# Using Commodity Graphics Hardware for Real-Time Digital Hologram View-Reconstruction

Lukas Ahrenberg, Andrew J. Page, Bryan M. Hennelly, John B. McDonald and Thomas J. Naughton

***Abstract*—View-reconstruction and display is an important part of many applications in digital holography such as computer vision and microscopy. Thus far, this has been an offline procedure for megapixel sized holograms. This paper introduces an implementation of real-time view-reconstruction using programmable graphics hardware. The theory of Fresnel-based view-reconstruction is introduced, after which an implementation using stream programming is presented. Two different fast Fourier transform (FFT)-based reconstruction methods are implemented, as well as two different FFT strategies. The efficiency of the methods is evaluated and compared to a CPU-based implementation, providing over 100 times speedup for a hologram size of 2048 $\times$ 2048.**

***Index Terms*—Digital Holography, View-Reconstruction, Graphics Hardware (GPU), Fresnel Transform**

## I. Introduction

In 1948, Gabor proposed a new two step imaging method, which he coined holography [1]. The first step is comprised of recording, on a photographic material, the interference pattern between an object wavefield and a known reference wavefield. Gabor recognized that this recorded intensity contained indirect information about the phase of the object wavefield. He proposed a second step to recover this phase information and to 'replay' the object image. This reconstruction step is comprised of illuminating the hologram with the same reference wavefield as was used in recording. In the ensuing years there were many contributions to the science of holography including a milestone paper [2] by Leith and Upatnieks in 1962. They outlined an optical 'off-axis' architecture, based on carrier frequencies in communication theory, to separate the reconstructed object image from the unwanted noise-like twin image. A few years later Goodman and Lawrence recorded a hologram on a vidicon with the lens removed [3]. To reconstruct the hologram they used a digital PDP-6 computer, to simulate optical propagation, in lieu of an optical replay. A similar contribution was made independently in [4]. This electronic recording and numerical replay of holograms is now known as 'digital holography.' Two significant probelms existed with these first steps in digital holography; 1) The electronic recording devices were of low resolution and poor quality and 2) the computers used for numerically reconstructing the holograms were slow and had limited memory. Thus, these early contributions received little attention until a rebirth almost 30 years later when Schnars and Juptner [5] applied modern CCD cameras and CPUs to the principle of digital holography. Since then there have been hundreds of further contributions to the subject. With continuing advances in the field, holographic recording is set to be an attractive alternative to current industrial microscopy and vision systems [6]. One crucial feature of any such system is the ability to render views from optically captured digital holograms in real-time. In contrast to optical display of digital holograms [7], efficient hologram computations are essential for practical implementations of reconstruction-based 3D object segmentation and recognition algorithms [8], [9], [10], [11], [12]. While numerically efficient image reconstruction algorithms exist today, the shear amount of computation required makes it infeasible on a standard CPU.

The speed and resolution of capturing technology has advanced constantly, but reconstruction technology has not kept pace. As the quality of a digital hologram is directly related to its resolution, a single frame can be in the order of several gigabytes, and can be expected to grow to terabytes with future advancements in imaging technology. Currently the only way to provide real-time view-reconstruction is to decrease the resolution of the images. Reconstructing images from optically captured holograms is thus a computationally expensive problem. Standard CPU computation rates have stagnated, with the focus switched to adding multiple low powered cores to a single chip, and thus cannot perform the required computations fast enough to allow for real-time view-reconstruction.

Numerical reconstruction of digital holograms is centred on digitally computing the Fresnel Transform of the electronically recorded 2D $N \times N$ hologram. The Fresnel transform [13] can be derived from the Helmholtz equation, which exactly describes free space light propagation, by assuming a number of approximations. In particular the paraxial approximation is applied which assumes that all the rays of light in the system make small angles with the optical axis of the system. While the discrete counterpart of the integral Fresnel transform is well defined [14], there have been numerous numerical algorithms proposed in the literature [15] for its computation. Each of these algorithms has advantages and disadvantages in terms of 1) accurate approximating the continuous integral transform and 2) the time taken to implement the computation. For a discussion on many of the available algorithms and a framework for their comparison please consult [15]. In the literature two algorithms in particular have received the most attention, the convolution method and the direct method;

L. Ahrenberg, A.J. Page, B.M. Hennelly, and J.B. McDonald are with the Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland e-mail: [ahrenberg, apage, bryanh, johnmcd]@cs.nuim.ie

T.J. Naughton is with the Department of Computer Science, National University of Ireland, Maynooth, Co. Kildare, Ireland, and also with the RFMedia Laboratory, University of Oulu, Oulu Southern Institute, 84100 Ylivieska, Finland e-mail:tomn@cs.nuim.ie .

likely because between them they are sufficient to satisfy the vast majority of digital holographic reconstructions. This paper is concerned with implementing these two algorithms on commodity graphics hardware to enable a considerable speed up over CPU implementations.

There has been very little research into reconstruction of optically captured digital holograms beyond the capabilities of modern CPUs. Page et al. [16] used the spare resources of a loosely–coupled distributed system to reconstruct holograms. However due to the cost of communication this method is only efficient for large holograms that can not be processed on a single memory system. Real-time reconstruction of optically captured holograms for display purposes require fast communication and thus a tightly coupled system. Very recently Shimobaba et al. [17] presented a software similar to ours that also utilizes graphics hardware for holographic view reconstruction. Their work is however targeted specifically towards digital holographic microscopy and only implements one of the two methods described in this paper. In addition we perform intensity computation and brightness adjustment on the GPU which saves a data transfer operation.

Some work has been presented on utilizing specialized and parallel hardware in computer generated holography (CGH). Ito et al. [18] have built several generations of a special purpose hardware platform for point-based CGH. Masuda et al. [19] and Ahrenberg et al. [20] have presented strategies for accelerated CGH generation on graphics hardware. Reicherter et al. [21] and Haist et al. [22] have shown that graphics processing units (GPUs) can be used to efficiently construct optical tweezers. All of these approaches are point based and designed specifically for efficiency in CGH, where the problem consists of a limited number of points located is 3D space. Each point in turn is treated as a light source, and the contribution over the full hologram from these are computed and accumulated individually. This $\mathcal{O}(N \times M)$ approach is not optimal for optical hologram reconstruction, which typically operates on a huge number of samples arranged in a 2D grid. Treating each hologram fragment as a source point thus leads to massive computations. However, as the hologram is contained is a function on a 2D plane, the Fresnel transform may be used for light transport. This function can in turn be computed using an $\mathcal{O}(N \log N)$ algorithm as discussed in Section II.

In this paper we present a method to reconstruct optically captured holograms in real-time using a standard commodity graphics card, which is commonly found in desktop PCs. The highly parallel nature of the GPU architecture makes them attractive for general data processing. While the main area of application still lies in computer graphics, the architectures are now general enough to offer significant performance improvements in many fields, such as numerical computation and simulation. Viewed from a general programming perspective a GPU can be seen as a powerful stream processor [23]. General purpose computation on GPUs (GPGPU) has been facilitated by tools which allow a programmer to access the hardware outside of a graphics API [24], [25]. As part of the contributions in this paper we present a stream processor formulation of the Fresnel transform, and show how to implement it using NVidia's CUDA GPGPU library [25].

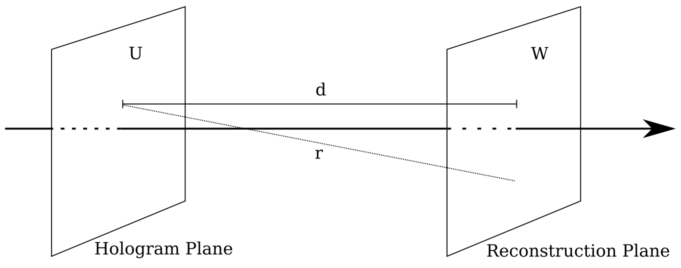


Fig. 1. Hologram and Reconstruction planes at a distance $d$. The distance $r$ is measured between each data point pair of $U$ and $W$.

In Sect. II we define the theory underlying the holographic reconstruction problem and the algorithms used. The implementation is described in Sect. III-A. Experimental results are presented in Sect. IV and we conclude in Sect. V.

## II. Methods for hologram reconstruction

Given a hologram distribution, $U$, we can reconstruct the object image in a plane parallel to the hologram plane and at distance, $d$, by modeling the light propagation. The operation is depicted in Fig. 1. Light propagation between parallel planes can be mathematically describe by the Fresnel-Kirchhoff integral

$$W(u,v) = \frac{i}{\lambda} \iint_{-\infty}^{\infty} U(x,y) \exp\left[\frac{-2\pi i}{\lambda} r(x,y,u,v)\right] dxdy, \tag{1}$$

where $r(x,y,u,v) = \left[(x-u)^2 + (y-v)^2 + d^2\right]^{\frac{1}{2}}$ and $\lambda$ is the wavelength of the light source.

If the reconstruction distance, $d$, is large compared to the hologram size, the Fresnel or paraxial, where $r$ is substituted by the linear and quadratic terms of its Taylor expansion, is valid [13], [26]. This leads to the expression

$$\begin{aligned} W(u,v) = \frac{i}{\lambda} \exp\left(\frac{-2\pi i}{\lambda} d\right) \exp\left[\frac{-i\pi}{\lambda d}\left(u^2+v^2\right)\right] \\ \times \iint_{-\infty}^{\infty} U(x,y) \exp\left[\frac{-i\pi}{\lambda d}\left(x^2+y^2\right)\right] \\ \times \exp\left[\frac{2\pi i}{\lambda d}\left(xu+yv\right)\right] dxdy. \end{aligned} \tag{2}$$

$W(u,v)$ is called the Fresnel Transform of $U(x,y)$

In digital holography the hologram, $U(x,y)$, will be represented as a real or complex valued array of size $N \times M$ elements. Discrete numerical reconstruction methods could be based on Eqns. 1 or 2. However, these would yield $\mathcal{O}(n^2)$ complexity for a full image reconstruction of $n$ samples. Instead, it has been shown, [13], that Eqn. 2 can be rewritten in terms of the continuous Fourier transform. By employing a discrete Fourier transform operator (which can be implemented using the FFT; with complexity $\mathcal{O}(n \log n)$), denoted as $\mathcal{F}$ below, as well as discrete chirp functions, one may easily derive the two algorithms [26], [27], [15].

The convolution approach is described by the following equation

$$W_d(m,n) = \mathcal{F}^{-1}\left[\mathcal{F}(U)\exp\left(\frac{2\pi i d}{\lambda}\right) \times \exp\left\{-\pi i \lambda d\left[\left(\frac{m}{M\delta_m}\right)^2 + \left(\frac{n}{N\delta_n}\right)^2\right]\right\}\right], \quad (3)$$

where $m \in [-\frac{M}{2}, \frac{M}{2})$ and $n \in [-\frac{N}{2}, \frac{N}{2})$ are discrete coordinates and $\mathcal{F}$ denotes the discrete Fourier transform. The convolution method is based on the observation that the original problem can be formulated as a convolution between the hologram function and a phase function. Thus the convolution theorem may be applied to express the operation as a multiplication in frequency space.

The direct method is derived from Eqn. 2 by rewriting it as a Fourier transform of the hologram times a phase factor. It is expressed by

$$W_d(m,n) = \mathcal{F}\left(U\exp\left\{\frac{\pi i}{\lambda d}\left[(m\delta_m)^2 + (n\delta_n)^2\right]\right\}\right) \times \exp\left\{\frac{2\pi i d}{\lambda} - \pi i \lambda d\left[\left(\frac{m}{M\delta_m}\right)^2 + \left(\frac{n}{N\delta_n}\right)^2\right]\right\}. \quad (4)$$

While the direct method requires only one Fourier transform, and thus is less computationally expensive than the convolution approach, it effectively changes the size of the reconstructed image. In other words, the output pixel size is linearly proportional to the distance parameter, $d$. A larger $d$ results in a larger pixel size in the reconstruction, and therefore a larger spatial area of the the reconstruction. This property is sometimes undesirable. The convolution method on the other hand will keep the size of each sample constant and thus is more suited for hologram analysis approaches where object sizes must be comparable. As each method has its advantages we have chosen to implement both in this paper.

A hologram encodes both phase and amplitude and will thus represent the full light-field at the sensor. This allows us to reconstruct different views of the captured scene. By only considering a sub-area of the total hologram, we are effectively creating a camera with a smaller aperture and consequently decreased resolution of the reconstruction. However, not only will this procedure lead to an increased depth of field, it will effectlively reconstruct an image based on light coming from only certain directions of the scene. Thus, the location of the aperture relative to the optical axis will dictate the view imaged through it. Therefore, by choosing size and location in the hologram plane different perspectives can be reconstructed.

The aperture procedure can be written as

$$U_A = T_{k,l}\left(UA^{s,t}_{k,l}\right)\exp\left[\frac{2\pi i}{\lambda d}\left(km\delta_m^2 + ln\delta_m^2\right)\right], \quad (5)$$

where $A^{s,t}_{k,l}$ is a binary valued box aperture function of dimensions $s \times t$, with its center located at the discrete coordinates $(k,l)$ in the hologram plane. It is defined as follows:

$$A^{s,t}_{k,l}(m,n) = \begin{cases} 1 & \text{if } (m-k, n-l) \in [\,\frac{-s}{2}, \frac{s}{2}\,), [\,\frac{-t}{2}, \frac{t}{2}\,) \\ 0 & \text{else.} \end{cases} \quad (6)$$

**Algorithm 1**: Main steps of displaying a hologram as an intensity reconstruction.

```
Data: Hologram: $U$, of resolution $(M,N)$ and sample
      size $(\delta_m, \delta_n)$. Reconstruction distance: $d$.
      Wavelength: $\lambda$. If a sub aperture is used, the size
      $(s,t)$ and location $(k,l)$ is also input.
Result: Intensity image of reconstruction.
1 if Sub-aperture used then
2     $U \leftarrow$ ApertureAt$(U, (s,t), (k,l))$;
3 end
4 $W =$ Propagate$(U, d, \lambda, (\delta_m, \delta_n))$;
5 $I =$ ComputeIntensities$(W)$;
6 Display intensity image $I$;
```

$T_{k,l}$ is an operator that translates the origin of the box aperture by $(-k,-l)$ to the optical axis of the hologram. This yields a common image center in the reconstructions. This operation will however introduce a phase shift in the holographic data, which in turn will act as a translation of the reconstructed object. This is counteracted by the exponential in Eqn. 5.

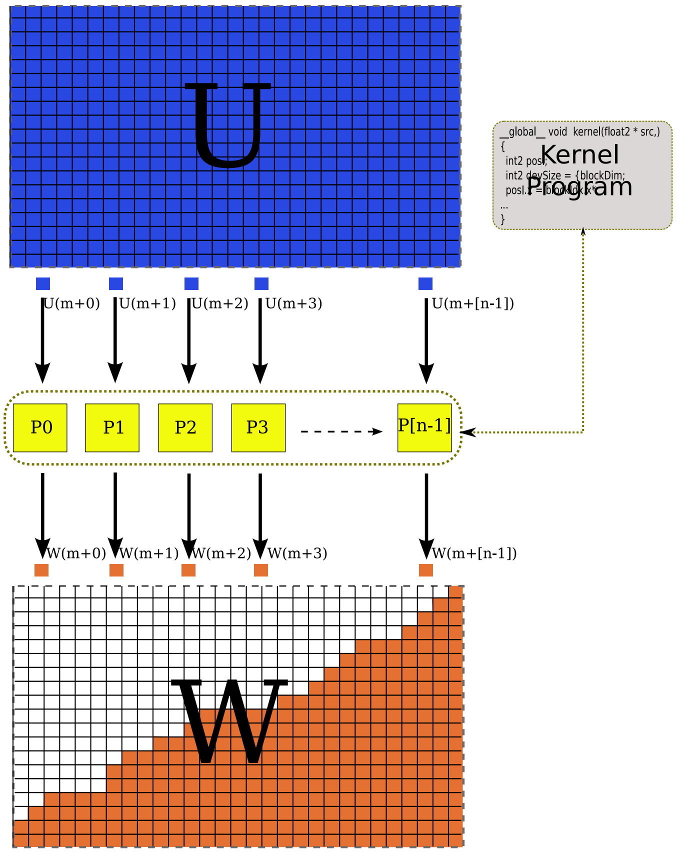
## III. Implementation

Algorithm 1 describes a three step method for hologram view-reconstruction. First, a sub-aperture is selected, corresponding to the desired view. Second, the wave field is propagated the reconstruction distance $d$. Finally, the intensities are computed and the resulting image is displayed on screen.

Of these operations, propagation is the most computationally expensive, and the main factor determining performance. Basing an implementation on the convolution (Eqn. 3) and direct (Eqn. 4) methods allows for the operations to be broken down into a set of Fourier transforms and multiplications. For instanc; the convolution method is described as 1) Fourier transform, 2) Multiplication of each element in the spectra by a phase function, and 3) an inverse Fourier transform. The algorithm has a complexity of $\mathcal{O}(n \log n)$, if implemented using the FFT. In an implementation on a CPU these operations still perform slowly due to the massive data sizes.

As noted in [16], this operation can be efficiently executed on a parallel system. In recent years, the stream processor model [23] has grown more prevalent with the increased utility of GPUs. Stream processing is based on the observation that, for certain algorithms, the same numerical operations need to be executed on a very large set of data repeatedly, in a single instruction multiple data (SIMD) fashion. This is typically the case in image processing and computer graphics, which is one of the reasons why highly parallellized dedicated hardware in these areas benefit from the model.

The basic principle of stream processing is to use multiple processors to execute the same *kernel* of code on a large set of data in parallel. Each processor unit executes the same code, but works independently. Thus large datasets can be processed as streams much faster than on a serial processor, where it would typically be implemented in a loop structure. Stream processing has the most advantage over single thread pro-



```
__global__ void  kernel(float2 * src,)
{
  int2 posI;
  int2 devSize = {blockDim;
  posI.x = blockIdx.x*
...
}
```

Fig. 2. Principles of stream programming. The same kernel code is loaded to all processors $P_0..P_{[n-1]}$. Each Processor works independently on a stream of fragments from the source $U$. The processed results are stored on appropriate indices in the destination $W$.

cessing if a computationally heavy kernel is to be performed independently on a large dataset.

The kernel is loaded to a set of virtual threads distributed between the different processors and the data stream split up between them, as illustrated in Fig. 2. This is an efficient model for problems requiring many arithmetical operations, or where the operations can be performed independently on the elements of the data.

Modern GPUs are stream processors, often with tens or hundreds of processors capable of advanced arithmetic and logic operations. Up until recently graphics processors were only programmable using graphics APIs. However, tools such as Brook [24] and CUDA [25] now exist that allow for a more general programming model. The GPU is thus a cost effective and efficient choice for high speed numerical computing.

## A. Implementation on graphics hardware

The main limitation of a pure GPU based approach to holographic reconstruction has previously been available memory. However, the challenge of implementing a general FFT method that can perform as well as the highly efficient libraries for CPUs available has also been a factor [28]. Today graphics boards come equipped with enough memory to allow reconstruction of digital holograms of sizes matching the current capture technology. In addition, efficient FFT methods specially designed for GPU execution has been presented and implemented [29], [30], [31].

Out of performance concerns we will only consider holograms where the sides sizes are a power of two. Other sizes are zero-padded to the closest power of two. While this costs some amount of additional memory it may allow the FFT algorithm, as well as the kernel planning described below, to optimize its output. It can be argued that in some special cases the hologram size may differ significantly from a power of two, and thus both cause a memory and processing penalty to the algorithm. However we have observed that in reality modern digital imaging equipment tend to have resolutions that lie close to a power of two and thus the padding will have little practical influence.

Returning to the program outlined in Alg. 1, the three functions `ApertureAt`, `Propagate` and `ComputeIntensities` can all be implemented as stream programs. We will start by discussing propagation using the direct and convolution methods. Algorithm 2 and 3 describes the steps needed to perform the convolution approach and direct approach respectively. In both cases, the functions `GPUFFT` and `GPUIFFT` denotes GPU-based FFT and inverse FFT calls.

While multi-dimensional FFTs are supported natively in most FFT libraries, the implementation is often more memory consuming than a set of one dimensional transforms. This is due to the fact that in order to optimize for speed, the data may be rearranged in a more cache friendly manner. For a parallel implementation, such as the one used in this paper, this may prove even more important. While we will opt for speed in favor of memory, for real-time reconstruction, there are situations when processing very large holograms is desired. In these cases the source data may take up most of the available GPU memory, and a 2D FFT is infeasible. It is thus also of interest to consider an alternative low memory approach for parallel implementations, such as the one described in [16].

This technique basically uses the classic row-column approach, splitting the full dimensional transform into a series of 1D FFT calls. However, as the algorithm is executed on a parallel architecture, the different data vectors are distributed between multiple processors. Thus, each processor only needs to allocate memory and optimize for a single 1D FFT. We have implemented support for both direct 2D FFTs, and the above outlined sequential transform in our reconstruction methods. A comparison can be found in Section IV.

In both Algs. 2 and 3, $M \times N$ threads are created to execute the non-FFT kernels, performing phase-shift multiplications of the data. There is one active thread per data element. As there are typically fewer processors than data elements each processor will have a queue of kernels waiting to be executed. An alternative approach would be to process $\frac{D}{P}$ elements per kernel, where $D$ is the total number of samples and $P$ the total number of processors. All data is thus processed in one go. However, such an approach requires several sequential operations to be performed in the kernel code by a loop. While our approach forces some sequential execution, due to an overhead of data elements, this does not occur within the kernel code. This allows the GPU scheduler to plan ahead

**Algorithm 2**: Pseudocode showing the GPU calls for the convolution method.

```
Data: Hologram: $U$, of resolution $(M, N)$ and sample
      size $(\delta_m, \delta_n)$. Reconstruction distance: $d$.
      Wavelength: $\lambda$.
Result: Complex valued wave field $W$ resulting from
        propagating $U$ a distance $d$.
1 $W \leftarrow$ GPUFFT$(U)$ ;
2 Create $M \times N$ threads ;
3 Set kernel described in Alg. 4 as active. ;
4 $W \leftarrow$ kernel$(W)$ ;
5 $W \leftarrow$ GPUIFFT$(W)$ ;
```

**Algorithm 3**: Pseudocode showing the GPU calls for the direct method.

```
Data: Hologram: $U$, of resolution $(M, N)$ and sample
      size $(\delta_m, \delta_n)$. Reconstruction distance: $d$.
      Wavelength: $\lambda$.
Result: Complex valued wave field $W$ resulting from
        propagating $U$ a distance $d$.
1 Create $M \times N$ threads ;
2 Set kernel described in Alg. 5 as active. ;
3 $W \leftarrow$ kernel$(U)$;
4 $W \leftarrow$ GPUFFT$(W)$;
5 Set kernel described in Alg. 6 as active. ;
6 $W \leftarrow$ kernel$(W)$;
```

due to the independence of the kernel operations. The strategy also keeps the number of memory operations per kernel to a minimum, as such operations typically are far more costly than arithmetic operations on a GPU architecture.

Pseudocode for the convolution approach kernel is presented in Alg. 4, and the two kernels needed for the direct method are presented in Algs. 5 and 6. The latter method requires two different kernels as multiplication by a phase factor is required in both frequency and spatial domains. The spatial kernel is loaded and executed first, after which the FFT is performed, and finally the frequency space kernel is loaded and executed.

In all cases the input is a pointer to the hologram, or its Fourier spectrum, resolution, sample size, distance and wavelength, as well as a virtual thread index. The index is a unique ID provided by the calling API for each kernel execution. Thus, we do not need to know the specification of the underlying hardware. The function `Index2Pos` simply maps between the thread index and a sample position in the data array. As can be seen, the kernels are only concerned with one specific element, performing one read and one write to global memory.

### *B. View computation*

As outlined in Section II, different 3D views of the scene captured can be reconstructed by considering a sub-window, acting as an aperture, instead of the whole hologram. The core of the procedure is described by Eqn. 5. Every sample in the

**Algorithm 4**: Kernel performing multiplication of the phase factors used in the convolution method. The function Index2Pos returns a unique data element position given a parallel thread index.

```
Data: Fourier spectra of hologram: $W_f$. Hologram
      resolution $(M, N)$ and sample size $(\delta_m, \delta_n)$.
      Virtual thread index $I$. Reconstruction distance: $d$.
      Wavelength: $\lambda$.
Result: One sample of the reconstruction Fourier spectra:
        $W_f(m, n)$
1 $(m, n) \leftarrow$ Index2Pos$(I)$ ;
2 $k \leftarrow \exp\left(\frac{2\pi i d}{\lambda}\right)$ ;
3 $W_f(m, n) \leftarrow k \times W_f(m, n)$
    $\times \exp\left\{-\pi i \lambda d \left[\left(\frac{m}{M\delta_m}\right)^2 + \left(\frac{n}{N\delta_n}\right)^2\right]\right\}$ ;
```

**Algorithm 5**: Kernel performing multiplication of the inner phase factor used in the direct method. The function Index2Pos returns a unique sample position given a parallel thread index.

```
Data: Hologram $U$. Hologram resolution $(M, N)$ and
      sample size $(\delta_m, \delta_n)$. Virtual thread index $I$.
      Reconstruction distance: $d$. Wavelength: $\lambda$.
Result: One sample of the hologram – phase factor
        product: $U(m, n)$
1 $(m, n) \leftarrow$ Index2Pos$(I)$ ;
2 $U(m, n) \leftarrow U(m, n)$
    $\times \exp\left\{\frac{\pi i}{\lambda d}\left[(m\delta_m)^2 + (n\delta_n)^2\right]\right\}$ ;
```

**Algorithm 6**: Kernel performing multiplication of the second, Fourier space, phase factor of the direct method. The function Index2Pos returns a unique sample position given a parallel thread index.

```
Data: Fourier spectra of hologram: $W_f$. Hologram
      resolution $(M, N)$ and sample size $(\delta_m, \delta_n)$.
      Virtual thread index $I$. Reconstruction distance: $d$.
      Wavelength: $\lambda$.
Result: One sample of the reconstruction Fourier spectra:
        $W_f(m, n)$
1 $(m, n) \leftarrow$ Index2Pos$(I)$ ;
2 $W_f(m, n) \leftarrow W_f(m, n)$
    $\times \exp\left\{\frac{2\pi i d}{\lambda} - \pi i \lambda d \left[\left(\frac{m}{M\delta_m}\right)^2 + \left(\frac{n}{N\delta_n}\right)^2\right]\right\}$ ;
```

window is phase shifted by multiplication with an exponential function, and translated to a corresponding window at the optical axis.

The kernel implementation is straight forward, and the process is performed in two steps: First the output array is set to zero. Then the sub-window is copied into the central region by a kernel that also performs the multiplication with the exponential. This will create an windowed version of the hologram, padded out to the original size. These operations are denoted by the function `ApertureAt` in in Alg. 1, and can be performed as a preprocessing stage to the propagation.

### C. Intensity computation

After propagation of a given distance, by either the convolution or the direct method, the result is an array of complex valued data representing the distribution in the reconstruction plane. In order to display this on screen we need to compute brightness values and to map these values to the dynamic range of the screen.

Intensity values can be directly computed from the square magnitude as

$$I(m,n) = W(m,n)W(m,n)^*, \tag{7}$$

where $*$ denotes the complex conjugate. However, this may lead to a poor contrast quality, and thus loss of detail, if used directly. Instead, we consider a gamma corrected value

$$I(m,n) = \left[W(m,n)W(m,n)^*\right]^\gamma, \tag{8}$$

where $\gamma$ has the effect of a nonlinear shift of the contrast distribution. For instance, a value of $\gamma = 0.5$ will give us the amplitude image.

We also have to consider that the resulting brightness range may be greater than what is representable on screen. Assuming that the display buffer intensities are normalized to the range $[0,1]$, we map the reconstructed image $I$, to screen, $S$, by

$$S(m,n) = \frac{I(m,n) - T_{\text{min}}}{T_{\text{max}} - T_{\text{min}}}. \tag{9}$$

In the above equation $T_{\text{min}}$ and $T_{\text{max}}$ are the upper and lower brightness thresholds, mapped to 0 and 1 respectively. Changing the thresholds will effectively change the visible dynamic range of the output.

Equations 8 and 9 have been implemented in a kernel performing the operations on a complex valued array and rendering an intensity image that can be displayed on a computer monitor. This is denoted by the stage `ComputeIntensities` in Alg. 1. The $\gamma$, $T_{\text{min}}$ and $T_{\text{max}}$ parameters can be changed interactively by the user in order to allow for better on–screen calibration.

### D. Implementation details

For the GPU dependent components of our implementation we have chosen to use CUDA [25], as the API is native to the NVidia GeForce 8 hardware platform that we are currently using. We also employ the CUFFT [31] library to perform the Fourier transforms. The hardware supports vector datatypes up to `float4` natively. The complex numbers were implemented using the `float2` data type, requiring a total of 8 bytes per sample.

After one pass of the main display loop, outlined in Alg. 1, we will have a floating point intensity image in graphics board memory. We then use CUDA's built-in OpenGL interoperability to copy the data to a frame buffer for direct display. As the whole operation is performed in device memory, there is no latency due to bus transfer speeds.

Finally, we have implemented a graphical user interface allowing the user to choose reconstruction method and parameters, such as distance, aperture and view. Thus, for the first time enabling real-time interactive numerical view-reconstruction of digital holograms.

## IV. Results and discussion

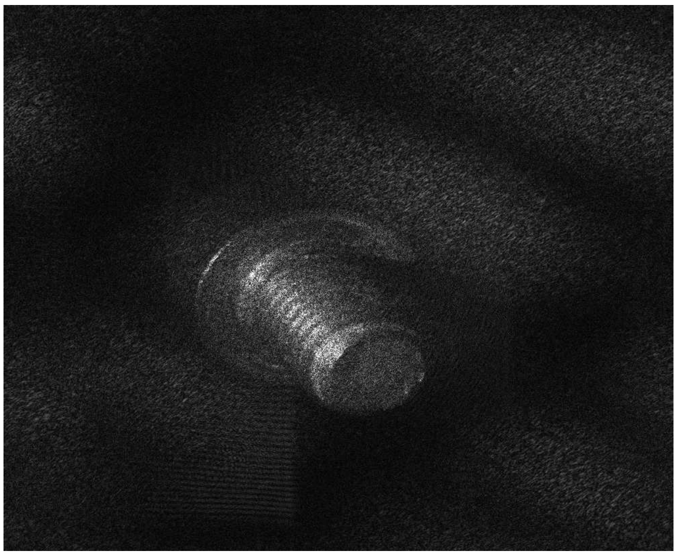
We have tested our implementation on a Linux PC with 2 Gigabytes system RAM, AMD Athlon Dual Core 64 bit processor and a GeForce 8800GTX graphics card with 768 Mbytes on board RAM and 128 stream processors. Fig. 3 shows two images reconstructed by our software. Table I show a comparison table of average reconstruction times for both the direct and convolution approaches and hologram sizes up to $4096 \times 4096$. The CPU-based methods were implemented in C++ using the fast FFTW library [32] (v. 3.1.2, single threaded) on a AMD Athlon 64 X2, 2.3 GHz equipped with 2 Gbytes RAM.

As can be seen from the table, our GPU implementation of the convolution approach provides a relative speedup of roughly 100 times compared to the CPU implementation. Thus, our implementation is capable of performing real-time view-reconstruction and rendering from modern digital holographic setups. The GPU version of the direct approach is still faster, but this is to be expected, as only one FFT needs to be performed by this algorithm. Note however that due to two additional multiplicative steps, the direct method on the CPU actually performs slower than the convolution approach in our reference implementation. This leads to a greater relative speedup for the GPU in this case, however with proper CPU level optimization it can be expected to lie around the same ratio as for the convolution approach. We have therefore chosen to compute the relative speedup based on the convolution approach.

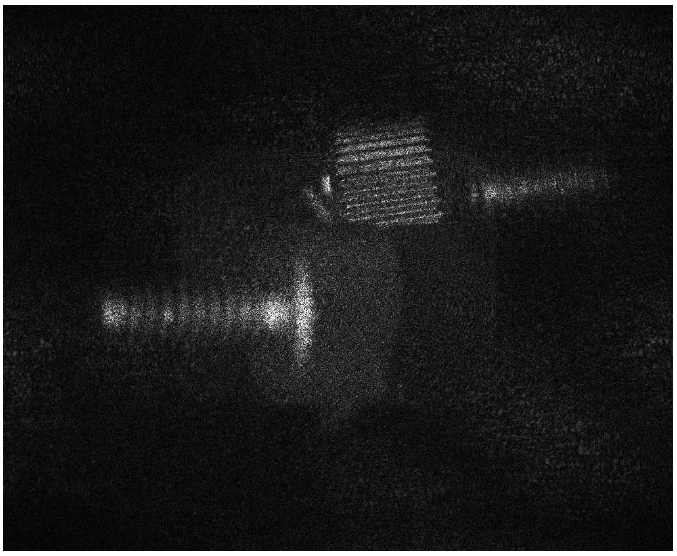
| | Reconstruction Time (ms) | | | | | |
|---|---|---|---|---|---|---|
| | convolution method | | | direct method | | |
| Resolution | G(2) | G(1) | C(2) | G(2) | G(1) | C(2) |
| $512 \times 512$ | 2.3 | 3.3 | 251.4 | 1.6 | 2.1 | 306.3 |
| $1024 \times 1024$ | 7.9 | 19.8 | 1060.6 | 5.5 | 11.5 | 1305.0 |
| $2048 \times 2048$ | 47.0 | 106.8 | 4550.8 | 29.4 | 59.3 | 5379.7 |
| $4096 \times 4096$ | 204.6 | 598.6 | 23530.4 | 126.2 | 322.3 | 23648.1 |

TABLE I
View-reconstruction times in milliseconds for the convolution method on GPU (G) and CPU (C), and the direct method on GPU. The times were averaged over 1000 runs. The number in parenthesis denotes the dimensionality of the FFT.

Table I also show results from using a set of 1D FFTs instead of the full 2D FFT. While this approach is slower than when using the native 2D FFT, it has the advantage of consuming much less memory.



(a)



(b)

Fig. 3. Two reconstructions by our software. (a) The hologram ”LargeScrew” at a distance of 336 mm. (b) The hologram ”TwoScrews” at a distance of 367 mm. Hologram size: $2048 \times 2048$ samples. The direct method was used for both reconstructions.

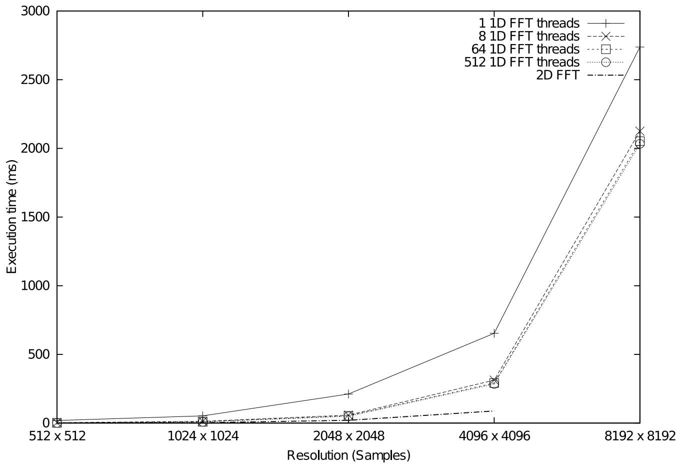


Fig. 4. Log–linear plot of the execution time of FFT calls at different resolutions. The 2D FFT has the over all best performance but could be used on resolutions up to $4096 \times 4096$ due memory constraints. The 1D FFTs were performed in sequence to achieve a full 2D operation. The number of threads denote how many rows of the 2D matrix were executed independently and thus in theory in parallel. The experiments were performed using CUDA on a NVidia GeForce 8800GTX graphics card, with 768 Mbytes RAM.

As described in Section III-A the efficiency of our reconstruction methods is largely dependent on the FFT implementation. We currently use the CUFFT library, and while the 2D FFT is very fast, it is also quite memory demanding. We have observed that, as a rule of thumb, the FFT planner allocates three times the hologram size of additional memory. Thus, as described above, we have implemented an alternative, sequential approach based on a series of 1D FFTs. While this approach is much less memory intensive than the direct 2D FFT, it does not yield the same processing speeds.

In order to evaluate the efficiency of both approaches we measured for a range of resolutions. Fig. 4 shows a graph of the execution time for a single 2D FFT for different resolutions and settings. The number of threads denotes how many parallel 1D FFTs that is using the CUFFT library. In practice one 1D FFT per row or column of the 2D matrix could be performed independently. We also plot the result of a native 2D FFT. Not surprisingly, this method outperforms the 1D approach, however, on our GeForce 8800GTX with 768 megabytes of memory, we could only perform FFTs of sizes up to $4096 \times 4096$ before running out of memory. Thus, the approach is useful for processing large holograms, requiring more memory than currently available on graphics hardware. The method improves with the number of threads used for individual FFTs, but the plot suggests that this will only be up to a certain level, dependent on the hardware used, and never as good as a pure 2D transform.

Employing more threads will also cost more memory, as an individual FFT must be planned for each. Fig. 5 show threads and FFT memory consumption plotted against execution time for a $2048 \times 2048$ FFT. The vertical and horizontal line show the constant memory consumption for a 2D FFT. As can be seen better performance costs some amount of dedicated memory. It may also be argued, that if using this approach, there is probably an optimal memory to efficiency relationship.

## V. Conclusions

We have shown that it is possible to perform real-time reconstruction of digital megapixel holograms using stream processing on graphics hardware. The GPU-implementation clearly outperforms a single CPU; with execution times in the millisecond range, and a speedup of over 100 times for a digital hologram resolution of today’s standard ($2048 \times 2048$). For display purposes, rendering directly to the GPU memory is very beneficial, and avoids unnecessary copying of the frame buffer data over the system bus.

The main limiting factor for our reconstruction method is the memory requirements of the CUFFT library. The current version uses an additional memory allocation of three times the input array size for 2D complex FFTs. A single hologram

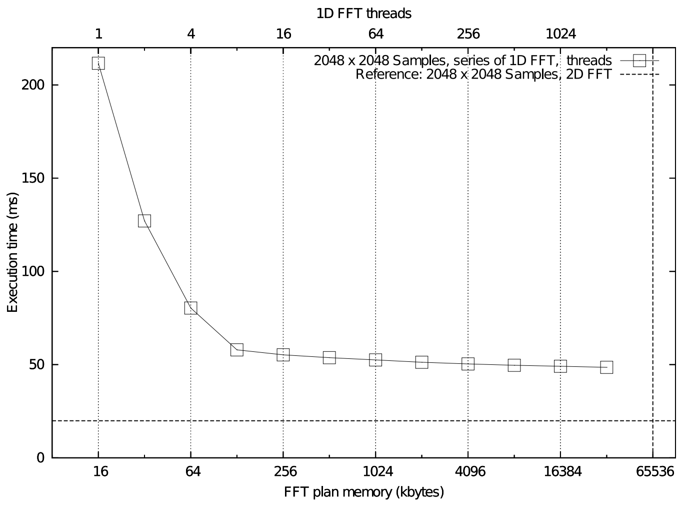


Fig. 5. Log–linear plot of additional memory usage vs. execution time for a $2048 \times 2048$ 2D FFT using sequential 1D FFT calls. The lower horizontal axis shows memory consumption in kilobytes, while the upper denotes the number of potential parallel 1D FFTs executed. The thick horizontal and vertical line denote reference values in the form of memory usage and execution time of a 2D FFT. The experiments were performed using CUDA on a NVidia GeForce 8800GTX graphics card, with 768 Mbytes RAM.

of size $4096 \times 4096$ require 128 Megabytes which amounts to one sixth of the total GPU memory we have at our disposal on our current card. Its 2D FFT plan would thus require 384 Megabytes using CUFFT.

We also investigated the alternative of using several 1D FFTs as an alternative to the faster 2D approach. From our tests we conclude that although the native 2D FFT is the method of choice for our main real-time implementation, the row-column method of 1D FFTs is a much less memory demanding alternative. Thus, it is useful for rapid processing of large holograms at interactive, but not real-time, rates.

Real-time hologram inspection has many attractive applications in areas such as digital holographic microscopy and computational holography. As shown in [17] on-line viewing from a holographic recording setup by reconstructing views from a digital video signal in real-time is possible with today's technology. Our software would be quite suitable for such an application, and while we expect the performace to be limited by the CPU to GPU memory transfer speed, and thus ultimately by the PCI-express bus, it can be expected to perform in real-time to interactive rates as only a single transfer per frame is necessary.

Finally, we note that there are numerous 'ping-pong' Fresnel based algorithms appearing in the literature mainly with applications to phase retrieval or twin image reduction [33], [34], [35], [36]. Often these algorithms are not employed because of their time intensive nature. Due to the fast speed of our Fresnel transform calculations we believe that the ideas presented in this paper will have widespread application in these areas.

## Acknowledgements

We thank Conor Mc Elhinney for capturing the digital holograms. We would like to thank our anonymous reviewers for helpful comments. Support is acknowledged from Science Foundation Ireland under the National Development Plan, and the European Commission through a Marie Curie Intra-European Fellowship.

## References

[1] D. Gabor, "A new microscopic principle," *Nature*, vol. 161, pp. 777–778, 1948.
[2] E. N. Leith and J. Upatnieks, "Reconstructed waefronts and communications theory," *JOSA*, vol. 52, pp. 1123–1130, 1962.
[3] J. W. Goodman and R. W. Lawrence, "Digital image formation from electronically detected holograms," *Applied Physics Letters*, vol. 11, no. 3, pp. 77–79, 1967.
[4] M. A. Kronrod, N. S. Merzlyakov, and L. P. Yaroslavskii, "Reconstruction of a Hologram with a Computer," *Soviet Physics Technical Physics*, vol. 17, pp. 333–+, Aug. 1972.
[5] U. Schnars and W. P. Juptner, "Direct recording of holograms by a CCD target and numerical reconstruction," *Appl. Opt.*, vol. 33, pp. 179–181, Jan. 1994.
[6] Y. Frauel, T. J. Naughton, O. Matoba, E. e Tajahuerce, , and B. Javidi, "Three-dimensional imaging and processing using computational holographic imaging," *Proceedings of the IEEE*, vol. 94, no. 3, pp. 636–653, March 2006.
[7] U. Gopinathan, D. Monaghan, B. Hennelly, C. McElhinney, D. Kelly, J. McDonald, T. Naughton, and J. Sheridan, "A projection system for real world three-dimensional objects using spatial light modulators," *Display Technology, Journal of*, vol. 4, no. 2, pp. 254–261, June 2008.
[8] Y. Frauel, E. Tajahuerce, M.-A. Castro, and B. Javidi, "Distortion-tolerant three-dimensional object recognition with digital holography," *Appl. Opt.*, vol. 40, no. 23, pp. 3887–3893, 2001.
[9] T. J. Naughton, Y. Frauel, B. Javidi, and E. Tajahuerce, "Compression of digital holograms for three-dimensional object reconstruction and recognition," *Appl. Opt.*, vol. 41, no. 20, pp. 4124–4132, 2002.
[10] B. Javidi, S. Yeom, I. Moon, and M. Daneshpanah, "Real-time automated 3d sensing, detection, and recognition of dynamic biological micro-organic events," *Opt. Express*, vol. 14, no. 9, pp. 3806–3829, 2006.
[11] M. DaneshPanah and B. Javidi, "Tracking biological microorganisms in sequence of 3d holographic microscopy images," *Opt. Express*, vol. 15, no. 17, pp. 10 761–10 766, 2007.
[12] C. P. McElhinney, J. B. McDonald, A. Castro, Y. Frauel, B. Javidi, and T. J. Naughton, "Depth-independent segmentation of macroscopic three-dimensional objects encoded in single perspectives of digital holograms," *Opt. Lett.*, vol. 32, no. 10, pp. 1229–1231, 2007.
[13] J. W. Goodman, *Introduction to Fourier Optics*, 3rd ed. Roberts & Company, 2005.
[14] B. Hennelly and J. Sheridan, "Fast numerical algorithm for the linear canonical transform," *JOSA A*, vol. 22, pp. 928–937, 2005.
[15] B. M. Hennelly and J. T. Sheridan, "Generalizing, optimizing, and inventing numerical algorithms for the fractional fourier, fresnel, and linear canonical transforms," *JOSA A*, vol. 22, no. 5, pp. 917–927, May 2005.
[16] A. J. Page, L. Ahrenberg, and T. J. Naughton, "Low memory distributed reconstructionof large digital holograms," *Opt. Express*, vol. 16, no. 3, pp. 1990–1995, 2008.
[17] T. Shimobaba, Y. Sato, J. Miura, M. Takenouchi, and T. Ito, "Real-time digital holographic microscopy using the graphic processing unit," *Opt. Express*, vol. 16, no. 16, pp. 11 776–11 781, 2008.
[18] T. Ito, N. Masuda, K. Yoshimura, A. Shiraki, T. Shimobaba, and T. Sugie, "Special-purpose computer horn-5 for a real-time electroholography," *Opt. Express*, vol. 13, pp. 1923–1932, 2005.
[19] N. Masuda, T. Ito, T. Tanaka, A. Shiraki, and T. Sugie, "Computer generated holography using a graphics processing unit," *Opt. Express*, vol. 14, pp. 603–608, 2006.
[20] L. Ahrenberg, P. Benzie, M. Magnor, and J. Watson, "Computer generated holography using parallel commodity graphics hardware," *Opt. Express*, vol. 14, no. 17, pp. 7636–7641, August 2006.
[21] M. Reicherter, S. Zwick, T. Haist, C. Kohler, H. Tiziani, and W. Osten, "Fast digital hologram generation and adaptive force measurement in liquid-crystal-display-based holographic tweezers," *Appl. Opt.*, vol. 45, no. 5, pp. 888–896, 2006.
[22] T. Haist, M. Reicherter, M. Wu, and L. Seifert, "Using graphics boards to compute holograms," *Computing in Science and Eng*, vol. 8, no. 1, pp. 8–13, 2006.

[23] U. J. Kapasi, S. Rixner, W. J. Dally, B. Khailany, J. H. Ahn, P. Mattson, and J. D. Owens, “Programmable stream processors,” *IEEE Computer*, pp. 54–62, August 2003.
[24] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, “Brook for gpus: stream computing on graphics hardware,” in *SIGGRAPH ’04: ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM, 2004, pp. 777–786.
[25] *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, 1st ed., Nvidia, June 2007.
[26] U. Schnars and W. Jueptner, *Digital Holography*. Springer, 2005.
[27] T. Kreis, *Handbook of Holographic Interfereometry*. Wiley-VCH, 2005.
[28] J. D. Owens, S. Sengupta, and D. Horn, “Assessment of graphic processing units (GPUs) for department of defense (DoD) digital signal processing (DSP) applications,” Department of Electrical and Computer Engineering, University of California, Davis, Tech. Rep. ECE-CE-2005-3, Oct. 2005.
[29] T. Jansen, B. von Rymon-Lipinski, N. Hanssen, and E. Keeve, “Fourier volume rendering on the gpu using a split-stream-FFT,” in *VMV*, 2004, pp. 395–403.
[30] I. Buck, T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan, “Brook for GPUs: stream computing on graphics hardware,” in *SIGGRAPH ’04: ACM SIGGRAPH 2004 Papers*. New York, NY, USA: ACM, 2004, pp. 777–786.
[31] *CUDA CUFFT Library*, 1st ed., Nvidia, June 2007.
[32] M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” *Proc. of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.
[33] R. W. Gerchberg and W. O. Saxton, “A practical algorithm for the determination of phase from image and diffraction plane pictures,” *Optik*, vol. 35, p. 227246, 1972.
[34] J. R. Fienup, “Phase retrieval algorithms: a comparison,” *Appl. Opt.*, vol. 21, no. 15, p. 2758, 1982.
[35] G. Liu and P. D. Scott, “Phase retrieval and twin-image elimination for in-line fresnel holograms,” *J. Opt. Soc. Am. A*, vol. 4, no. 1, p. 159, 1987.
[36] T. Latychevskaia and H.-W. Fink, “Solution to the twin image problem in holography,” *Phys. Rev. Lett.*, vol. 98, no. 23, 2007.

**Lukas Ahrenberg** received the M.Sc. degree in scientific computing from Uppsala University, Sweden in 2001, and the Ph.D. degree in computer science from Max-Planck-Institut für Informatik / Universität des Saarlandes, Germany in 2007. He has worked as a software engineer and consultant for several companies; as a Junior Lecturer at Uppsala University and Gävle University, Sweden, and has been a visiting researcher at University of Aberdeen, U.K., at Technische Universität Braunschweig, Germany, and at Chalmers University of Technology, Sweden. He is currently a Postdoctoral Researcher at the National University of Ireland, Maynooth where he is working on digital computational holography and computer generated holograms.

**Andrew J. Page** received a B.Sc. degree in Computer Science and Software Engineering from the National University of Ireland, Maynooth in 2003. He is currently working toward his Ph.D. degree in the same university. His research interests include scheduling, distributed computing and digital holography. He is now a postdoctoral researcher at the National College of Ireland, Dublin.

**Bryan M. Hennelly** received his BE degree in Electronic Engineering from University College Dublin in 2001. He completed his PhD in optical engineering three years later and worked as a temporary lecturer in the Department of Electrical and Electronic Engineering from 2004-2005 teaching optics, digital electronics and electronic engineering to final year students. His PhD consisted of three separate theses on optical encryption, the discrete modelling of optical systems and metrology. After a year of lecturing he took a position as a post-doctoral researcher in the optics group in the department of Computer Science, NUI Maynooth. His main focus of research is now digital holographic microscopy and also discrete signal processing. He is also an active researcher in the area of phase-space optics and has recently been invited by McGraw-Hill to co-edit a book on the subject. He is currently a fellow of the Irish Research Council for Science, Engineering and Technology under the Embark Initiative Scheme. He is to be the principal investigator for NUIM in a recently awarded EC FP7 grant in digital holography and has published over 60 journal and conference proceeding papers.

**John McDonald** received the B.Sc. degree (double honours) in computer science and mathematical physics from the National University of Ireland (NUI), Maynooth, Ireland. He is a Lecturer in the Department of Computer Science at NUI Maynooth, with a permanent appointment since 2001. In 2002 he co-founded the Computer Vision and Imaging Laboratory at that department, which focuses on imaging science, computer vision and digital holography. His research interests include computer vision and pattern recognition, facial image processing and analysis, multiple view vision systems, and intelligent vehicle systems. He was Chair of the International Machine Vision and Image Processing Conference 2007. He has published over 45 papers in peer-reviewed journals and conferences proceedings and holds two patents. He is a member of the IAPR and a member of the IEEE.

**Thomas J. Naughton** received the B.Sc. degree (double honours) in computer science and experimental physics from the National University of Ireland, Maynooth, Ireland. He has worked at Space Technology (Ireland) Ltd. and has been a visiting researcher at the Department of Radioelectronics, Czech Technical University, Prague, and the Department of Electrical and Computer Engineering, University of Connecticut, Storrs. He is a Senior Lecturer in the Department of Computer Science, National University of Ireland, Maynooth, with a permanent appointment since 2001. Since 2007 he has been a European Commission Marie Curie Fellow at Oulu Southern Institute, University of Oulu, Finland. He leads the EC FP7 three-year eight-partner collaborative project Real 3D. His research interests include optical information processing, computer theory, and distributed computing. He has served as a committee member on 12 international IEEE, ICO, and SPIE conferences. He has co-authored more than 150 publications including 35 journal articles and 20 invited conference papers. He is co-recipient of the 2008 IEEE Donald G. Fink Prize Paper Award.