REAL-TIME EXTRACTION OF LOCAL PHASE FEATURES FROM VOLUMETRIC MEDICAL IMAGE DATA

Alborz Amir-khalili[⋆] *Antony J. Hodgson*[†] *Rafeef Abugharbieh*[⋆]

[⋆] Biomedical Signal and Image Computing Lab (BiSICL),
Department of Electrical and Computer Engineering; and
[†]Department of Mechanical Engineering, The University of British Columbia

ABSTRACT

We present a novel real-time implementation of local phase feature extraction from volumetric image data based on 3D directional (log-Gabor) filters. We achieve drastic performance gains without compromising the signal-to-noise ratio by pre-computing the filters and adaptive noise estimation parameters, and streamlining the remainder of the computations to efficiently run on a multi-processor graphic processing unit (GPU). We validate our method on clinical ultrasound data and demonstrate a 15-fold speedup in computation time over state-of-the art methods, which could potentially facilitate a wide range of practical applications for real-time image-guided procedures.

Index Terms— Local Phase Features, Real-Time, Segmentation, CUDA, GPU, Ultrasound, 3D Feature Extraction

1. INTRODUCTION

Image guided medical procedures commonly deployed in minimally-invasive surgeries and, more recently, in emerging robot assisted interventions are generating increasing demands for real-time data processing and analysis. With the steady improvement in spatial and temporal scanning resolution and the continuing increase in complexity of data analysis algorithms, real-time processing is becoming a challenging bottleneck that requires thorough consideration in most practical applications. Feature extraction from 3D images for critical tasks such as real-time segmentation and registration are prime examples. For instance, image features extracted from local phase information have recently been shown to be robust in computer aided orthopedic applications. In previous works [1] we demonstrated the effectiveness of phase symmetry (PS) features for segmentation and localization of bone fractures in 3D ultrasound. In an extension to this work we implemented a step towards clinical ultrasound guided intervention [2], in which we used PS features to register intra-operative ultrasound to pre-operative computed tomography (CT) images. Other local phase features such as phase-asymmetry (PA) and phase-congruency (PC) can be used to localize soft-tissue boundaries in imaging modalities such as ultrasound, CT, and magnetic resonance imaging [3].

In a previous work by Eklund et al. [4] a fast GPU implementation of phase feature extraction was proposed for the purpose of multimodal image registration. Although their approach performs in real-time, it suffers from being highly specialized within a registration framework which only considers phases features at a single scale in the three principal directions. In this paper, we present a novel and generalized tool for real-time extraction of local image phase features that deploys efficient pre-computations and GPU processing.

2. METHODOLOGY

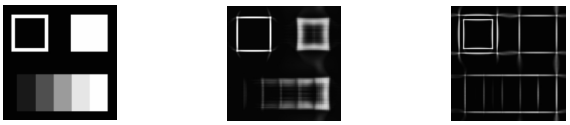
In most stages of local phase feature extraction, each pixel may be processed independently. The CUDA parallel programming model developed by Nvidia Corporation (Santa Clara, CA, USA) takes advantage of the hundreds of cores available on a GPU to run a computationally expensive algorithm in parallelized chunks, each operating independently on a separate stream processor of the GPU. The libraries designed for CUDA, which are included in its freely available Software Development Kit, also contain many fundamental tools for scientific computation, such as the fast Fourier transform (CuFFT).

There is strength in numbers, but a single core of a GPU is still significantly slower than a CPU. Furthermore, there is significant overhead associated with data transfer to and fro as well as sharing of information. In this section, we present the methods used in optimizing the implementation of local phase feature extraction to fully take advantage of GPU capabilities.

2.1. Local Phase Feature Extraction

Phase features are local, unitless measures that can be derived using quadrature odd and even (denoted o and e respectively) wavelet pairs [5]. Such features can be extracted across a set of different wavelengths (denoted s for scales) and at different angles in 3D space (denoted r for orientations). Currently available implementations of 3D local phase image feature

extraction based on log-Gabor filters generate the filters 'on-the-fly'. However, in this work, in order to speed up the algorithm we split the algorithm into two parts, one that can be computed off-line (pre-computed or templated at compile-time) and another that is on-line (computed live as images are acquired). Since the process of generating the filters is time-consuming, all our filters are generated off-line and stored in a filter-bank – as they can be generated independent of the images. Only filters requested for computation are loaded to GPU on demand (see subsection 2.2). These filters may be requested either by manual parameter setup or automatically using our parameterization method in [6].



(a) Original (b) PS (c) PA

Fig. 1. 2D cross-sections of synthetic 3D test pattern showing different local phase features at 2 scales and 3 orientations.

The on-line portion of the algorithm computes, for every image I and set of log-Gabor filters $[M^e_{s,r}, M^o_{s,r}]$, the responses to these filters, which can be represented as the following vectors: $[e_{s,r}, o_{s,r}] = [I * M^e_{s,r}, I * M^o_{s,r}]$. The magnitude of these vectors can be represented by $A_{s,r} = (e^2_{s,r} + o^2_{s,r})^{\frac{1}{2}}$. For the sake of speed, the convolution of the image with the filter pairs is done in the Fourier domain using element-wise multiplication (detailed in subsection 2.3).

Local phase features, such as PS and PA (Figure 1), are defined as the normalized thresholded combinations of filter pair responses, as follows:

$$PS(x,y,z) = \frac{\sum_{\forall s,r} \lfloor [|e_{s,r}(x,y,z)| - |o_{s,r}(x,y,z)|] - T_r \rfloor}{\sum_{\forall s,r} A_{s,r}(x,y,z) + \epsilon} \quad (1)$$

$$PA(x,y,z) = \frac{\sum_{\forall s,r} \lfloor [|o_{s,r}(x,y,z)| - |e_{s,r}(x,y,z)|] - T_r \rfloor}{\sum_{\forall s,r} A_{s,r}(x,y,z) + \epsilon} \quad (2)$$

where T_r is a threshold calculated from noise energy estimated for each filter orientation. Noise is assumed to be additive and Gaussian and is estimated from the noise response to the smallest scale filter pair (subsection 2.4). PS is ideal for localizing ridge-like features (empty box in Figure 1b) such as bone in ultrasound (Figure 3) and PA is used to define step-edges (full box and gradient in Figure 1c).

2.2. Filter Generation

Filters are uploaded to the GPU only if there is a significant change in the anatomy being imaged. This change may be defined in terms of rotational tolerance of the anatomy as a function of the standard deviation of angular filter components.

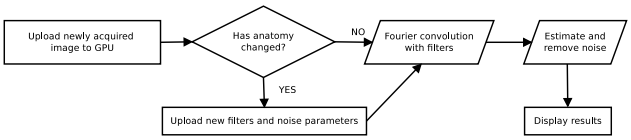


Fig. 2. Flow chart of methodology

(a) 3D rendered B-mode (b) Overlaid PS

Fig. 3. Bone surface extracted from 3D B-mode ultrasound scan of a hip phantom using PS at 2 scales and 3 orientations.

Data transfer rates between host RAM and GPU memory are a significant bottleneck. It takes on average 6.28 ms to transfer a volume with 128^3 double precision (64-bit) gray-scale voxels (approximately 16MB) from RAM to a professional-grade Nvidia Tesla C2050 video card. This transfer occurs twice per frame: the image captured from the device is transferred to GPU, processed and then transferred back to the host to be stored or displayed.

In Figure 3, a total of 6 filters (2 scales and 3 orientations) were required to produce the desired result. The filters are generated in the frequency domain and each filter is the same size as the image to allow for quick element-wise multiplication of the filter elements with the Fourier transformed image. Since the scale isolating components of the filters are essentially Gaussian radial band-pass filters, they can be generated separately from the angular (orientation related) components in the Fourier domain. The filters are combined to create different angular filters at each scale. In the shown examples only 5 $(s + r)$ filters need to be transferred to the GPU. If 3 scales and 4 orientations are desired, 7 filters would be transferred instead of 12.

In our implementation, all filters are transferred together to minimize overhead costs.Without using page-locked or pinned memory, the time it takes to transfer 6 filters together (96 MB) is 28 ms (effective speed: 3.3 GB/s) which is less than the time it takes to transfer them individually 6x6.28 ms (effectively 2.55 GB/s).

2.3. Fourier Transform

The transformation of the image into the Fourier domain is performed on the GPU using the Nvidia CuFFT libraries. After the image has been convolved (element-wise multiplication in Fourier domain) with every filter in parallel, the resultant products are concatenated into a 1D array and subjected to an inverse transform as a whole. The CuFFT libraries are

optimized to perform in a parallel manner. The performance gain of CuFFT increases with the size of the array being operated on and optimal performance is achieved when the image volume dimensions are a power of 2 (e.g. 128^3 voxels).

2.4. Noise Parameter Estimation

Noise parameter estimation of local phase features is critical for retaining a high signal-to-noise ratio in clinical applications such as accurate bone surface extraction [1]. The log-Gabor filters used in calculating the local phase features are very sensitive to noise, especially at small scales, since a high-frequency noise response is typically present at smaller wavelengths. Here we employ Kovesi's [7] original noise estimation technique with the following three assumptions: Noise is additive, the noise power spectrum is constant (white noise), and the features of interest only occur at isolated locations in the image.

In 3D, assuming that the smallest scale ($s = 1$) is predominantly noise, the noise signal g_r at a certain orientation r is estimated by evaluating the expected total energy of the image filter response at the smallest scale $A_{1,r}$ and dividing it by the expected total energy spectrum of the corresponding filter pair $M_{1,r}$. Denoting the Fourier Transform $\mathcal{F}(f) = \hat{f}$, as per Parseval's theorem, the magnitude of the noise spectrum is defined by:

$$|\hat{g_r}|^2 \simeq \frac{\mathbb{E}(A_{1,r}^2)}{\mathbb{E}(\hat{M}_{1,r}^2)}.$$

The numerical calculation of $\mathbb{E}(\hat{M}_{1,r}^2)$ of the filter itself is done off-line during the filter generation step. However, the total energy of $A_{1,r}$ has to be computed on-line. The expected value of $A_{1,r}^2$ is estimated, from its median response, in order to increase the performance of the algorithm. The magnitude of the energy vector $A_{1,r}$ will be primarily noise with a Rayleigh distribution, with some contamination as a result of image feature responses. Therefore the expected energy of $A_{1,r}^2$ may be characterized by a χ^2 distribution with 2 degrees of freedom. We can achieve a robust estimate by calculating the median of $A_{1,r}^2$ as:

$$\mathbb{E}(A_{1,r}^2) = \frac{-\text{median}(A_{1,r}^2)}{\ln(1/2)}.$$

The noise threshold T at orientation r is then calculated from this estimate. Let the variable k be a multiple of $\sigma_{R,r}$ (standard deviation of the Rayleigh distribution describing the noise energy response), typically ranging from 2 to 3. Then, $T_r = \mu_{R,r} + k\sigma_{R,r}$ (see Kovesi [7] for derivation) where:

$$\mu_{R,r} = \sigma_{G,r}\sqrt{\frac{\pi}{2}}; \quad \sigma_{R,r} = \sigma_{G,r}\sqrt{\frac{4-\pi}{2}}$$

$$\sigma_{G,r} = |\hat{g_r}|\sqrt{\mathbb{E}\left(\sum_{\forall s} M_{r,s}^2\right) + 2\mathbb{E}\left(\sum_{s_i<s_j}(M_{r,s_i}\cdot M_{r,s_j}\right)}$$

Computing the median of $A_{1,r}^2$ is time consuming. CUDA Thrust libraries are used to calculate the median value in this case as the GPU quickselect algorithm performs poorly on the GPU [8]. Even with the CUDA Thrust libraries, the noise estimation step is relatively costly compared to the other parts of the algorithm (see Figure 4).

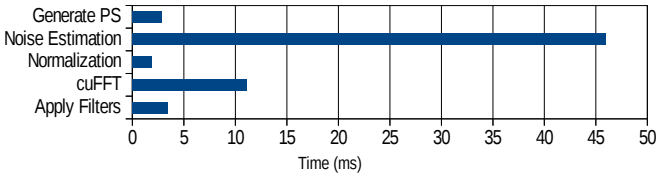


Fig. 4. Detailed breakdown of on-line runtime costs at 3 orients and 1 scale, running on Tesla c2050.

3. RESULTS AND DISCUSSION

To enable direct comparisons, the algorithm was prototyped in MATLAB as a benchmark, based on the code provided by Kovesi (available online). The results tabulated in Table 1 were obtained running MATLAB R2011b on a dual processor host machine with two Xeon x5472 CPUs @ 3GHz (Intel Corp., Mountain View, California, US) with 8GB of RAM using the same volume illustrated in Figure 3. Some functions in MATLAB such as FFT have been highly optimized to take advantage of all eight cores of the Xenon machine. A parallel CPU implementation using `parfor` and a pool of eight MATLAB workers performed three times slower than a standard implementation in our experiments.

The only other publicly available 3D implementation of local phase feature extraction by Hatt [9] (PS only C++ code) was then clocked on the same Xeon machine. In order to ensure fairness in comparison, the times reported in Table 1 do not include filter generation for all compared methods, including Hatt's. It is worth noting that the reported speed-ups over Hatt's algorithm are rather conservative as Hatt's implementation does not include the costly noise estimation step. The threshold used instead is set to a constant value passed into the algorithm as a filter parameter.

Our proposed algorithm was then ported to CUDA and executed on a host machine with a Tesla c2050 GPU. The results shown only include the on-line portion of the code. This includes: loading a single volume from host to device, FFT convolution, noise estimation, and PS calculation. The time it takes to transfer the filters is not included as this operation does not occur frequently.

4. CONCLUSIONS AND FUTURE WORK

We proposed a novel efficient implementation for real-time extraction of local phase features from volumetric images. We also presented quantitative validation on real 3D medical

Table 1. Averaged double precision runtimes (mean of 20 trials) of our algorithm (MATLAB and CUDA) compared to Hatt [9]

Orientations	Scales	Benchmark MATLAB (CPU)	Hatt C++ (CPU)	Proposed Algorithm CUDA (GPU)	Speed-up factor CUDA/MATLAB	Speed-up factor CUDA/Hatt
$r = 1$	$s = 1$	430 ms	674 ms	30 ms	14.3	22.5
	$s = 2$	625 ms	949 ms	36 ms	17.4	26.4
	$s = 3$	787 ms	1204 ms	42 ms	18.7	28.7
	$s = 4$	964 ms	1450 ms	48 ms	20.1	30.2
$r = 2$	$s = 1$	639 ms	993 ms	52 ms	12.3	19.1
	$s = 2$	989 ms	1472 ms	66 ms	15.0	22.3
	$s = 3$	1270 ms	2086 ms	78 ms	16.3	26.7
	$s = 4$	1609 ms	2545 ms	90 ms	17.9	28.3
$r = 3$	$s = 1$	918 ms	1294 ms	74 ms	12.4	17.5
	$s = 2$	1396 ms	2049 ms	96 ms	14.5	21.3
	$s = 3$	1851 ms	2787 ms	116 ms	16.0	24.0
	$s = 4$	2306 ms	3556 ms	136 ms	17.0	26.1
$r = 4$	$s = 1$	1198 ms	1523 ms	97 ms	12.4	15.7
	$s = 2$	1825 ms	2655 ms	127 ms	14.4	20.9
	$s = 3$	2447 ms	3646 ms	155 ms	15.8	23.5

data demonstrating that our CUDA implementation results in a minimum of 12-fold speedup over our MATLAB benchmark and 15-fold speedup over Hatt's [9] implementation.

The performance of our algorithm is expected to increase in the future as it scales very well the hardware capabilities of the GPU device. Figure 5 provides a comparison between the MATLAB benchmark, an entry level GPU (Nvidia NVS 5400M mobile graphic card with 1GB of memory), and a professional GPU (Nvidia Tesla c2050). Both of these GPUs are based on Nvidia's previous hardware architecture, Fermi. Nvidia is reporting a 3-fold increase in single-precision floating point performance per Watt of power with their new Kepler architecture. This algorithm will perform considerably faster on the next generation of Kepler based Tesla cards, expected to become available by the end of 2012.

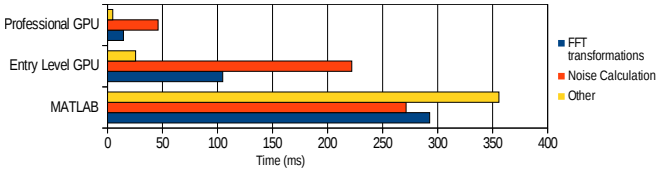


Fig. 5. On-line runtime of PS with 3 orientations and 1 scale.

We will extend our work by incorporating the automatic parameterization methods [6] in real-time such that they can both be implemented within our Ultrasound to CT registration framework [2].

5. REFERENCES

[1] I. Hacihaliloglu, R. Abugharbieh, A.J. Hodgson, and R.N. Rohling, "Bone surface localization in ultrasound using image phase-based features," *UMB*, vol. 35, no. 9, pp. 1475–1487, 2009.

[2] A. Brounstein, I. Hacihaliloglu, P. Guy, A.J. Hodgson, and R. Abugharbieh, "Towards real-time 3D US to CT bone image registration using phase and curvature feature based GMM matching," *MICCAI*, pp. 235–242, 2011.

[3] A. Wong and W. Bishop, "Efficient least squares fusion of MRI and CT images using a phase congruency model," *Pattern Recognition Letters*, vol. 29, no. 3, pp. 173–180, 2008.

[4] A. Eklund, M. Andersson, and H. Knutsson, "Phase based volume registration using CUDA," in *ICASSP*. IEEE, 2010, pp. 658–661.

[5] P. Kovesi, "Symmetry and asymmetry from local phase," *Tenth Australian Joint Converence on Artificial Intelligence*, pp. 2–4, 1997.

[6] I. Hacihaliloglu, R. Abugharbieh, A. Hodgson, and R. Rohling, "Automatic data-driven parameterization for phase-based bone localization in US using log-gabor filters," *Advances in Visual Computing*, pp. 944–954, 2009.

[7] P. Kovesi, "Image features from phase congruency," *Videre: Journal of Computer Vision Research*, vol. 1, no. 3, pp. 1–26, 1999.

[8] G. Beliakov, "Parallel calculation of the median and order statistics on GPUs with application to robust regression," *arXiv preprint arXiv:1104.2732*, 2011.

[9] C. Hatt, "Multi-scale steerable phase-symmetry filters for ITK," 2012.