Processing and rendering of Fourier domain optical coherence tomography images at a line rate over 524 kHz using a graphics processing unit

Janarthanan Rasakanthan,[a] Kate Sugden,[a] and Peter H. Tomlins[b]
[a]Aston University, Photonics Research Group, Department of Electronic Engineering, Birmingham, B4 7ET, United Kingdom
[b]Queen Mary, University of London, Barts & The London School of Medicine and Dentistry, Turner Street, London, E1 2AD, United Kingdom

Abstract. In Fourier domain optical coherence tomography (FD-OCT), a large amount of interference data needs to be resampled from the wavelength domain to the wavenumber domain prior to Fourier transformation. We present an approach to optimize this data processing, using a graphics processing unit (GPU) and parallel processing algorithms. We demonstrate an increased processing and rendering rate over that previously reported by using GPU paged memory to render data in the GPU rather than copying back to the CPU. This avoids unnecessary and slow data transfer, enabling a processing and display rate of well over 524,000 A-scan/s for a single frame. To the best of our knowledge this is the fastest processing demonstrated to date and the first time that FD-OCT processing and rendering has been demonstrated entirely on a GPU. [DOI: 10.1117/1.3548153]

Keywords: optical coherence tomography; graphics processing unit; quadratic interpolation resampling; wavelength to wavenumber resampling.

Paper 10532LRR received Oct. 2;, 2010; revised manuscript received Jan. 4, 2011; accepted for publication Jan. 6, 2011; published online Feb. 22, 2011.

Optical coherence tomography (OCT)[1] is a noninvasive, noncontact technique which enables high resolution cross-sectional images of tissue structures to be obtained. OCT systems fall into two categories: time domain OCT (TD-OCT) and Fourier domain FD-OCT.[2,3] FD-OCT records the depth profile by detecting the interference signals over a broad bandwidth of light using a spectrometer.[3] Notable advantages of FD-OCT over TD-OCT are an improved signal to noise ratio (sensitivity) and higher imaging speeds.[4,5] FD-OCT is capable of dynamically recording detailed three-dimensional information. The disadvantage of FD-OCT is that it requires significantly more data processing to handle the large amount of information recorded and historically this has incurred a significant time overhead. Therefore, the use of graphics processing units (GPUs) for OCT data processing is of interest since they are optimized for high parallelism and memory bandwidth. GPUs require parallel programming algorithms to exploit their performance power, for which specific programming platforms are available. CUDA[6] (Computer Unified Device Architecture) is NVIDIA's (Santa Clara, California) parallel programming platform, which can be used to program their GPUs for general purpose computation.[6]

Watanabe and Itagaki[7] used a GPU (NVIDA GeForce GTX 280) with a linear-in-wavenumber spectrometer to achieve a processing speed of 55,800 lines/s. Subsequently, Zhang and Kang[8] used linear-spline interpolation and a GPU (NVIDIA Quadro FX5800) to accelerate the processing and rendering of OCT data from a nonlinear *k*-space spectrometer. They reached a processing rate of 680,000 lines/s for overall volume processing. However, for smaller A-scan numbers the processing bandwidth decreased to just greater than 200,000 lines/s for the range of 1,000 to 10,000 A-scans. Van der Jeught et al.[9] compared nearest-neighbor, linear, and cubic-spline interpolation to resample algorithms running on a GPU (Geforce 9800GT) and achieved a processing speed of 25,600 lines/s. However, progress to date has not explicitly exploited the different GPU memory types available or combined both GPU processing and rendering capabilities to increase efficiency and speed. Here we employ the use of paged memory to increase the data transfer rate from the CPU (host) to the GPU (device). Furthermore, OCT image data is rendered on the GPU to increase overall speed, rather than copying the data back to the CPU. All algorithms are optimized for the specific GPU used.

In this study a GPU (NVIDIA Tesla C1060, 1.3 GHz) with 30 streaming multiprocessors (SM) was used. This is identical to that used by Zhang and Kang[8] with the exception that it does not include a hardware graphics display outlet. Each SM has 8 scalar processor cores making a total of 240 processors. The GPU has 4 GB dedicated global memory and 16 KB shared memory per SM. The card is hosted in a PC workstation (Intel Core i7, 2.67 GHz, 6 GB RAM) running the Ubuntu 9.10 Linux operating system. The custom programs were written in C using NVIDIA's CUDA programming extensions and executed with the CUDA 3.0 driver and tools.[6]

Test data sets were acquired using a commercial OCT system (EX1301, Michelson Diagnostic, UK) at the National Physical Laboratory, UK. This OCT system contains a swept source operating at a center wavelength of 1305 nm. It has four interferometric channels focusing on different depths to obtain an extended depth of field. For simplicity this analysis was carried out on a single channel of data but is equally able to process all four channels. OCT images were taken from an optical test phantom that was created with a femtosecond laser. Details of this fabrication technique are given elsewhere.[10] The phantom contained a 3-D structure of lines written at different depths and lateral positions within a silica substrate.

Figure 1 shows a flow chart of the OCT data processing methodology. The acquired data from the OCT system was saved on the PC hard-drive (16-bit integer format) prior to processing. The raw data comprised four components: the source reference spectrum S_j, mean A-scan noise floor η_j, resample index table $J_{j,}$ and the OCT interference signal $I_{i,j}$, where the subscripts j and i represent the array indices for spectrally detected data points and individual A-scan numbers, respectively.

Address all correspondence to: Janarthanan Rasakanthan, Tel. 00447988682011; Fax: 07988682011; E-mail: rasakanj@aston.ac.uk, ra.jana@yahoo.co.uk.

1083-3668/2011/16(2)/020505/3/$25.00 © 2011 SPIE

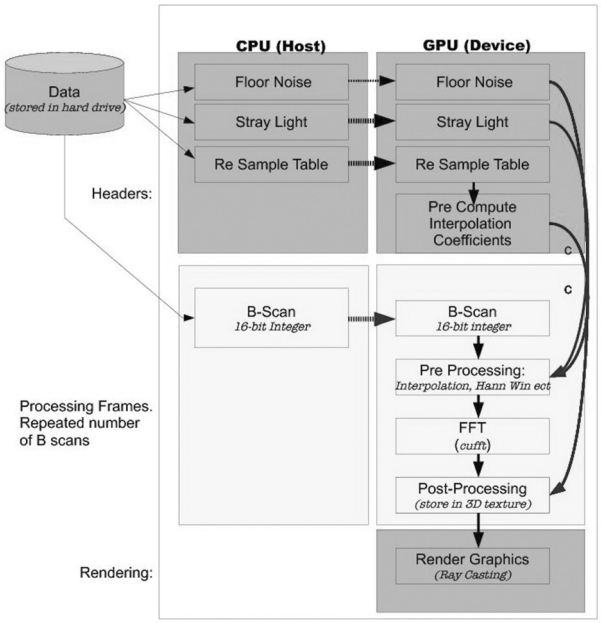

Fig. 1 OCT processing flow chart for GPU-CPU hybrid.

The reference, noise, and resampling data values were read directly from file and copied to GPU memory. The resample indices were used to pre-compute interpolation coefficients M_{m_j}, which were stored in the GPU global memory. A third-order Lagrange interpolating polynomial was used to resample the interference data from discrete measurements at j to linear points in wavenumber space[2] J, where the resampled interference signal, $I'_{i,j}$, is given by:

$$I'_{i,j} = \sum_{m=0}^{3} M_{m,j} I_{i,j+m-2}, \tag{1}$$

where the interpolation coefficient is:

$$M_{m,j} = \prod_{n=0,n\neq m}^{3} \frac{J_j - j - n + 2}{m - n}. \tag{2}$$

Interpolation coefficients were computed once for each OCT volume data set. To generate OCT images, the OCT spectral interference signals corresponding to each A-scan were loaded sequentially from the file into the host memory and then transferred into the GPU global memory, essentially mimicking data acquisition from a frame grabber. Each OCT spectrum was preprocessed by subtracting the reference spectrum, resampling using Eq. (1), and multiplying by a Hann window function to reduce noise. This step was implemented as a CUDA kernel that executed on the GPU, processing multiple A-scans simultaneously. Following the pre-process, a real to complex fast Fourier transform (FFT) of each A-scan was computed using the CUFFT kernel (a dedicated FFT kernel available with CUDA).[6] A custom post-processing kernel then determined the modulus from all of the Fourier transforms and stored the result in GPU memory as 3-D texture array (cached global memory). Once the entire data volume was processed, the data from global memory was rendered using the CUDA ray-casting (volume rendering) method, which avoids copying back to host memory and the associated time overhead. The processed data volume was also copied back to the host memory for storage. The code was optimized by the use of shared memory, constant memory, and registers. All GPU computations were carried in single precision, which is natively supported by the present device.

To test the speed of this process, we used two OCT data sets obtained from the test phantom and recorded the processing time for individual processes using CUDA events.[5] Both data sets comprised 1000 B-scans with an A-scan length of 1024 pixels. However, the number of A-scans per B-scan differed, with samples 1 and 2 containing 240 and 1836 A-scans per B-scan, respectively. The optimal A-scan batch size was investigated by arranging 200 A-scans as frames, processing multiple frames together in batches to understand processing improvement with increased A-scans. To confirm that the data was not corrupted by this approach, we validated the CUDA results by comparing them with the output of MATLAB code using cubic-spline interpolation function.

The parallel preprocessing algorithm was carefully optimized for the specific GPU capability. For example, our GPU is limited to a maximum of eight thread blocks per SM. The number of threads per block is limited by the size of the processor core shared memory and available registers, with the total number of threads per SM limited to a maximum of 1024. For the present work, a block size of $16\times16=256$ threads was found to be optimal, allowing 4 blocks to run on each of the 30 SMs. In our configuration, each block was used to simultaneously resample 16 data points from 16 different A-scans (256 data points in total). The whole A-scans were distributed across 64 blocks, enabling parallel resampling of 16 complete A-scans. The remaining 56 blocks were used to resample partial A-scans and thus maximize processor utilization. To maximize speed the A-scan data was loaded from global to shared memory along with the corresponding 16 resampling coefficients, from where it was operated on by the preprocessing kernel. Once the resampling operation was completed for all data points within a block, the next 16 points were loaded for each A-scan. Hence, 4 blocks were required to load an entire A-scan consisting of 1024 elements. We were therefore able to simultaneously preprocess 120 A-scans.

The recorded processing times were averaged over 500 runs and 1000 frames and are given in Table 1 for both data sets. The total time for processing a single frame for sample 1 and 2 is about 0.63 and 3.50 ms, which gives a processing rate of 377,768 and 524,205 lines/s for samples 1 and 2, respectively. It is clear that sample 2, with the larger number of A-scans, shows a higher processing rate. Our result indicates that the interpolation coefficients were calculated in about 0.03 ms, although these are only computed once per data set. Preprocessing took approximately 0.20 and 1.24 ms per B-scan of samples 1 and 2 respectively.

Copying back to CPU after processing took about 0.25 ms per frame for sample 1 and about 1.00 ms per frame for sample 2. To avoid this time overhead we employed a method available in CUDA[3] to render the final result from the GPU, which eliminates the need for copying the data back to host (CPU) and hence saves a considerable amount of time. The approximate rendering time for a sample of size $1{,}024\times1{,}836\times1{,}000$ is about 3.50 ms.

To facilitate qualitative comparison, Fig. 2 shows a B-scan from sample 1, processed using (a) MATLAB cubic-spline interpolation and (b) our CUDA algorithm. The images are visually

Table 1 Time for different processes per frame: these data are averaged over 1000 frames.

Process	Sample 1 time (ms)	Sample 2 time (ms)
Frame size	1024*240	1024*1836
Quadratic coefficient	0.028	0.028
Copy data to device (GPU)	0.183	0.860
Preprocessing	0.198	1.237
CUFFT	0.356	1.390
Modulus	0.050	0.269
Copy result to host (CPU)	0.250	0.969
Frame	0.640	3.502
Processing rate without copying back to the CPU	377,768	524,205

identical, confirming that the algorithm faithfully reproduces the expected OCT image. Figure 3 shows that as the number of A-scans per frame increases as the processing time decreases, although the processing rate plateaus around 694,330 lines/s for A-scan batch sizes between 3500 and 4500. This also shows that the maximum processing speed of the card can be reached more quickly than demonstrated elsewhere using comparable hardware.[7] This is achieved because our processing exploits all of the different types of memories available in the GPU effectively.

In conclusion, the use of GPUs enables real-time processing and visualization of OCT data. To maximize the

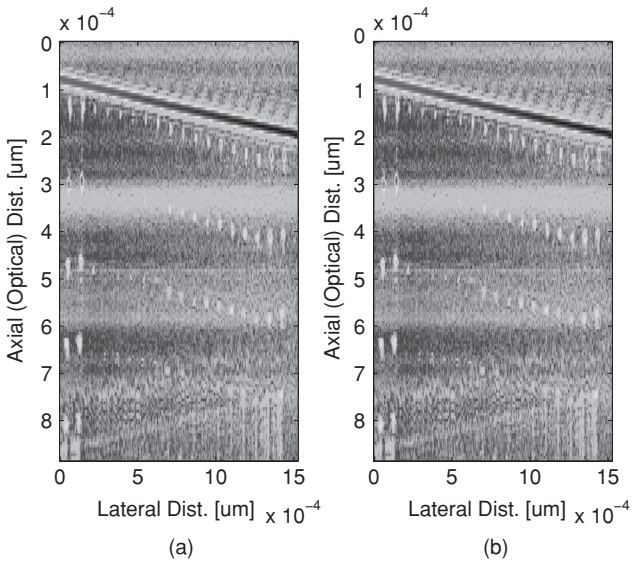

Fig. 2 (a) Comparison of image quality of B-scan data processed using built-in MATLAB spline interpolation and (b) our CUDA processor.

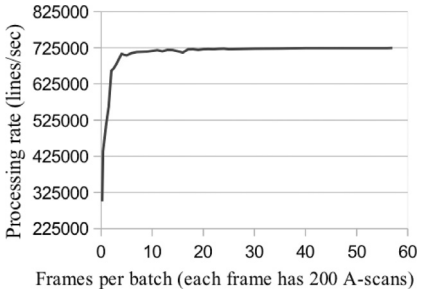

Fig. 3 A-scan batch processing rate for up to 20,000 A-scans.

speed of this approach we have used GPU paged memory to increase the data transfer rate from the CPU to the GPU and rendered the data on the GPU rather than copying to the CPU. We found that CPU to GPU transfer takes nearly 18% of the total processing time per frame and CPU to GPU transfer takes 20% of the total processing time per frame. Therefore avoiding unnecessary transfer improves the performance and rendering time significantly. Data movement between host and device memory is a comparatively slow process and should be minimized to achieve high performance. By taking this into account, we achieved processing rates over 524,205 lines/s for a B-scan with 1836 A-scans. Additionally, we showed that a processing rate greater than 724,314 lines/s is achievable for batch processing. There is still room for enhancing the processing rate by further optimizing our CUDA code for the particular GPU in use.

Acknowledgments

We would like to acknowledge the assistance of Graham Smith from Aston University in providing the test OCT artifact used in this study and the financial support of NPL via a CASE studentship.

References

1. D. Huang et al., "Optical coherence tomography," *Science* **254**, 1178–1181 (1991).
2. A. F. Fercher, C. K. Hitzenberger, G. Kam, and S. Y. El-Zaiat, "Measurement of intraocular distances by backscattering spectral interferometry," *Opt. Commun.* **117**, 43–48 (1995).
3. A. F. Fercher, W. Drexler, C. K Hitzenberger, and T. Lasser, "Optical coherence tomography–principle and application," *Rep. Prog. Phys.* **66**, 239–303 (2003).
4. P. H. Tomlins and R. K. Wang, "Theory, developments and applications of optical coherence tomography," *J. Phys. D* **38**(15), 2519–2535 (2005).
5. R. Leitgeb, C. Hitzenberger, and A. Fercher, "Performance of Fourier domain vs. time domain optical coherence tomography," *Opt. Express* **11**, 889–894 (2003).
6. NVIDIA CUDA Zone: http://www.nvidia.com/object/cuda_home_new.html.
7. Y. Watanabe and T. Itagaki, "Real-time display on Fourier domain optical coherence tomography system using a graphics processing unit," *J. Biomed. Opt.* **14**(6), 060506 (2009).
8. K. Zhang and J. U. Kang, "Real-time 4D signal processing and visualization using graphics processing unit on a regular nonlinear-k Fourier-domain OCT system," *Opt. Express* **18**(11), 11772–11784 (2010).
9. S. Van Der Jeught, A. Bradu, and A. G. Podoleanu, "Real-time resampling in Fourier domain optical coherence tomography using a graphics processing unit," *J. Biomed. Opt.* **15**(3), 030511 (2010).
10. P. H. Tomlins, P. D. Woolliams, G. Smith, J. Rasakanthan, and K. Sugden, "Femtosecond laser micro-inscription of optical coherence tomography resolution test phantoms," submitted to *Opt. Express* (2010).