Hardware Acceleration of Mutual Information-Based 3D Image Registration

Carlos R. Castro-Pareja and Raj Shekhar†

Department of Diagnostic Radiology, University of Maryland School of Medicine, Baltimore, Maryland, USA

Real-time image registration is potentially an enabling technology for the effective and efficient use of many image-guided diagnostic and treatment procedures relying on multimodality image fusion or serial image comparison. Mutual information is currently the best-known image similarity measure for intensity-based multimodality image registration. The calculation of mutual information is memory intensive and does not benefit from cache-based memory architectures in standard software implementations, i.e., the calculation incurs a large number of cache misses. Previous attempts to perform image registration in real time focused on parallel supercomputer implementations, which achieved significant speedups using large, expensive supercomputers that are impractical for clinical deployment. We present a hardware architecture that can be used to accelerate a number of linear and elastic image registration algorithms that use mutual information as an image similarity measure. A proof-of-concept implementation of the architecture achieved speedups of 30 for linear registration and 100 for elastic registration against a 3.2 GHz Pentium III Xeon workstation. Further speedup can be achieved by using several modules in parallel.

Journal of Imaging Science and Technology 49: 105–113 (2005)

Introduction

Image registration is the process of aligning two images that represent the same anatomy from different points of view, at different times or using different imaging modalities. Image registration is an important tool in medical imaging, where it is used to merge or compare images obtained from a variety of modalities, such as magnetic resonance imaging (MRI), computed tomo-graphy (CT), positron emission tomography (PET), single photon emission computed tomography (SPECT) and ultrasound. In medicine, real-time image registration can be an enabling technology for the effective and efficient use of many diagnostic and image-guided treatment procedures relying on either multimodality image fusion or serial image comparison.[1] Image registration algorithms can be classified into linear and elastic registration algorithms. Linear registration algorithms use global transformations with a combination of rotation, translation, scaling and shear components. Elastic image registration refers to the case in which one of the images is transformed using a nonlinear, continuous transformation. The process of image registration yields the transformation field that best locates one of the images (the floating image) into the other (the reference image), and can be used to track changes in the anatomy. Both linear and elastic registration are useful tools in many medical applications.[2–7]

While a linear transformation can be simply defined as a superposition of rotations, translations, scaling and shear, and is always applied globally, there is no agreement in the image processing community about the best way to define and estimate an elastic transformation. Many algorithms for elastic image registration that have been reported in the literature are based on a subdivision of the image using a mesh of control points which is defined at progressively smaller subvolume sizes.[3,8–11] Such algorithms optimize the position of control points to find the best local transformation. The images are compared using an image similarity measure. For multimodality image registration the measure of choice is typically mutual information.[12,13]

The use of mutual information for image registration was first introduced by Wells et al.[14] Pluim et al. presented a comprehensive survey of the literature on mutual information-based registration.[13] The goal of mutual information-based registration is to find the transformation that best aligns two images by maximizing the mutual information between them. Figure 1 describes the mutual information-based registration process. The optimization algorithm is used to find the set of transformation parameters that maximizes the mutual information between the reference image and the transformed floating image. Depending on the particular application, the transformation can be linear (rigid or affine) or elastic. For linear transformations the parameters being optimized are commonly the rotation, translation and/or scaling coefficients, while for elastic transformations defined by a mesh of control points the parameters being optimized are the control point locations in the floating image space.

Original manuscript received July 16, 2004

† Corresponding Author: Raj Shekhar, rshekhar@umm.edu

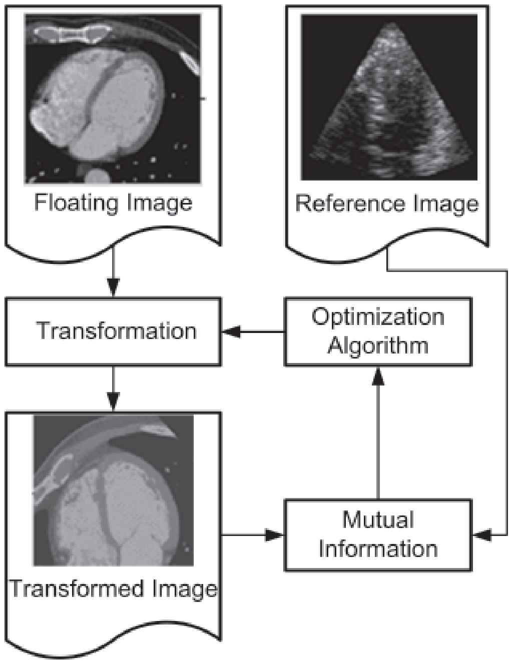


Figure 1. Mutual information-based registration flowchart.

Previous attempts to perform image registration in real time focused on parallel supercomputer implementations, which achieved real-time performance using large, expensive supercomputers that are too impractical to be deployed in a hospital.[15–18] Our Fast Automatic Image Registration (FAIR) architecture accelerated mutual information-based linear registration.[19] In this article we present a second generation architecture, called FAIR-II, that has been designed to accelerate both linear and elastic registration and achieves even higher speeds than the original FAIR architecture. FAIR-II architecture achieves registration speeds 30 to 100 times faster than a 3.2 GHz Pentium III-based PC workstation, depending on the underlying transformation. The single-module speedup results from using parallel memory access and parallel calculation pipelines. Using several modules in parallel can increase the overall speedup. An implementation of the architecture can be customized to meet the particular requirements of most control point mesh-based elastic registration algorithms.[8,10,11,17,20]

Mutual Information Calculation

Mutual information is defined as:

$$MI(RI,FI) = H(RI) + H(FI) - H(RI,FI), \tag{1}$$

where *RI* is the reference image, *FI* is the floating image (the one being transformed), *H*(*RI*) and *H*(*FI*) are the individual image entropies, and *H*(*RI*,*FI*) is their joint entropy. The entropies are computed as follows:

$$H(RI) = -\sum_{a} p_{RI}(a) \ln p_{RI}(a) \tag{2}$$

$$H(FI) = -\sum_{b} p_{FI}(b) \ln p_{FI}(b) \tag{3}$$

$$H(RI,FI) = -\sum_{a,b} p_{RI,FI}(a,b) \ln p_{RI,FI}(a,b) \tag{4}$$

The joint voxel intensity probability $p_{RI,FI}(a,b)$, i.e., the probability of a voxel in the reference image having an intensity a given that the corresponding voxel in the floating image has an intensity b, can be obtained from the joint histogram of the two images. The joint histogram represents the joint intensity probability distribution. In the process of mutual information-based registration, the dispersion of values within the joint histogram is minimized, which in turn minimizes the joint entropy and maximizes the mutual information. Calculation of mutual information involves first obtaining the individual and joint histograms of the two images and then calculating their individual and joint entropies using the histograms' values. Joint histogram calculation is a memory intensive task that does not benefit from cache-based memory architecture, i.e., it incurs a large number of cache misses, in standard software implementations.[19]

Architecture

As mentioned before, mutual information calculation can be divided into two steps. In the first step the mutual and individual histograms are formed. In the second step, the entropies are calculated and added according to Eqs. (1) through (4). The individual and joint voxel intensity probabilities are calculated from the joint and individual histograms. The time required to perform the first step is proportional to the image size, while the time required to perform the second step is proportional to the joint histogram size. Since typical 3D image sizes are between 64^3 and 512^3 voxels, and typical joint histogram sizes range between 32^2 and 256^2 bins, joint histogram calculation speed will normally be the most important factor in mutual information calculation time. We showed earlier that, for large images, joint histogram calculation can take 99.9% of the total mutual information calculation time in software.[19] On the other hand, off-chip entropy calculation requires the additional overhead of transmitting the joint histogram back to the host computer, which can take a significant amount of time. Because of this overhead we decided to implement entropy calculation on-chip. A block diagram of the FAIR-II architecture is shown in Fig. 2. The following subsections describe the operation of its different elements.

Joint Histogram Computation

The first step in mutual information calculation involves applying the current estimate of the inverse transformation to each voxel coordinate of the reference image to find the corresponding voxel coordinates in the floating image, and updating the joint histogram at the location dictated by the intensities of the voxel pair. The corresponding coordinates do not normally coincide with a grid point in the floating image, thus requiring interpolation. Partial volume (PV) interpolation, as introduced by Maes et al.[21] was chosen because it provides smooth changes in the histogram values as a result of small changes in the transformation.[13] The algorithm for joint histogram computation is shown in Fig. 3. Step (a) of the algorithm is performed by the voxel coordinate transformation unit. Steps (b) and (c) are performed in parallel. The reference image coordinate values are passed to the reference image controller, the integer components of the floating image coordinates are passed to the floating image controller, and its fractional components to the PV interpolator. Finally the interpolation weights calculated by the PV interpolator are accumulated into the joint histogram.

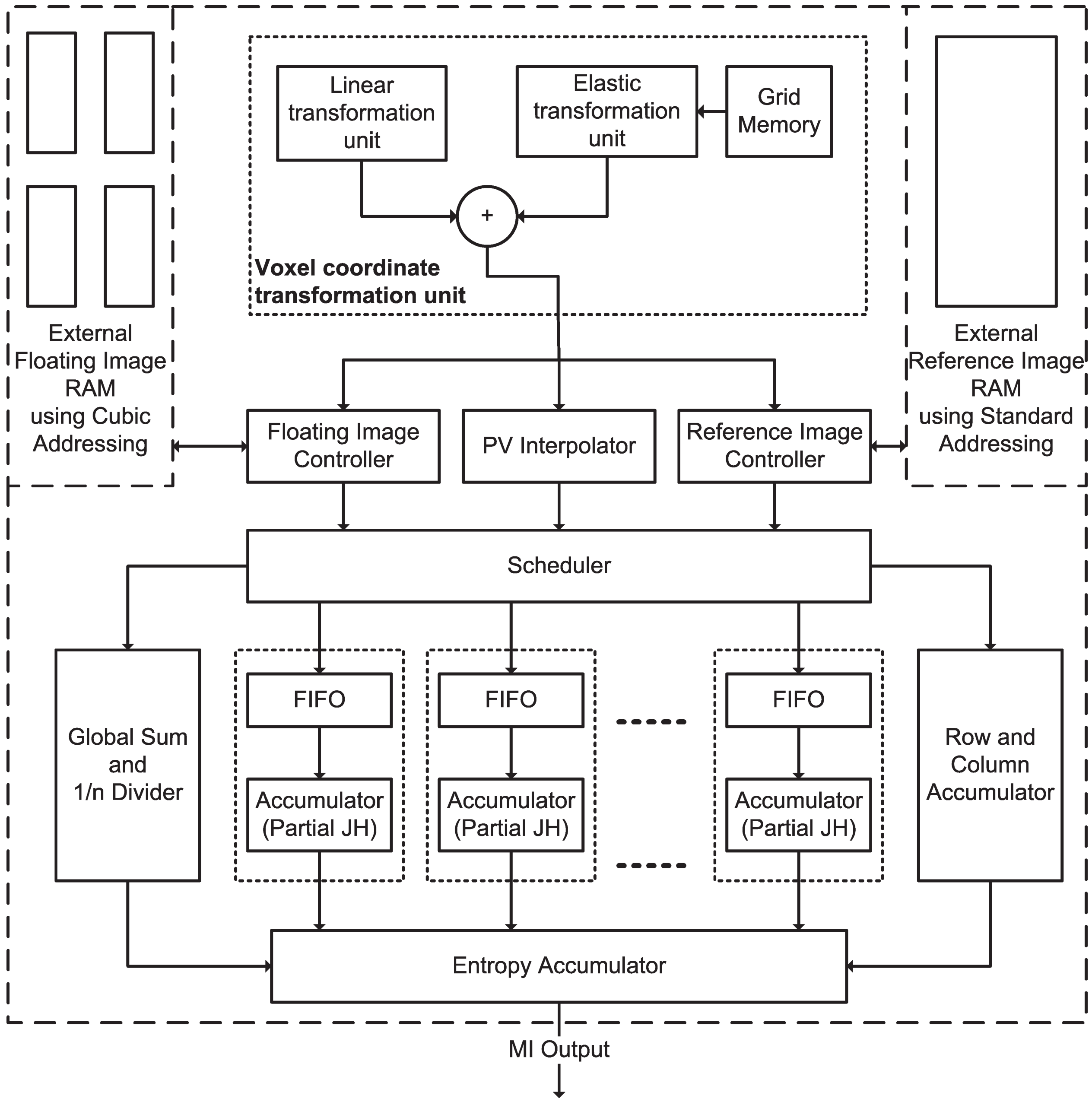

Linear transformation unit
Elastic transformation unit
Grid Memory
+
Voxel coordinate transformation unit
External Floating Image RAM using Cubic Addressing
External Reference Image RAM using Standard Addressing
Floating Image Controller
PV Interpolator
Reference Image Controller
Scheduler
Global Sum and 1/n Divider
FIFO
Accumulator (Partial JH)
Row and Column Accumulator
Entropy Accumulator
MI Output

Figure 2. Block diagram of the FAIR-II architecture.

Voxel Coordinate Transformation
The voxel coordinate transformation unit performs the transformation of reference image voxel coordinates into locations in the floating image space. An elastic transformation is separated into two components: a global linear transformation and a local, control point mesh-based transformation, each calculated by an independent submodule. Floating image voxel coordinates are calculated as shown in Eq. (5):

$$\bar{v}_f = T_{global} \cdot \bar{v}_r + \bar{d}_{local}(\bar{v}_r), \qquad (5)$$

where $\bar{v}_r$ is a vector containing the coordinates of a voxel in the reference image, $\bar{v}_f$ is a vector containing the corresponding floating image coordinates, T_{global} is the global linear transformation (defined by a 4×4 matrix)

For each voxel in the reference image,

a. Calculate corresponding floating image coordinates (apply coordinate transformation)

b. Load corresponding floating image $2 \times 2 \times 2$ voxel neighborhood (8 intensity values are loaded in parallel)

c. Calculate the 8 partial volume interpolation weights (similar to trilinear interpolation, but weights are not added at the end)

d. Accumulate interpolation weights into corresponding joint histogram bins.

Figure 3. Joint histogram computation algorithm.

and $\bar{d}_{local}$ is the local deformation function. The local deformation function is defined by a uniform 3D mesh of control points, which is stored in the Grid Memory module. Each control point is mapped to a fixed location in the reference image space and is associated to the corresponding value of $\bar{d}_{local}$. Local deformation values for reference image coordinates located between mesh control points are calculated using interpolation.

The effects of both transformations are superimposed by addition. The main reason to decompose the transformation into its linear and local components is to reduce the hardware requirements for the Grid Memory and the control point coordinate interpolator. This way the dynamic range of mesh control point coordinates is restricted to the range of allowable internal deformations, which is normally much smaller than the range of the global linear transformation, which includes the whole floating image space. It also allows performing linear transformations, by setting the control point values to zero, i.e., $\bar{d}_{local}(\bar{v}_r) = \bar{0}$.

The particular implementation of the voxel coordinate transformation unit depends on the algorithm being accelerated. Some of the most commonly used transformation algorithms include local linear transformations,[11] and control-point-mesh-based transformations using trilinear interpolation[8,22] or cubic (spline) interpolation.[9,20,23] Since all the aforementioned transformation algorithms can be implemented using a hardware pipeline, the choice of a particular transformation algorithm does not have any impact in the final mutual information calculation speed. However, since more complex transformations require a larger number of arithmetic units and/or a deeper pipeline, the hardware requirements of a particular implementation will depend on the transformation algorithm being implemented.

Interpolation Weight Calculation

The reference image coordinates coming from the voxel coordinate transformation unit ($\bar{v}_r$ in Eq. (5)) are always integer and correspond to exact reference image voxel locations. However, their corresponding floating image coordinates ($\bar{v}_f$) have fractional components and can land outside of the actual image. As in the original FAIR architecture,[19] the integer components of the floating image coordinates are used to locate the corresponding $2 \times 2 \times 2$ floating image voxel neighborhood and thus passed to the Floating Image Controller, while the fractional components are used for interpolation. As mentioned before, Partial Volume interpolation is used.[21] As a result of PV interpolation, eight interpolation weights $w_{0:7}$ are generated. Each of these interpolation weights is associated to a voxel belonging to the $2 \times 2 \times 2$ floating image neighborhood that contains $\bar{v}_f$. These weights are passed along with the corresponding reference image intensity value and eight floating image intensity values to the Scheduler module.

Image Memory Access

Each image memory controller has an independent memory bus, therefore allowing parallel access of voxel intensity pairs. The reference image controller loads the reference image voxel intensity values. At the same time, the floating image controller accesses the corresponding $2 \times 2 \times 2$ voxel neighborhoods in parallel through a memory bus implemented using cubic addressing.[24] The reference image memory is accessed sequentially, using burst accesses. While the image memories are accessed, the PV interpolator unit calculates the interpolation weights. Since floating image accesses are not sequential, it is recommended to use SRAMs to store the floating image, thus allowing one $2 \times 2 \times 2$ neighborhood access per external clock cycle.

Joint Histogram Accumulation

For a given reference image voxel, there are eight voxel intensity pairs, each with its own interpolation weight. Since the joint histogram memory needs to be read and written once for each intensity pair, 16 accesses to the joint histogram memory are required per reference image voxel. To provide 16 accesses in the time of a single image memory access, three different techniques are used:

1. The joint histogram is implemented using internal on-chip memory, running at a higher frequency than the external image memory. This reduces the number of necessary accesses per clock cycle by a factor equal to the ratio between the internal and external clock rates.
2. Dual-ported memories are used to allow simultaneous read and write accesses. This reduces in half the number of necessary accesses per memory port.
3. The joint histogram memory is partitioned to allow a higher number of accesses to occur at the same time.

The joint histogram memory is partitioned into $2n$ blocks. Each partition has its own independent accumulator pipeline. The number of joint histogram partitions depends on the internal clock speed. The following inequality must hold to ensure that the data flow into the accumulator FIFOs is equal or larger than their output flow:

$$JH\_Partitions \geq \frac{8 \cdot f_{RI}}{f_{acc}}, \tag{6}$$

where f_{RI} is the reference image clock frequency and f_{acc} is the accumulator clock frequency. The joint histogram memory is arranged such that the address of a joint histogram bin is obtained by concatenating the corresponding reference image and floating image voxel intensity values. Joint histogram partitioning is based on the recognition that the neighboring voxels even in relatively uniform regions in clinical images vary slightly, in terms of a few LSBs. Partitioning is, therefore, performed according to the n LSBs of the memory addresses ($n = \log_2(JH\_Partitions)$). The least significant bits (LSBs) of the joint histogram memory addresses correspond to the LSBs of the floating image voxel intensity values. This partitioning scheme maximizes the likelihood that different floating image voxel intensity values in the same neighborhood will be assigned to different joint histogram memory blocks, thus allowing a higher degree of parallelism in joint histogram RAM accesses. An example with a joint histogram of size 8×8 and four partitions ($n = 2$) is shown in Fig. 4. In this example the intensity values are three bits wide. Therefore, a voxel intensity pair will have six bits. The three most significant bits, labeled $vr2{:}0$, contain a reference image voxel intensity value, while the three least significant bits, labeled $vf2{:}0$, contain the corresponding floating image voxel intensity value. To map the voxel intensity pair to a given joint histogram partition, its least two significant bits, labeled $JHP1$ and $JHP0$, are used. These bits correspond to $vf1{:}0$. The remaining four upper bits of the voxel intensity pair, labeled $JH3{:}0$, are used to address a location within the joint histogram partition. For

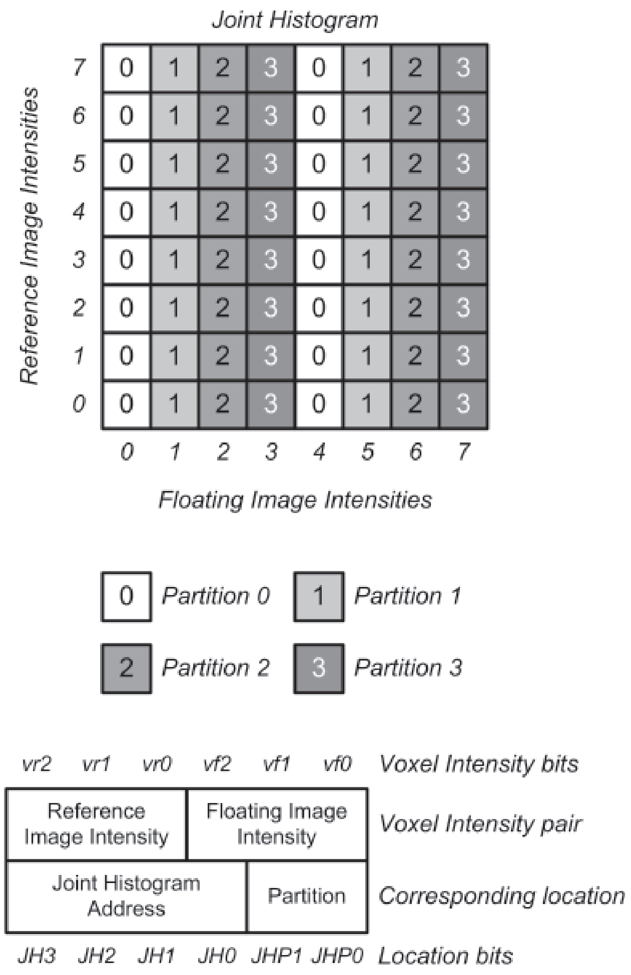


Figure 4. Joint histogram partitioning example

example, if a voxel intensity pair consists of the reference image value $vr2{:}0 = 010_2$ (the subscript denotes the base), and the floating image value $vf2{:}0 = 110_2$, the corresponding voxel intensity pair will be 010110_2, which will be mapped to Partition 2 (since $JHP1{:}0 = 10_2 = 2$), at the address 0101_2 (since $JH3{:}0 = 0101_2$).

A limitation of this partitioning scheme is that, in the presence of large image areas of identical intensity, it would not provide the desired parallelism. To prevent this problem, the scheduler groups equivalent voxel intensity pairs and adds their corresponding interpolation weights together before sending them to the accumulators. This operation also prevents read-after-write hazards in the accumulation pipeline due to consecutive accesses to the same voxel intensity pair. FIFOs are used as buffers to handle traffic spikes to individual joint histogram blocks.

While the joint histogram is computed, the row and column accumulator unit computes the individual histograms. The architecture of the row and column accumulator is similar to the joint histogram accumulation pipeline. The global sum module keeps track of the number of voxels accumulated into the histograms and calculates its reciprocal value, i.e., it applies the $1/n$ function to it. The resulting value is used during entropy accumulation to calculate the voxel intensity probabilities from the individual and joint histogram values.

The size of the internal joint histogram memory must be equal to the largest joint histogram size being used in the algorithm of choice. However, variable joint histogram sizes can be implemented in hardware by performing a bit AND operation against the contents of a bit mask register. Having a one-bit mask register for the floating image and a different mask register for the reference image allows independent control over the lateral joint histogram dimensions. This is useful when dealing with multimodality images and also when performing nonrigid registration, since performing local refinements to the transformation at small scales affects smaller portions of the floating image, thus requiring the use of smaller joint histogram sizes to reduce the dispersion of values in the joint histogram.

Entropy Accumulation

In the second step of mutual information calculation, the accumulator modules send the partial joint histogram values to the entropy accumulator. In the entropy accumulator the individual and joint voxel intensity probabilities are calculated from the joint histogram values, and the $p \cdot \ln(p)$ function is applied to them. The results are then accumulated, thus obtaining the mutual information value.

To calculate mutual information, it is necessary to evaluate the function $f(p) = p \cdot \ln(p)$ for all the individual and joint intensity probabilities. Since the probabilities range from 0 to 1, the $\ln(p)$ function will have a range of $[-\infty, 0]$, with the problem that it is undefined for $p = 0$. Fortunately, the $p \cdot \ln(p)$ function has a much more limited range of $[-e^{-1}, 0]$ and is defined in the full range of $0 \le p \le 1$. Hardware-based logarithm calculation is usually performed using series approximations or a combination of series approximations and look-up tables (LUTs).[25–29] Using series approximation requires implementing a series of arithmetic units that consume significant hardware resources, and take several clock cycles to finish calculation. Existing LUT-based approaches for logarithm calculation have a limited accuracy (up to 24 fixed-point bits) that makes them unsuitable for the current application.[28,29] In this subsection we present an algorithm for the calculation of $p \cdot \ln(p)$ that allows simple, LUT-based implementation in FPGAs and has a small enough error range acceptable for its application in mutual information calculation.

LUT-Based Entropy Calculation

The function $f(p)$ is approximated in the range [0, 1] by using the piecewise-polynomial function $\hat{f}_{N,m}(p)$ with m segments, defined in Eq. (7) below.

$$\hat{f}_{N,m}(p) = \begin{cases} P_0(p) & \text{for } 0 \le p < \Delta p \\ P_1(p \bmod \Delta p) & \text{for } \Delta p \le p < 2\Delta p \\ \vdots & \vdots \\ P_i(p \bmod \Delta p) & \text{for } i \cdot \Delta p \le p < (i+1) \cdot \Delta p \\ \vdots & \vdots \\ P_{m-1}(p \bmod \Delta p) & \text{for } (m-1) \cdot \Delta p \le p < 1 \end{cases}, \quad (7)$$

where $\Delta p = 1/m$ is the segment size of $\hat{f}_{N,m}(p)$ and P_i is a polynomial of order $N - 1$. To minimize the maximum approximation error, each P_i is obtained by calculating the Chebyshev approximation for $f(p)$, defined in Eq. (8) for $i \cdot \Delta p \le p < (i + 1) \cdot \Delta p$.[30] The Chebyshev approximation is simple to calculate for continuous functions and has the advantage that it is very close to the minimax approximation, the most accurate polynomial approximation.[31] Equation (9) is used to calculate the coefficients.

$$P_i(p \bmod \Delta p) \approx \left[\sum_{k=0}^{N-1} c_{i,k} T_k\left(\frac{2(p \bmod \Delta p)}{\Delta p} - 1\right)\right] - \frac{1}{2} c_{i,0} \quad (8)$$

$$c_{i,k} = \frac{2}{N} \sum_{l=1}^{N} f\left[\frac{\Delta p}{2}\cos\left(\frac{\pi(l-1/2)}{N}\right) + \Delta p \cdot (i+1/2)\right]\cos\left(\frac{\pi k(l-1/2)}{N}\right) \quad (9)$$

The Chebyshev polynomials are defined by $T_n(x) = \cos(n \cdot \arccos(x))$, for $-1 \le x \le 1$. The Chebyshev polynomials of order zero to three are shown in Eq. (10). Since each polynomial is used to approximate $f(p)$ in a specific $[i \cdot \Delta p, (i+1) \cdot \Delta p]$ range, whereas the Chebyshev polynomials are defined in $[-1, 1]$, the variable conversion shown in Eq. (11) was applied to the equations.

$$T_0(x) = 1,\ T_1(x) = x,\ T_2(x) = 2x^2 - 1,\ T_3(x) = 4x^3 - 3x \quad (10)$$

$$x = \left(2(p \bmod \Delta p) - \Delta p\right)/\Delta p \quad (11)$$

To keep our arithmetic pipeline simple, we considered only first, second and third-order approximations in our architecture. Equation (12) defines the ith polynomial component of $\hat{f}_N(p)$:

$$P_i(p_d) = k_{i,3} \cdot p_d^3 + k_{i,2} \cdot p_d^2 + k_{i,1} \cdot p_d + k_{i,0}, \quad (12)$$

where $p_d = p \bmod \Delta p$. The polynomial coefficients $k_{i,j}$ are stored in the ith entry of the LUT. They are calculated from the Chebyshev coefficients as shown in Eqs. (13) through (16), which are derived from Eqs. (8), (10) and (11):

$$k_{i,0} = 0.5 \cdot c_{i,0} - c_{i,1} + c_{i,2} - c_{i,3} \quad (13)$$

$$k_{i,1} = \left(c_{i,1} - 4 \cdot c_{i,2} + 9 \cdot c_{i,3}\right) \cdot (2/\Delta p) \quad (14)$$

$$k_{i,1} = \left(c_{i,1} - 4 \cdot c_{i,2} + 9 \cdot c_{i,3}\right) \cdot (2/\Delta p) k_{i,2} = \left(2 \cdot c_{i,2} - 12 \cdot c_{i,3}\right) \cdot (2/\Delta p)^2 \quad (15)$$

$$k_{i,3} = 4 \cdot c_{i,3} \cdot (2/\Delta p)^3 \quad (16)$$

Error Characteristics

The approximation error in the ith segment of $\hat{f}_{N,m}(p)$ is given by:

$$\varepsilon_i(p) = \hat{f}_{N,m}(p) - f(p) = P_i(p \bmod \Delta p) - f(p) \quad (17)$$

The position $p_{\max\_ei}$ of the maximum absolute error e_i in this segment is obtained by numerically evaluating p when the derivative of the error is equal to zero:

$$d\varepsilon_i/dp = \dot{P}_i\left(p_{\max\_ei} \bmod \Delta p\right) - \dot{f}\left(p_{\max\_ei}\right) = 0 \quad (18)$$

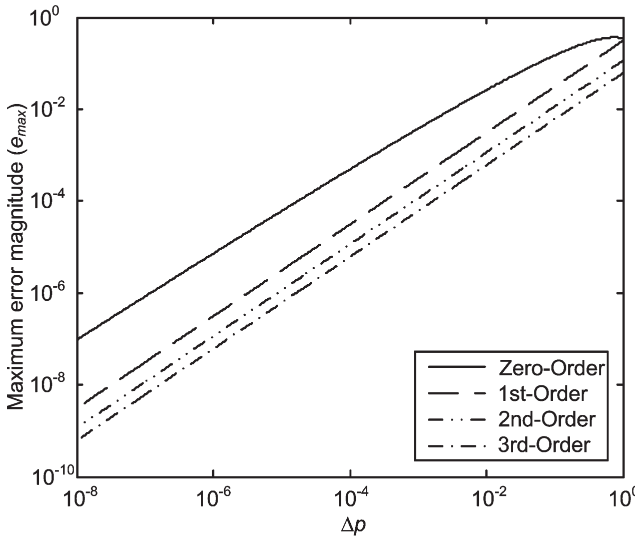


Figure 5. Plots of maximum approximation error $e_{\max}$ versus Δp for different approximation polynomial orders.

$$\Rightarrow 3k_{i,3} \cdot \left(p_{\max\_ei} \bmod \Delta p\right)^2 + 2k_{i,2} \cdot \left(p_{\max\_ei} \bmod \Delta p\right) + k_{i,1} = 1 + \ln\left(p_{\max\_ei}\right) \quad (19)$$

The maximum absolute error is then found by inserting $p_{\max\_ei}$ into the error equation:

$$e_i = \max_{i \cdot \Delta p \le p < (i+1) \cdot \Delta p} \varepsilon_i(p) = \varepsilon_i\left(p_{\max\_ei}\right) \quad (20)$$

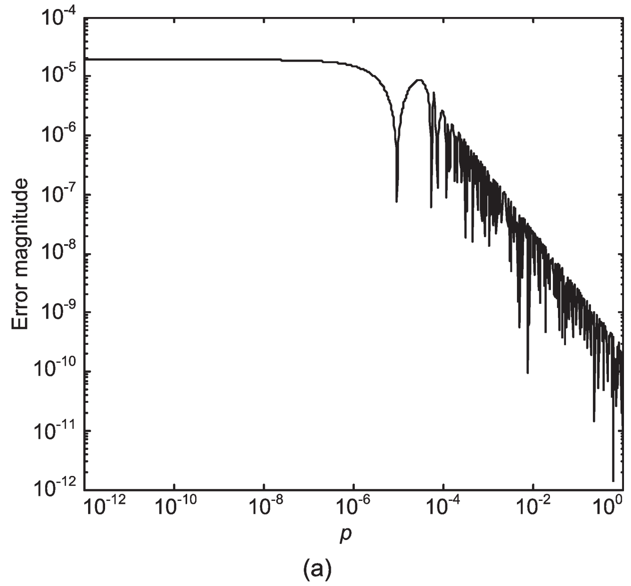
Since using an Nth-order piecewise-polynomial approximation assumes that the Nth derivative of $f(p)$ is constant in each segment of $\hat{f}_{N,m}(p)$, the segment size Δp must be determined by the rate of change of the Nth derivative and the desired maximum error. The first and second derivatives of $f(p)$ are $\dot{f}(p) = 1 + \ln(p)$ and $\ddot{f}(p) = 1/p$. The rate of change of all derivatives approaches infinity as p nears zero and decreases steadily as p increases. Therefore, the approximation error is higher for smaller values of p, which further implies that this error is largest in the first segment of $\hat{f}_{N,m}(p)$, i.e., $e_{\max} = e_0 = \varepsilon_0(p_{\max0})$).

For a given Δp, using a higher-order polynomial approximation yields a lower maximum error. Therefore, for a given number of LUT entries, $e_{\max}$ will decrease as the polynomial order increases. The plot on Fig. 5 illustrates the dependence between Δp, $e_{\max}$ and the polynomial order. To be able to store the LUT using internal memory, a reasonable maximum number of entries is 16K (2^{14} bins). For a LUT with 16K entries, the maximum error is in the order of 10^{-8} to 10^{-10} for first- to third-order approximations.

Multi-LUT Approach

It was shown in the previous section that the maximum error will occur in the first segment of $\hat{f}_{N,m}(p)$. Figure 6(a) shows the error magnitude plot for an LUT with 2^{14} bins, using third-order polynomials and double precision floating point numbers. As predicted before, the error decreases as p increases.

To further reduce LUT size and improve accuracy, we used a multi-LUT approach, where each LUT had a different Δp value and was based on a different



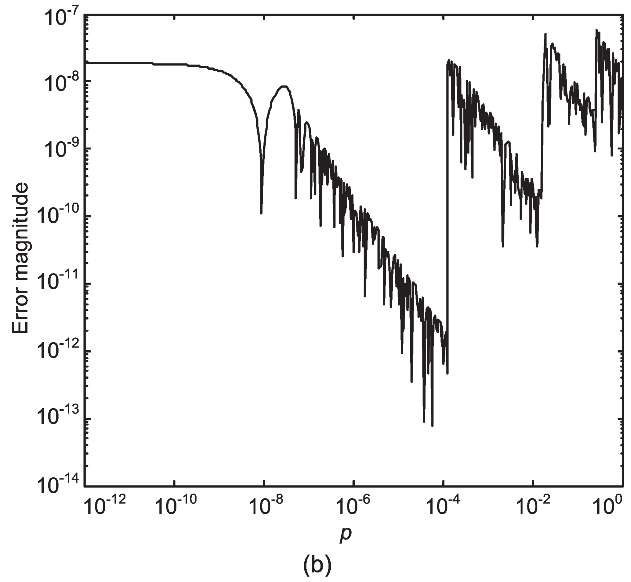


Figure 6. (a) Error magnitude plot for a single LUT with 2^{14} entries; and (b) Error magnitude plot for 4 LUTs with 2^{11} entries each.

approximation function $\hat{f}_{N,m_l}(p)$. The general n-LUT form of this approximation is shown in Eq. (21), where $\Delta p_1 < \Delta p_2 < \ldots < \Delta p_n$. Having larger values of Δp for higher values of p is equivalent to approximating the function using variable segment sizes, which allows us to reduce the LUT size while keeping the approximation error below the allowable maximum.

$$\hat{F}_N(p) = \begin{cases} \hat{f}_{N,m_1}(p) & \text{for } 0 \le p < p_{\max_1} \\ \hat{f}_{N,m_2}(p) & \text{for } p_{\max_1} \le p < p_{\max_2} \\ \vdots & \vdots \\ \hat{f}_{N,m_n}(p) & \text{for } p_{\max_{n-1}} \le p < 1 \end{cases} \tag{21}$$

Let $p_{min_i} = p_{max_{i-1}}$ and $p_{max_0} = 0$. Given a maximum allowable error and polynomial order, we can vary Δp and use Eqs. (18) and (20) to determine the range [$p_{\min}$, 1] over which the error criterion is satisfied. As explained above, the smaller the Δp, the wider is the [$p_{\min}$, 1] interval, and hence smaller is $p_{\min}$. In our analysis, we considered only those Δp values that were powers of two to make the selection of the LUT and LUT-segment simple. For a given probability range, a higher-order approximation leads to a larger Δp value, thus resulting in smaller LUT sizes. The following algorithm was used to determine the number and the probability range of the LUTs, given a target size (number of samples) for each LUT:

1. Given the target LUT size m_1, obtain the desired Dp_1 for the segment starting at $p_{\min} = 0$ using Eqs. (18) and (20). The probability range of the first LUT thus equals [0, p_{max_1}], where $p_{max_1} = \Delta p_1 \cdot m_1$.
2. Use p_{max_1} from the first LUT to obtain Δp for the second LUT. Determine the range of the second LUT.
3. Iterate step 2 to obtain the Δp values and LUT ranges for the subsequent LUTs, until the entire [0, 1] range has been covered.

TABLE I. Characteristics of a 4-LUT Implementation of the Entropy Calculation Pipeline

		LUT Range		
LUT Number	Number of Entries (m)	$p_{\min}$	$p_{\max}$	Δp
1	2048	0	2^{-13}	2^{-24}
2	2048	2^{-13}	2^{-6}	2^{-17}
3	2048	2^{-6}	2^{-2}	2^{-13}
4	2048	2^{-2}	2^{-0}	2^{-11}

As stated before, the range of $f(p)$ for p in [0, 1] is $[-e^{-1}, 0] \subset [-0.5, 0]$. Since the desired accuracy is 10^{-9} ($\sim 2^{-30}$), a 30-bit fixed-point representation will have sufficient dynamic range for the $f(p)$ calculation. Using a higher number of bits will reduce the rounding error.

Using this approach we designed the first-order polynomial, 4-LUT configuration shown in Table I. The maximum error was in the order of 10^{-8} for a total of 8K entries. Figure 6(b) shows a plot of the error magnitude versus p for this configuration. Using lower-order polynomials has the advantage of reducing the LUT data width and the number of arithmetic components in the pipeline.

Implementation and Results

The system was implemented as a proof of concept using an Altera Stratix EP1S40 FPGA in a PCI prototyping board manufactured by SBS, Inc.[32] The FPGA ran at a maximum internal frequency of 200 MHz, with a 50 MHz reference image RAM bus and a 100 MHz floating image RAM bus. Joint histogram RAM was partitioned into eight blocks, and implemented using internal 4 Kb memory blocks. Reference and floating image memories were implemented using standard PC100 SDRAMS and high-speed SRAMs, respectively. Entropy calculation was implemented using the 4-LUT, first-order polynomial configuration shown in Table I. As shown before, for a given LUT, the error magnitude decreases as p increases. All LUTs were implemented inside one internal 512 Kbit memory block. Mutual information was calculated using 32-bit fixed point numbers.

As implemented, the system was able to process 50 million reference image voxels per second. Timing characteristics of the system were compared against optimized software implementations and the hardware implementation of the previous FAIR architecture.[19] Table II shows the characteristics of the different systems used

TABLE II. Comparison System Characteristics

System Name	Processor Type	Processor Speed	Memory Type	Memory Speed
SW1	Intel PIII Xeon	500 MHz	SDRAM	100 MHz
SW2	Intel PIII Xeon	3.2 GHz	DDRAM	266 MHz
FAIR[19]	Altera Acex EP1K100 FPGA	80 MHz	SDRAM	80 MHz
FAIR-II	Altera Stratix EP1S40 FPGA	200 MHz	SRAM	50 MHz

TABLE III. Comparison of Mutual Information Calculation Times for Different Implementations

	Mutual information calculation time (milliseconds)								
Image Size (voxels)	2^{18}			2^{21}			2^{24}		
Number of subvolumes	1*	8	64	1*	8	64	1*	8	64
SW1	562	1682	1685	4712	13434	13530	38500	106269	105972
SW2	120	499	485	1025	3547	3945	10200	32351	33983
FAIR[19]	62	**	**	430	**	**	3370	**	**
FAIR-II	5	5	5	42	42	42	335	335	335

*In the 1-subvolume case an linear transformation was used in the software version

**FAIR supports only linear transformations

for benchmarking, and Table III shows the timing results. The number of subvolumes refers to the number of $2 \times 2 \times 2$ control point neighborhoods in the control point mesh used to define the local deformations, and is an indicator of the resolution at which local deformations are defined. When local deformations are used the software implementations become roughly three times slower due to the extra computational load from accessing the grid memory and performing the extra interpolations. The software execution times prove the fact that mutual information calculation time is more dependent on the external memory speed than the processor speed, especially for large image sizes. FAIR-II delivered a speedup of 30 for linear registration and 100 for elastic registration against the corresponding software implementations on a 3.2-GHz PIII Xeon workstation with 1 GB of 266 MHz DDRAM.

Conclusions

Elastic image registration is currently an active field of research in the medical imaging community, far from being considered a solved problem. Most current algorithms for elastic image registration are tailored to specific applications, although many of them share significant characteristics. The FAIR-II architecture presented in this paper does not aim to accelerate a specific algorithm, but rather any algorithm based on a uniform control point mesh-based transformation that uses mutual information as an image similarity measure. Mutual information was chosen because it is widely accepted in the imaging community as the best image similarity measure currently available for single- or multimodality image registration. FAIR-II achieves speedup rates an order of magnitude higher than those achieved by the first-generation FAIR architecture for linear registration and also supports elastic registration.

Future work on the acceleration of mutual information calculation can be performed on the estimation of the joint voxel intensity probability distribution, which is calculated from the joint histogram. The current architecture uses Partial Volume interpolation to update the joint histogram. When performing elastic registration, a more accepted way to calculate the joint histogram of small image segments is using Parzen windowing. The use of Parzen windowing for calculating the joint intensity probability distribution has been well documented in the literature.[14,33–36] It is based on accumulating the joint histogram using a Gaussian kernel centered at the location corresponding to the current voxel intensity pair. Using Parzen windowing implies accessing the joint histogram more often than when using Partial Volume interpolation. Since the joint histogram is accessed at different locations by definition, implementing Parzen windowing will require the use of deeper partitioning than presented, but with simpler accumulation pipelines.

Acceleration of elastic image registration algorithms is widely recognized as one of the current requirements to bring these algorithms into use in clinical procedures, especially in the operating room.[13,37] The FAIR-II architecture is an important step in this direction since it allows the implementation of image registration systems that are economical and compact and thus suitable for clinical deployment.

Acknowledgment. This work was supported by the National Medical Technology Testbed grant DAMD17-97-2-7016.

References

1. R. Shekhar, V. Zagrodsky, C. R. Castro-Pareja, V. Walimbe, and J. M. Jagadeesh, High-speed registration of three- and four-dimensional medical images by using voxel similarity, *Radiographics* **23,** 1673 (2003).
2. J. V. Hajnal, D. J. Hawkes and D. L. G. Hill, *Medical Image Registration*, CRC Press, Boca Raton, FL, 2001.
3. D. Rueckert, Nonrigid registration: concepts, algorithms, and applications, *Medical Image Registration*, J. V. Hajnal, D. L. G. Hill, and D. J. Hawkes, Eds., CRC Press, Boca Raton, FL, 2001.
4. R. Myers, The application of PET-MR image registration in the brain, *Brit. J. Radiol.* **75,** S31 (2002).
5. G. Xiao, J. M. Brady, J. A. Noble, M. Burcher, and R. English, Nonrigid registration of 3D free-hand ultrasound images of the breast, *IEEE Trans. Med. Imaging* **21,** 405 (2002).
6. M. A. Wirth, J. Narhan and D. Gray, Nonrigid mammogram registration using mutual information, *Proc. SPIE* **4684,** 562 (2002).
7. Z. Lee, K. K. Nagano, J. L. Duerk, D. B. Sodee, and D. L. Wilson, Automatic registration of MR and SPECT images for treatment planning in prostate cancer, *Academic Radiology* **10,** 673 (2003).
8. U. Kjems, S. C. Strother, J. Anderson, I. Law, and L. K. Hansen, Enhancing the multivariate signal of [^{15}O] water PET studies with a new nonlinear neuroanatomical registration algorithm, *IEEE Trans. Med. Imaging* **18,** 306 (1999).
9. T. Rohlfing and J. Maurer, Intensity-based non-rigid registration using adaptive multilevel free-form deformation with an incompressibility constraint, *Lecture Notes in Computer Science* **2208,** 111 (2001).

10. J. A. Schnabel, D. Rueckert, M. Quist, J. M. Blackall, A. D. Castellano-Smith, T. Hartkens, G. P. Penney, W. A. Hall, H. Liu, C. L. Truwit, D. L. G. Gerritsen, D. L. G. Hill, and D. J. Hawkes, A generic framework for nonrigid registration based on nonuniform multi-level free-form deformations, *Lecture Notes in Computer Science* **2208,** 573 (2001).
11. V. Walimbe, V. Zagrodsky, S. Raja, B. Bybel, M. Kanvinde, and R. Shekhar, Elastic registration of three-dimensional whole body CT and PET images by quaternion-based interpolation of multiple piecewise linear rigid-body registrations, *Proc. SPIE* **5370,** 119 (2004).
12. F. Maes, D. Vandermeulen and P. Suetens, Medical image registration using mutual information, *Proc. IEEE* **19,** 1699 (2003).
13. J. P. W. Pluim, J. B. A. Maintz and M. A. Viergever, Mutual information-based registration of medical images: A survey, *IEEE Trans. Med. Imaging* **22,** 986 (2003).
14. W. M. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis, Multi-modal volume registration by maximization of mutual information, *Medical Image Analysis* **1,** 35 (1996).
15. G. E. Christensen, M. I. Miller, M. W. Vannier, and U. Grenander, Individualizing neuroanatomical atlases using a massively parallel computer, *IEEE Computer* **29.** 32 (1996).
16. S. K. Warfield, F. A. Jolesz, and R. Kikinis, A high performance approach to the registration of medical imaging data, *Parallel Computing* **24,** 1345 (1998).
17. T. Rohlfing and C. R. Maurer, Non-rigid image registration in shared-memory multiprocessor environments with application to brains, breasts, and bees, *IEEE Trans. Information Technol. Biomedicine* **7,** 16 (2003).
18. F. Ino, K. Ooyama, A. Takeuchi, and K. Hagihara, Design and implementation of parallel nonrigid image registration using off-the-shelf supercomputers, *Lecture Notes in Computer Science* **2878,** 327 (2003).
19. C. R. Castro-Pareja, J. M. Jagadeesh and R. Shekhar, FAIR: A Hardware Architecture for Real-Time 3D Image Registration, *IEEE Trans. Information Technol. Biomed.* **7,** 426 (2003).
20. D. Rueckert, L. I. Sonoda, C. Hayes, D. L. G. Hill, M. O. Leach, and D. J. Hawkes, Nonrigid registration using free-form deformations: Application to breast MR images, *IEEE Trans. Med. Imaging* **18,** 712 (1999).
21. F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens, Multimodality Image Registration by Maximization of Mutual Information, *IEEE Trans. Med. Imaging* **16,** 187 (1997).
22. M. Otte, Elastic registration of fMRI data using Bezier-spline transformations, *IEEE Trans. Med. Imaging* **20,** 193 (2001).
23. C. Studholme, R. T. Constable and J. S. Duncan, Accurate alignment of functional EPI data to anatomical MRI using a physics-based distortion model, *IEEE Trans. Med. Imaging* **19,** 1115 (2000).
24. M. Doggett and M. Meissner, A memory addressing and access design for real time volume rendering, *Proc. IEEE International Symposium on Circuits and Systems, ISCAS '99*, IEEE Press, Los Alamitos, CA, 1999, p. 344.
25. D. K. Kostopoulos, An algorithm for the computation of binary logarithms, *IEEE Trans. Computers* **40,** 1267 (1991).
26. D. M. Mandelbaum and S. G. Mandelbaum, A fast, efficient parallel-acting method of generating functions defined by power series, including logarithm, exponential, and sine, cosine, *IEEE Trans. Parallel and Distributed Systems* **7,** 33 (1996).
27. J. Hormigo, J. Villalba and M. J. Schulte, A hardware algorithm for variable-precision logarithm, *Proc. of IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, IEEE Press, Los Alamitos, CA, 2000, p. 215.
28. H. Hassler and N. Takagi, Function evaluation by table look-up and addition, *Proc. of 12th Symposium on Computer Arithmetic*, 1995, p. 10.
29. S. L. SanGregory, C. Brothers, D. Gallagher, and R Siferd, A Fast, Low-Power Logarithm Approximation with CMOS VLSI Implementation, *Proc. 42nd Midwest Symposium on Circuits and Systems*, IEEE Press, Piscataway, NJ, 388 (1999).
30. W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, Cambridge, England, 1992.
31. M. J. D. Powell, *Approximation Theory and Methods*, Cambridge University Press, Cambridge, 1981.
32. *Tsunami PCI FPGA Procesor Technical Data Sheet*, Release version 1.2 ed. SBS Technologies, Waterloo, Ontario, Canada, 2003.
33. P. Thevenaz and M. Unser, Optimization of mutual information for multiresolution image registration, *IEEE Trans. Image Processing* **9,** 2083 (2000).
34. G. Hermosillo and O. Faugeras, Dense image matching with global and local statistical criteria: A variational approach, *Computer Vision and Pattern Recognition* **1,** 73 (2001).
35. J.-F. Mangin, C. Poupon, C. Clark, D. Le Bihan, and I. Bloch, Eddy-current distortion correction and robust tensor estimation for MR diffusion imaging, *Lecture Notes in Computer Science* **2208,** 186 (2001).
36. D. Sarrut and S. Clippe, Geometrical transformation approximation for 2D/3D intensity-based registration of portal images and CT scan, *Lecture Notes in Computer Science* **2208,** 532 (2001).
37. K. Cleary, H. Y. Chung, and S. K. Mun, OR 2020 workshop overview: operating room of the future, *Proc. 18th International Congress on Computer Assisted Radiology and Surgery*, vol. 1268, H. U. Lemke, M. W. Vannier, K. Inamura, Eds., Elsevier, Amsterdam, 2004, p. 847.