-IBM broadband engine (the Cell). We focus on two-dimensional (2D) and three-dimensional (3D) FFTs in a stream. The stream may be along temporal, scale or spectral dimension. The key and dominant computation task in the method described in Section 2 is a stream of FFTs. There is an additional reason for this prototype study. In conventional approach, a multi-dimensional FFT is computed stand alone, one dimension at a time, and with slower performance than the 1D FFT.

The approach introduced in this section is different in the way each and every multi-dimensional FFT is parallelized and in the way a stream of FFTs is handled. By exploiting both the mathematical structures in a multi-dimensional FFT and the architectural features of the Cell, we have made 2D and 3D FFTs on the Cell reach highly competitive performance in comparison to the 1D FFT performance. Furthermore, we increase substantially the overall FFT throughput rate. In the rest of the section, we describe first the Cell architecture, then the distributed computation of 2D or 3D FFT on the Cell, exploiting the parallelism within each FFT, finally the streaming strategy exploring a different type of parallelism across multiple FFTs in succession.

3.1 The Cell architecture

We give a very brief description of the architectural features that we utilize for streaming multi-dimensional FFTs. The Cell is a multi-core processor with high computing performance at low power consumption rate. It was originally designed for gaming or multimedia applications. It has one Power Processing Element (PPE) and eight Synergistic Processing Elements (SPEs) connected by a broadband interconnect bus (EIB). Parallelism is supported at different granularity levels. The peak performance of a single SPE is 25.6 giga floating arithmetic operations per second (GFLOPs) in single precision. This is theoretically estimated and experimentally confirmed. Every SPE has 128 local 128-bit registers and a local store of 256KB, which is for both instructions and data. Most of the instructions can process 128-bit operands in single-instruction-multiple-data (SIMD) mode; in particular, single-precision floating-point operands, or 32-bit integer operands, are processed as 4-way SIMD vectors, fully pipelined. The SPEs can issue direct memory access (DMA) commands for explicit memory accesses across the local stores and to or from the main memory. Data transfers between any local store and the main memory can be handled by the memory interface controller (MIC). The DMA controllers and the MIC are all connected to the EIB and operate independently from the SPEs, providing parallelism between computation and data communication. The EIB supports a peak bandwidth of 204.8 Gbytes/s for data transfers among the PPE, the SPEs and the MIC, while the MIC provides a peak bandwidth of 25.6 Gbytes/s to and from main memory. Further details about the Cell architecture can be found in reference.[10] For each 2D or 3D FFT, we focus on the level of distributed memory associated with 8 SPEs. For streaming FFTs, we utilize the EIB bandwidth.

3.2 Distributed computation of multi-dimensional FFTs

We introduce a distributed algorithm for 2D or 3D FFTs, exploring and exploiting the partial ordering, and hence parallelism, among the transforms along different dimensions and utilizing the distributed Local Store (LS) memories on the SPEs and the DMA on the Cell.

An m-dimensional discrete Fourier transform (DFT) can be expressed as follows

$$v = (F_{n_m} \otimes \cdots \otimes F_{n_1})\, u = \prod_{d=1:m} \left(I_{n/(n_d s_d)} \otimes F_{n_d} \otimes I_{s_d} \right) \, u \,, \qquad n = n_1 \cdots n_m, \tag{3}$$

where $\otimes$ denotes Kronecker product and F_{n_d} is the 1D DFT of length n_d applied to the data array, initially u, along the d-th dimension, with stride $s_d = \prod_{j=1}^{d-1} n_j$, $2 \le d \le m$. The ordering among the directional transforms is mathematically arbitrary.

The 1D DFT along each dimension can be computed with a fast Fourier transform (FFT) algorithm, which is based on recursive factorization into sparse transforms. For example, the Cooley-Tuckey[11] algorithm can be described by the following particular factorization,

$$v = F_n u = (F_q \otimes I_p) D_{q,p} (I_q \otimes F_p) P_{n,q} u \ , \tag{4}$$

wherein the factors are applied to the data from right to left. First, $P_{n,q}$ permutes u with stride q. Next, $I_q \otimes F_p$ specifies the independent application of F_p to q vector segments of length p, with I_k as the identity matrix of order k. Then, the data are scaled by the diagonal matrix $D_{n,p}$, the diagonal elements are referred to as the twiddle scaling coefficients. Finally, $F_q \otimes I_p$ is a transform with a different stride. The factorization can be transposed because the DFT matrix is complex symmetric.

The factorization of 1D FFT may be interpreted as reshaping the 1D data u into 2D array and transforming the 2D data along each split dimension, with the twiddle scaling in between. In other words, the two dimensional transforms in (4) follow a total ordering, while the multiple dimensional transforms in (3) do not have to follow such a total ordering.

In the distributed algorithm for an m-dimensional FFT, we merge the partial ordering among dimensional DFTs and the factorization ordering within each dimensional FFT, enhancing the parallelism between arithmetic computation at each and every SPE and data exchange among different SPEs. Table 1 demonstrates in detail the communication and computation steps required by a distributed algorithm for 8 SPEs for the computation of a 3D FFT of size $n_1 \times n_2 \times n_3$. In detailed description, Step 0 requires the use of the DMA for contiguous

Table 1. Algorithm description.

```
**Distributed Algorithm** for a $n_1 \times n_2 \times n_3$ FFT with 8 SPEs.

  0. Data array partition and placement. $P_{\text{in}} = P_{8,4}(I_2 \otimes P_{4,2})$.
       $Y_i := X\left(1{:}n_1,\ 1{:}n_2,\ P_{\text{in}}(i) \times \frac{n_3}{8} + [1{:}\frac{n_3}{8}]\right),$          $i = 0, \ldots, 7.$

  1. Data exchange, butterfly and twiddle operations.
       Do in parallel:
            $Y_i := Y_i + Y_{i+1};$                    $i \in \{0, 2, 4, 6\}.$
            $Y_i := (Y_{i-1} - Y_i)d_i;$               $i \in \{1, 3, 5, 7\}.$
       Do in parallel:
            $Y_i := (Y_i + Y_{i+2})w_i;$               $i \in \{0, 1, 4, 5\}.$
            $Y_i := (Y_{i-2} - Y_i)w_i;$               $i \in \{2, 3, 6, 7\}.$
       Do in parallel:
            $Y_i := (Y_i + Y_{i+4})W_i;$               $i \in \{0, 1, 2, 3\}.$
            $Y_i := (Y_{i-4} - Y_i)W_i;$               $i \in \{4, 5, 6, 7\}.$

  2. Local FFTs.
       $Y_i := \text{FFT}_3(Y_i),$                     $i = 0, \ldots, 7.$

  3. Data permutation and write out. $P_{\text{out}} = P_{n_2,8}$
       $(P_{\text{out}}^{-1}Y)\left(1{:}n_1,\ 1{:}n_2,\ i \times \frac{n_3}{8} + [1{:}\frac{n_3}{8}]\right) := Y_i.$          $i = 0, \ldots, 7.$
```

data transfers from main memory to the SPE LS. Step 1 requires DMA transfers from LS to LS interwoven with local computations. Step 2 requires no use of DMA, all computations are local and no references to the main memory are required. Step 3 requires the use of the DMA for contiguous data transfers to the main memory from the SPE LS.

3.3 Streaming strategy

The streaming strategy consists of two aspects: spatial partition in memory and temporal scheduling for overlapping data uploading and downloading with arithmetic operations at the SPEs. We describe first the temporal relation to the distributed algorithm in Table 1. During the time the local computations are performed in Step 2, we exploit the availability of DMA to carry out Step 3 of the previous FFT instance, followed by Step 0 of the next FFT instance. Since the operation of the DMA is independent of the operations at the SPEs, as long as there are no LS memory bank conflicts, Steps 0 and 3 can take place concurrently with Step 2.

Figure 2 depicts the memory allocation scheme at the local store of each SPE. The LS is divided into four blocks. One of the blocks is for the program instructions and the twiddle factors, the latter take only modest space. The other three blocks are of the same size. The middle of the three blocks is used as the working buffer for exchanged data, while the other two take turns to host the previous FFT output and the next FFT input.

3.4 Experimental Results

We present the experimental results. The input data are stored in the main memory and fetched to the local stores at SPEs. The output data arrays are written back to the main memory. Each complex data array is placed in memory into two real arrays. This is consistent and convenient with the packing scheme in the computation of local correlation coefficients.

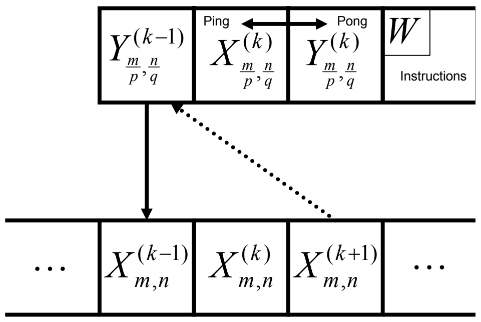
Ping
Pong
$Y^{(k-1)}_{\frac{m}{p},\frac{n}{q}}$
$X^{(k)}_{\frac{m}{p},\frac{n}{q}}$
$Y^{(k)}_{\frac{m}{p},\frac{n}{q}}$
W
Instructions
$\cdots$
$X^{(k-1)}_{m,n}$
$X^{(k)}_{m,n}$
$X^{(k+1)}_{m,n}$
$\cdots$

Figure 2. The computation of the local FFTs on the current data array in overlap with unloading the previous output and loading the next input.

Table 2 shows the performance in GFLOPS of FFTs on 8 SPEs with complex data in single precision, up to the memory capacity of the local stores. The figures are based on the execution time to complete each FFT while computing a streaming sequence of FFTs. The number of arithmetic operations for an array of volume n is counted as $5n\log_2(n)$. The twiddle factors are pre-computed and used repeatedly for successive FFTs. The time for computing the twiddle factors is therefore not included.

We note that the throughput is about 12000 FFTs of size 256×256 per second on the Cell, which is more than 100 times faster than the corresponding computation in MATLAB on a 3.4 GHz Pentium 4 workstation.

Table 2. Execution time for each 2D/3D FFT in a stream.

Data	Parallel on 8 SPEs			
Size	Total GFLOPs	GFLOPs/SPE	cycles$\times 10^3$	μs
128×256	46	5.75	167	53.4
256×128	46.5	5.81	165	52.8
256×256	67.9	8.48	241	77.2
$32\times32\times32$	55.9	6.98	137	43.9
$32\times16\times64$	57.1	7.13	134	43.0
$32\times32\times64$	72.1	9.01	227	72.7

4. CONCLUDING REMARKS

We have introduced an effective method for accelerating the computation of local correlation coefficients via the use of the FFT and the utilization of modern or emerging architectures. The use of multi-dimensional FFTs in computation of local correlation coefficients without sacrificing the local normalization seems new although simple. The parallelization of a single multi-dimensional FFT differs from other existing parallel FFTs on the Cell in that it merges the partial ordering among dimensional transforms with the total ordering in each dimensional transform and hence enhances the concurrency between arithmetic computation and data exchange among processing units. The streaming strategy, which is not commonly provided by existing FFT library routines, explores the parallelism across FFTs in succession and increases the throughput rate. The study on the Cell may be similarly carried out on other existing and emerging high-performance architectures, including graphics processing units and field programmable gate arrays. Experimental results have shown the promise and potential of employing the method in real-time applications or large scale computational simulations where the computation of local correlation coefficients is a major or basic step. The FFT can be made even faster by skipping certain data permutations that are not necessary for the computation of correlation coefficients.

Acknowledgments

This work is supported in part by DARPA/MTO/DESA program. The authors thank also Terra Soft Solutions for providing accesses to an IBM Cell blade QS20.

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] Casasent, D. and Psaltis, D., "Position oriented and scale invariant optical correlation," *Appl. Opt.* , 1793–1799 (1976).

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

[4] Castro, E. D. and Morandi, C., "Registration of translated and rotated images using finite Fourier transforms," *IEEE Trans. Pattern Anal. Mach. Intell.* **9**(5), 700–703 (1987).

[5] Lee, D. J., Kpile, T. F., and Mitra, S., "Digital registration techniques for sequential fundus images," in [*IEEE Proc. SPIE: Applic. Digital Image Processing X*], **829**, 293–300 (1987).

[6] Appicella, A., Kippenhan, J. S., and Nagel, J. H., "Fast multi-modality image matching," in [*SPIE Med. Imaging 111: Image Processing*], **1092**, 252–263 (1989).

[7] Brown, L. G., "A survey of image registration techniques," *ACM Comput. Surv.* **24**(4), 325–376 (1992).

[8] Zitova, B. and Flusser, J., "Image registration methods: a survey," *Image and Vision Computing* **21**, 977–1000 (2003).

[9] Reddy, B. S. and Chatterji, B. N., "An FFT-based technique for translation, rotation, and scale-invariant image registration," *IEEE Trans. Image Proc.* **5**(8), 1266–1271 (1996).

[10] Kahle, J. A., Day, M. N., Hofstee, H. P., Johns, C. R., Maeurer, T. R., and Shippy, D., "Introduction to the Cell Multiprocessor," *IBM J. Res. Dev.* **49**(4/5), 589–604 (2005).

[11] Cooley, J. W. and Tukey, J. W., "An algorithm for the machine calculation of complex Fourier series," *Mathematics of Computation* **19**, 297–301 (April 1965).