Challenges in using GPUs for the reconstruction of digital hologram images

I D Reid, J J Nebrensky and P R Hobson

Centre for Sensors & Instrumentation, School of Engineering & Design, Brunel University, Uxbridge UB8 3PH, UK

E-mail: ivan.reid@cern.ch.

Abstract. In-line holographic imaging is used for small particulates, such as cloud or spray droplets, marine plankton, and alluvial sediments, and enables a true 3D object field to be recorded at high resolution over a considerable depth. To reconstruct a digital hologram a 2D FFT must be calculated for every depth slice desired in the replayed image volume. A typical in-line hologram of ~100 micrometre-sized particles over a depth of a few hundred millimetres will require O(1000) 2D FFT operations to be performed on an hologram of typically a few million pixels. In previous work we have reported on our experiences with reconstruction on a computational grid. In this paper we discuss the technical challenges in making efficient use of the NVIDIA Tesla and Fermi GPU systems and show how our reconstruction code was optimised for near real-time video slice reconstruction with holograms as large as 4K by 4K pixels. We also consider the implications for grid and cloud computing approaches to hologram replay, and the extent to which a GPU can replace these approaches, when the important step of locating focussed objects within a reconstructed volume is included.

1. Introduction

In the last decade digital in-line holography has replaced silver-halide based techniques for the 3D recording of volumes of small particles. It has applications in a number of areas of science and engineering, for example cloud or spray droplets [1], marine plankton [2] and sediments [3]. Digital holography replaces the older technique, which primarily used high-resolution silver-halide emulsions, with direct recording onto a solid-state pixel sensor such as a CCD or CMOS array. There are a number of different mathematical techniques that can be used to calculate the replayed image volume; ours is based upon the convolution approach [4]. To replay the image volume encoded by a digital hologram requires that the two-dimensional discrete Fourier transform of the hologram is multiplied by a phase factor related to the recording beam and the distance of the replaced depth slice and then the image is created by taking the inverse discrete Fourier Transform and calculating its magnitude. A typical CCD or CMOS sensor used in digital holography has a few million pixels, although individual sensors with over eighty million pixels have been made. To fully reconstruct the volume recorded on a digital hologram requires 2D depth-slices at sub-millimeter intervals to be reconstructed over the entire recorded depth which is often a few hundred millimeters in extent. Thus for a single hologram

of order one thousand individual 2D images will be replayed. The computational task, even with modern multi-core processors and efficient discrete Fourier transform algorithms such as FFTW [5] is considerable. In previous work [6], [7] we have exploited the trivially parallel nature of the hologram replay to enable us to use computational grid resources. Unfortunately the stochastic nature of submission to grid resources, via a broker, and the possibility of being queued until the validity of a job's X.509 proxy certificate expires means that the initial significant speed advantage over a single local computational node is rarely maintained and quite often not all of a volume is reconstructed. In earlier work [8] a number of different jobs each reconstructing ten, fifty or one hundred sequential depth slices were submitted to nodes on the EGEE grid. The total number of depth planes required to fully reconstruct the volume was 2200. Figure 1 illustrates the initial large advantage over a single processor but the rate in all cases dramatically slowed and only a fraction of all the jobs completed. This behavior is typical of current scientific production grids.

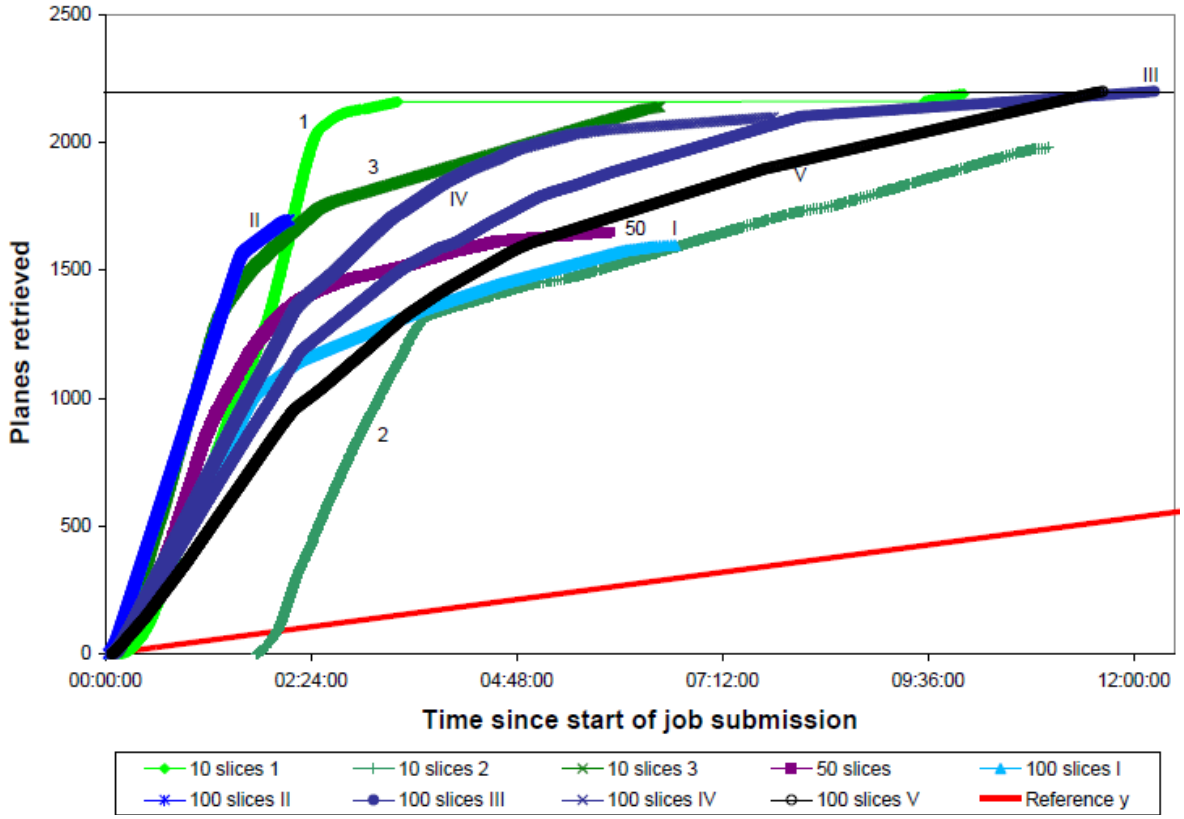

Figure 1. Overall rate of grid job completion (times in hh:mm:ss format after submission). The straight line labelled "Reference y" refers to serial replay on a single processor (AMD Athlon XP model 10 2600+ (1920 MHz), 1.5 GB memory) [8].

With the advent in recent years of powerful and increasingly inexpensive GPU chips considerable effort from ourselves [9] and many other researchers (see for example [10], [11], [12]) has been expended in porting, or developing, digital hologram replay code to run efficiently on these systems. In this paper we describe our latest work using NVIDIA Tesla and Fermi class GPU systems, the optimization techniques that we have deployed to achieve very significant replay speed, and the

prospects for further improvement. Finally we speculate on whether a hybrid approach of GPU and distributed (grid or cloud) based computing may be optimum when the task of both replay and object extraction is considered.

2. Recording and replaying a digital hologram

We consider in this paper only the recording and replay of in-line (Fraunhofer) holograms recorded digitally on a pixel area sensor. Other forms of digital hologram, for example those recorded in off-axis holography, do not differ in the essential principles of replay.

2.1. Hologram recording

We recorded in-line (Fraunhofer) holograms of objects in water using an 8 megapixel camera (Atmel Camelia 8M, 2300 by 3500 pixels with 12-bit depth) with a collimated beam from a c.w. HeNe laser (λ=633 nm, 1 mW). As in earlier work [9] we recorded a sample volume consisting of cenospheres mostly of 100-300 µm dia. (Fillite Trelleborg Specialty Grade (High Alumina) SGHA 500) dispersed in a water tank (figure 2). A variety of other subjects, for example air bubbles in water have also been used as test objects.

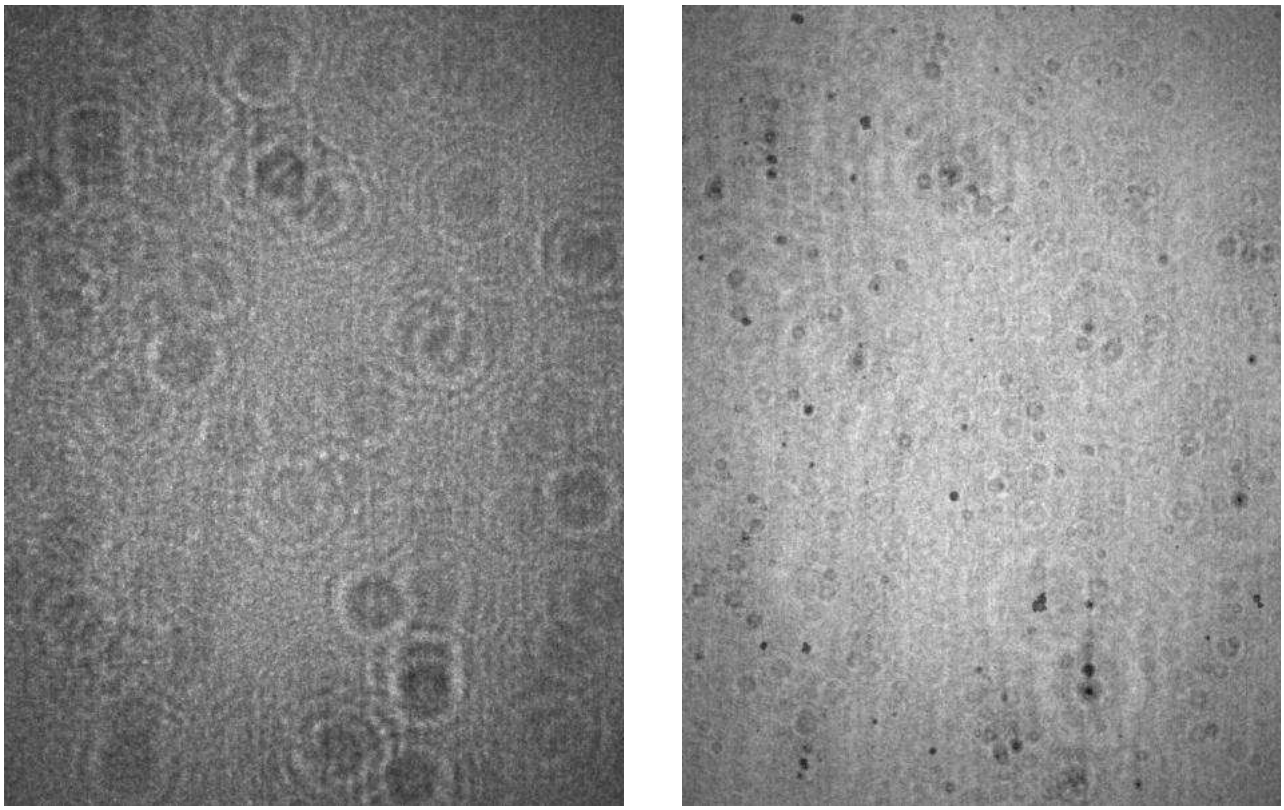

Figure 2. (Left): An extract (10 mm wide) from a digital hologram of cenospheres in water. (Right): The region of a numerically-reconstructed arbitrary plane through the sample volume. The smallest particles imaged here are approximately 30 micrometres in size [8].

2.2. Hologram replay

Our general approach is based on the convolution method [4] and can be summarised as follows:

- Numerically multiply the (zero-padded) hologram by a complex amplitude reference wave (in the case of collimated beam in-line holography the reference wave is 1+0i).
- Perform a 2D discrete Fast Fourier transform (FFT); real-valued data is transformed to complex-valued data.
- Multiply the result by a transfer function in the frequency domain based on the Rayleigh-Sommerfeld equation. The transfer function depends on the wavelength and the distance of the desired depth plane from the recording plane.
- Perform an inverse 2D FFT to generate the reconstructed image for a single depth plane. Extract the magnitudes, remove the zero-padding and normalise intensity.

This process is computationally heavy as it requires multiple 2D complex FFT to be computed. Regarding only the reconstruction of a digital hologram, and ignoring the challenging and similarly computationally intensive task of locating "regions of interest" (ROI) within the replayed volume, it can be seen that every depth slice may be calculated in parallel. Since the initial Fourier transformed hologram is common to any depth slice calculation there are clearly potential advantages in doing this step once and making the transformed data available to all processors. With the recent availability of very powerful GPU processors and the porting of the FFTW algorithm to the NVIDIA CUDA library, most recent work in the digital holography field has made use of such GPU based systems. In the next section we describe in detail our work on porting our existing replay code *HoloReco* [13] to the NVIDA CUDA environment, the refactoring needed to generate significant replay speeds on commodity GPUs and some analysis of the remaining bottlenecks and the prospects for further improvement.

3. Code refactoring for the NVIDIA CUDA environment.

The original *HoloReco* code was written to carry out all of the four steps outlines in subsection 2.2 for every depth slice. Carrying out the first FFT and then using it subsequently for a series of different depth slices eliminates all but one of the time intensive serial reads from host memory to GPU memory. The most significant improvement to the processing speed was finding a way to parallelise the transfer function (unique to each depth plane). After many code rearrangements, a way to parallelise the inner loops of the transfer function was realised. This allowed the results of the forward FFT to be kept in the GPU memory for the transfer function step and reverse FFT, thus no data movement is needed until the output stage. Some further optimization of the input file format (PGM) and the power spectrum and intensity normalization steps resulted in a very significant improvement in computational speed compared to a conventional processor.

3.1. Tesla based system with no video display

Using a system containing three NVIDIA Tesla S1070 (240 thread processors per GPU core, shaders clocked at 1.30 GHz) we were able to process our hologram at 13.6 frames/second. This should be compared to our original code running on an eight-core Xeon E5420 @ 2.50GHz based system where a speed of only 0.057 frames/second was achieved. Thus code refactoring and using three GPUs provided a factor of nearly 240 improvement in reconstruction speed. These results used V2.2 of the FFTW libraries for CUDA. It might be worth noting that at the time this work was done (2009) the relative speed per pound Sterling spent on the hardware was in favour of the GPU based system by only about a factor of six.

3.2. Tesla and Fermi based systems with video display

There are two clearly different user cases for hologram data visualisation:

- An expert user (e.g. a Marine Biologist) searches through a video stream of replayed hologram depth slices looking for “interesting” objects. This is a very good use of the expert *until they become bored!*
- An image-processing system searches through replayed hologram image slices looking for “objects”. The set of detected objects is then sent to classifier.

A better use of the human expert is for them to locate quickly holograms that contain some objects of interest, for example a zooplankton species, and then to create a selected list of holograms to be systematically replayed in full and processed by an image-processing system and object classifier. Alternatively the expert user could look, in context, at those regions of the replayed volume in which an automated system had previously located and classified objects. Some Tesla and many Fermi-class GPUs have a video output and an OpenGL compatibility library is available. For human visualisation these are ideal since one can efficiently copy (using **cudaMemcpy()**) the replayed hologram to the video buffer and the overall performance is not compromised. The sequence of events for each replayed frame is shown in figure 3.

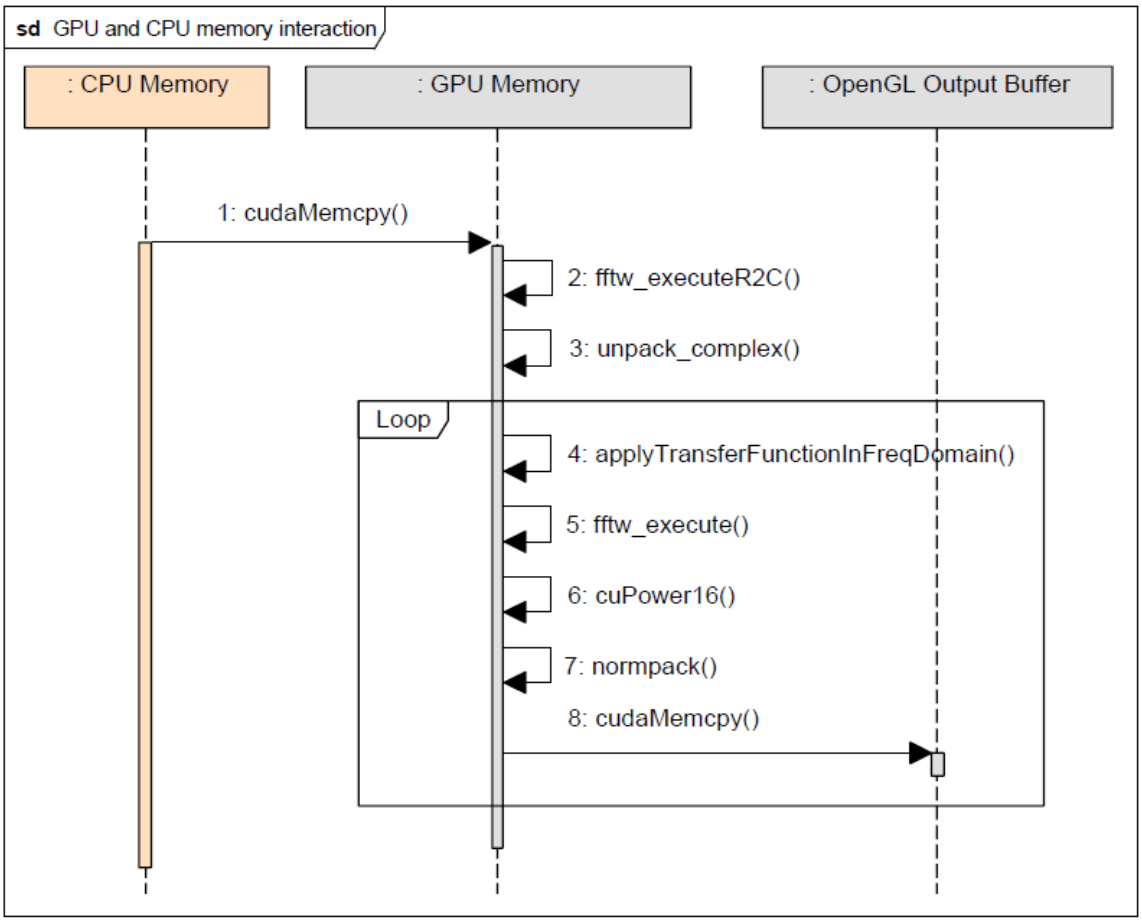

Figure 3. A UML representation of the sequence of key events in our software that occur during the processing of a digital hologram. If a number of frames from the same hologram are replayed to video then the sequence numbers 4 through 8 are repeated as often as required.

Table 1 lists the video rates we achieved for two different sizes of hologram using a Tesla C2070 (448 thread processors in one GPU core, shaders clocked at 1.15 GHz); CUDA 4.0; Driver 285.03; in a Xeon 5620 @ 2.40 GHz, Ubuntu 11.04.

Table 1. Frame rates for two different hologram sizes and an NVIDIA C2070 GPU.

Hologram size	Padded size	Video replay rate (frames/s)
2300×3500	4096×4096	15.8
2048×1389	2048×2048	41.2

3.3. Profiling the code on a Tesla GPU

The CUDA SDK also includes a profiler for mapping GPU usage and this is a powerful tool for locating and understanding which operations dominate the GPU processor time. Unfortunately the GUI version turned out to be temperamental on our systems; because of this we ran the CLI version to get a timestamp snapshot of the programme running for 15 seconds on a 4-Core Xeon computer running Ubuntu 11.04 and equipped with a Tesla C2070 GPU. This gave the start-up and 50 iterations of the display loop (at ~4.4 frames/s; without the profiler it ran at ~16 frames/s). The hologram was 3500x2300 and zero-padded to 4096x4096 for processing. Table 2 shows information on GPU usage for this run.

Table 2. (a) GPU and CPU procedure call timings and GPU occupancy for the initialisation step common to all slices. (b) GPU and CPU timings and GPU occupancy, averaged over 50 iterations, of the procedure calls that are executed for each depth slice calculated and then displayed. Note that there are two occupancy figures for the FFT because there are two separate kernels executed for each FFT; one executes once, the other twice.

(a) GPU kernel	Number of calls	GPU time (µs)	CPU time (µs)	GPU occupancy	Comment
memset32_post	3	6.7	27	0.042	
memcpyHtoD	1	12203	12313		Copy data from Host to Device
Forward FFT operations	3	5011	36	0.667/0.417	
Post-process	1	2379	5	1.000	
memcpyHtoD	1	2.0	7		Parameters for unpacking
copy_kernel	4095	11123	18524	1.000	cudaBlas unpacking FFT
insert_conj	1	15609	4	0.167	Reconstruct 2nd half of FFT
setUpX22_kernel	1	421.1	9	0.021	Calculate a lookup table for transfer

(b) GPU kernel	Number of calls/loop	Average GPU time/loop (µs)	Average CPU time/loop (µs)	Average GPU occupancy	Comment
doInnerLoop	4096	20780	19891	0.167	Parallelized Transfer Function

Reverse FFT operations	3	9898.9	26.0	0.667/0.333	
cuPower16_kernel	1	22333	4.6	0.500	Extract amplitudes from complex & remove zero-padding
memcpyDtoH	1	2.6	40.1		Return subroutine results
memcpyDtoH	1	2.1	33		Return subroutine results
memcpyHtoD	1	1.6	5.4		Parameters for minmax to GPU
thrust::minmax_element	2	516.0	11.7	0.500	Find min & max
memcpyDtoH	1	2.0	539.0		Return subroutine results
setminmax	1	4.3	6.5	0.021	Place scaling factors in device
packfloat	1	5128.4	4.4	0.167	Create scaled, packed pixels
memcopyDtoD	1	272.4	7.8		Copy to OpenGL buffer

The timing data show that, once the initial file is read, the majority of the GPU time is spent manipulating data – performing the transfer function to prepare for the reverse FFT and converting back from the complex transform to real magnitudes. While we have profiled our code, we have yet to use these profiling data in our code optimisation; for example the GPU warp occupancy figures in table 2 imply that further refactoring to increase the occupancy could improve performance.

4. Discussion

With our refactored code achieving a replayed frame rate approaching standard video for holograms as large as 4096×4096 pixels and significantly exceeding it for 2048×2048 pixel holograms, there is no doubt that for visualisation a GPU is far superior to a conventional CPU or a computational grid. Recent advances (for example the GTX460 and GTX560 based systems) in inexpensive "desk-top Fermi" processor GPU systems, developed primarily for the high-end computer gaming community, now enable a very impressive price-to-performance ratio to be obtained for a modest monetary outlay. It is clear from our work that considerable refactoring of existing code is needed to obtain the significant performance increases that a GPU can provide and further optimisation, with an attendant increase in replay speed, is likely if we can improve the occupancy figures. However the picture is less clear when considering the overall task of replaying a digital hologram and then searching in 3D for in-focus images of "interesting" objects, extracting these and then classifying them.

Much existing code, including extensive image processing and classification libraries, would need to be ported to a GPU to generate a significant speed increase. The challenge lies not in the porting but in the code refactoring needed to make efficient use of the GPU. In our earlier work [9] the naive approach of recompiling our unmodified *HoloReco* code with the CUDA FFTW libraries resulted in only a factor of six improvement in processing speed and only extensive code refactoring coupled with detailed profiling has enabled the current factor of 240 to be achieved. In addition the task of locating objects in a replayed 3D volume is not a trivially parallel one. Several correlated depth slices need to be considered in order to locate, using some appropriate focusing metric [14], the best one so that each object may be extracted for subsequent classification. We hypothesise that a hybrid approach, with

replay using a GPU and subsequent processing of the datasets using grid or commodity cloud computing may be the most appropriate combination of the two different architectures.

References

[1] Yang Yan, Kang Bo-seon 2011 *Opt.Lasers Eng* **49** 1254–63
[2] Sun H, Benzie P W, Burns N, Hendry D C, Player M A and Watson J 2008 *Phil. Trans. R. Soc. A* **366** 1789–806
[3] Graham G W, Smith W and Nimmo A M 2010 *Limnol.Oceanogr.Meth*. **82** 1–15
[4] Kreis T M, Adams M and Juptner W P O 1997 *Proc. of SPIE* **3098** 224–33
[5] Frigo M and Johnson S G 2005 *Proc. IEEE* **93** 216–31
[6] Nebrensky J J, Hobson P R and Fryer P C 2005 *Proc. of SPIE* **5775** 285–96
[7] Nebrensky J J and Hobson P R 2006 *Proc. of SPIE*. **6252** 62521I.1–6
[8] Nebrensky J J and Hobson P R 2009 Replay of Digitally-Recorded Holograms Using a Computational Grid [available] http://bura.brunel.ac.uk/handle/2438/3443
[9] Nebrensky J J, Hobson P R and Reid I D 2009 Digitally-Recorded Hologram Replay - a Comparison Between a Computational Grid and a GPU Based System, EOS Topical Meeting on Blue Photonics, Aberdeen, UK 18-19 August 2009
[10] Pandey N, Kelly D P, Naughton T J and Hennelly B M, 2009 *Proc. of SPIE* **7442** 744205–1
[11] Zulfiqar A, Hyun-Eui Kim, Dongbiao Han, Jae-Hyeung Park and Nam Kim 2011 *3D Res.* **2** 1–5
[12] Zhuqing Zhu, Min Sun, Heping Ding, Shaotong Feng and Shouping Nie 2009 Fast numerical reconstruction of digital holography based on graphic processing unit [Available] http://dx.doi.org/10.1109/CLEOPR.2009.5292249
[13] [Available] http://holoreco.sourceforge.net/
[14] Meinecke T, Sabitov N and Sinzinger S 2010 *Appl.Opt.* **49** 2446–55