Experimental Techniques
https://doi.org/10.1007/s40799-021-00520-x

APPLICATIONS PAPER

A Range and Performance Optimized Version of the Computer-Aided Speckle Interferometry Algorithm for Real-Time Displacement-Strain Field Monitoring

L. Keene[1]

Received: 2 August 2021 / Accepted: 29 September 2021

Abstract
This work presents an optimized implementation of the Computer-Aided Speckle Interferometry algorithm which enables full-field determination of displacements and strains on commodity Graphics Processing Units at high resolution and frame rates. By combining careful control of the average speckle size in a laser speckle pattern with a simple sampling rate conversion scheme, a compact representation of the optical speckle is achieved. This allows for optimal use of Graphics Processing Unit architecture with robust range extension. The optimal mapping of the Computer-Aided Speckle Interferometry algorithm to Graphics Processing Unit architecture is shown in detail, and a straightforward method for disambiguating large displacements is illustrated. Lastly, this paper demonstrates a two-step subimage-tapering modification to the original algorithm that enables robust range enhancement while maintaining resolution. Results from numerical simulations on synthetic speckle patterns are shown, and runtime performance metrics are provided, with performance ranging up to 60 frames per second in some cases. The method is suitable for interactive experimental mechanics research, process and testing or any application where real-time high-resolution displacement-strain monitoring is needed. A .NET Framework class library enabling the incorporation of the algorithm into 3rd -party applications is available for download.

Keywords Computer-aided Speckle Interferometry · General-purpose Graphics Processing Unit · GPGPU · Speckle Interferometry · Digital image correlation · Experimental mechanics

Introduction

The application of optical methods for resolving full-field displacements and strains is a mature field. Two primary approaches – one statistical and one interferometric – are commonly chosen for this purpose. The Digital Image Correlation (DIC) approach involves direct spatial-statistical analysis of digitized image pairs via application of a cross-correlation calculation [1]. This method attempts to generate a deterministic measure of the degree of similarity between two speckle patterns and is generally very robust. The second (interferometric) approach pre-dates the correlative one and consists of holographic and laser speckle interferometry methods that leverage optical interference phenomena [2]. The difference between the two – while interesting from a technical and academic standpoint - is subtle, and ultimately both methods generate equivalent outputs consisting of correlation fringes (i.e., displacement isolines). Both types have found widespread use in fluid and solid mechanics research. Due to its straightforward methodology, robustness, and the increasing power of digital computers, direct cross-correlation analysis eventually became the more commonly adopted method. The interferometric approach has continued to evolve, however, and reached its zenith with the Computer-Aided Speckle Interferometry (CASI) algorithm [3]. This hybrid approach combines the sensitivity characteristics of speckle interferometry with the automation capabilities of digitized imagery by computing the interference within computer memory.

In the field of fluid mechanics, the primary goal is often characterizing the two-dimensional flow field [4–6]. This typically involves seeding a fluid with tracer particles and then illuminating it with a laser beam expanded in one direction (usually parallel to the dominant flow direction).

✉ L. Keene
keene6@llnl.gov

[1] Lawrence Livermore National Laboratory, Livermore, CA, USA

Published online: 05 November 2021

Particle image pairs - made visible by their scattering of the laser illumination - are then analyzed either by Laser Speckle Velocimetry (LSV, an interferometric approach leveraging a speckle diffraction effect) if high fluid-particle concentrations exist, or the more general Particle Image Velocimetry (PIV, a statistical analysis approach) for use with either dense or sparse particle concentrations [5]. Both cases fundamentally generate two-dimensional displacement fields as their dataset outputs, which are then optionally post-processed via numerical differentiation to estimate velocity vector fields.

Likewise, the field of solid mechanics has seen a parallel but independent effort to develop and apply similar methods. The DIC method is most often employed and, like PIV, uses statistical analysis to map displacements by cross-correlating small subimage pairs within a predefined search region of optical images [1]. Like its PIV counterpart in fluid mechanics, the DIC approach ultimately outputs a data set characterizing the two-dimensional displacement field which is then optionally post-processed via numerical differentiation to estimate surface strain fields. This approach is often referred to as "white light speckle" due to its reliance on natural surface texture or the application of high-contrast random paint patterns to the surface under study, an approach analogous to seeding the fluid flow with scattering particles in the style of PIV/LSV. Alternatively, a coherent light source such as a laser can be used to illuminate the object surface provided that the surface roughness is greater than the order of the wavelength of the illuminating light. In this case, a stochastic interference pattern known as laser speckle [7] is generated that can be used to uniquely characterize the surface of the object; since the local micro-relief of the surface governs the scattering behavior, and therefore the formation of the speckle pattern itself. The local speckle pattern (i.e., the speckle pattern associated with small subimages extracted from some source image) is anchored to the local relief and translates in response to a translation of the associated surface, thus mirroring its deformation. As in PIV/LSV/DIC, source image pairs consisting of speckle patterns are processed to generate a two-dimensional displacement field which can then be post-processed to estimate the surface strains [8–11]. Additional advances in the field have also enabled the use of sparse data sets for generating approximations of full-field displacements with less computational burden [12–15].

As optical methods have advanced, so have the digital processing capabilities of personal computers. Key architectural enhancements such as out-of-order execution, branch prediction, Single Instruction Multiple Data (SIMD) vector processing, many-core designs and branch prediction have all improved the runtime performance of CPUs [16]. In the last decade some of the largest improvements in runtime performance have been realized in the graphics processing subsystem, specifically the Graphics Processing Unit (GPU). These were initially intended as task-specific processors optimized for rendering graphics output but have gradually evolved into fully programmable processors that, in some cases, are ideally suited to highly parallel computational workloads. To facilitate the application of these devices to more general workloads, several Application Programming Interfaces (APIs) have been introduced to expose this underlying capability to non-graphics programmers. Among the most widespread is Nvidia's CUDA™ GPU API [17]. Applications written using CUDA are executable only on Nvidia GPUs. While these are widespread, this poses a limitation with respect to general compatibility. In response to this restriction, the open-source OpenCL framework was created to enable GPU-specific code to execute on multiple hardware vendor platforms including Nvidia, Intel and AMD [18]. Another, less well-known, framework is a Microsoft API called DirectCompute™ which operates within the Direct3D™ runtime (version 11 or higher) on the Windows operating system and allows low-level programming of pixel shaders to enable computational workloads other than graphics. While powerful, DirectCompute™ was difficult to use for programmers unfamiliar with graphics programming. In response to this, Microsoft created the C++ Accelerated Massive Parallelism (C++ AMP) open specification [19] - in effect, enabling the use of DirectCompute™ with more familiar C++ programming idioms. Microsoft provides an implementation of C++ AMP in the C++ compiler included with all versions of their Visual Studio Integrated Development Environment (IDE). Applications using C++ AMP to target the GPU can execute on any GPU that supports DirectX 11 or higher, regardless of manufacturer. The enhanced version of CASI described in this paper is implemented with Microsoft's C++ AMP compiler.

This paper is organized as follows: the principles of the laser speckle diffraction measurement technique are described, followed by their expression in the form of the CASI I/II algorithms. The mapping of key stages of the base CASI II algorithm to commodity GPU hardware for efficient parallel execution via C++ AMP is described in detail. Two additional modifications to the base algorithm are then proposed: (1) a straightforward method to disambiguate large (but resolvable) displacements from translation aliasing effect, and (2) a subimage-tapering modification involving an oversampling-downsampling approach that greatly extends the measurement range of the original method while preserving the ability to efficiently map the algorithm to GPUs. Results on synthetic speckle patterns are shown and timings for an example GPU are provided.

Principle

Single-Beam Speckle Diffraction Interferometry

Single-beam speckle diffraction interferometry has its origins in the fundamental work of Burch and Tokarski [20], was expanded upon by Archbold et al. [21, 22] and later refined by Khetan and Chiang [23]. Conceived as a full-field non-contact method for determining surface in-plane deformations, the basic experimental setup and procedure is illustrated in Fig. 1. In the case of solid mechanics, this involves illuminating an object with a diffused coherent light source (typically a laser beam passed through a microscope objective such that the emitted beam forms a diverging cone of light) and capturing the subjective speckle pattern on a photographic negative. The object under study is then stressed or perturbed in some way and a second speckle pattern is captured *on the same photographic negative*. This is referred to as a "double exposure specklegram". For fluid mechanics studies implementing LSV, the process is the same with the exception that the two speckle patterns are two reflectance specklegrams of seed particles captured some time interval apart either

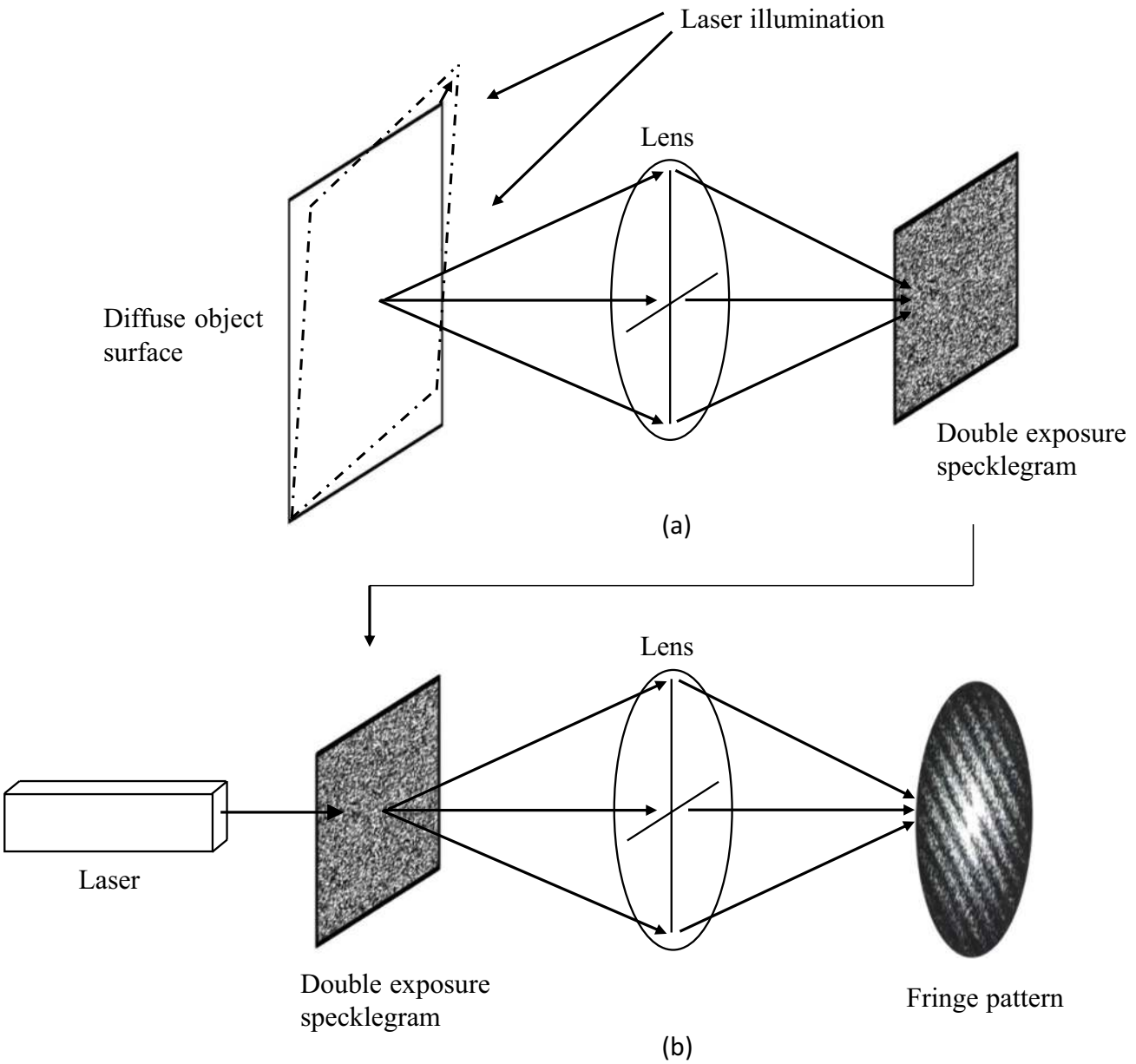


Fig. 1 **a**) Optical arrangement for collection of double exposure specklegram of coherently illuminated object undergoing strain. **b**) Optical processing of double exposure specklegram to generate Young's fringes

by pulsed laser or shuttered continuous wave laser. Once developed, the negative contains a randomly distributed array of varying transparency corresponding to the irradiance pattern of the original speckle patterns. When illuminated by a plane wave source, the photographic negative acts as a random transmittance function causing the impinging wave to diffract. Since the negative is a double exposure containing the photographic superposition of both speckle patterns (reference and perturbed), the transmittance function embodied by the photographic plate is essentially two superimposed non-uniform partially correlated diffraction gratings. The gratings are non-uniform due to the randomness of the speckle pattern, and partially correlated due to the displacement imposed between images. If the displacement between images is not overly large, a spatial correspondence will exist between neighboring speckles. When illuminated by a plane wave such as a collimated laser beam, these speckles act as aperture pairs that diffract the impinging wave. When viewed in the far field, a central diffraction halo modulated by sinusoidal fringes (i.e., Young's fringes) corresponding to the displacement between images can be observed. The fringe orientation is perpendicular to the direction of the local motion within a ±180-degree ambiguity. The object's displacement is inversely proportional to the resulting fringe pitch and is given by [2].

$$Disp = \frac{\lambda F_l}{MP_f} \tag{1}$$

Here P_f is the fringe pitch, λ is the wavelength, F_l is the focal length of the imaging lens used in the optical processing step, M is the system's optical magnification and $Disp$ is the object's displacement at the location of optical processing. In practice, the photographic plate is usually mounted in an XY translation gantry and illuminated by collimated laser beam at many discreet locations. Therefore, one can obtain its two-dimensional displacement field by scanning the beam in an XY plane perpendicular to the laser's optical axis over the entire image of the object, analyzing the resulting fringe pattern at each position. Implicit in this method is the assumption that the displacement is essentially uniform within the illuminated region due to its small size relative to the size of the overall object within the negative. Therefore, care must be taken to ensure that a proper size ratio between object and interrogation spot is maintained. This ratio will be dependent on the target spatial resolution of the resulting displacement-strain field as well as the complexity of the strain field itself; more complex strain fields will require smaller interrogation area. In many cases, successful application of the technique requires several iterations, and some a priori knowledge of the object's approximate deformation is helpful.

CASI I

Taking advantage of the advent of powerful personal computers and CCD cameras of sufficient resolution, Chen and Chiang [24] proposed to reconstruct the speckle diffraction interference using a two-step Fast Fourier Transform (FFT) [25] numerical simulation rather than implementing it via physical optics as described earlier. Here, a brief overview of this method is presented. Approximating the deformed speckle pattern as a shifted version of the reference (i.e., undeformed) speckle pattern, a subimage of the reference speckle pattern is denoted as

$$h_1(x, y) = h(x, y) \tag{2}$$

An equally sized subimage portion of the deformed speckle pattern is approximated as

$$h_2(x, y) = h(x - u, y - v) + n(x, y) \tag{3}$$

where u and v are the displacements in the x and y direction, respectively, and $n(x, y)$ is an additive uncorrelated noise component that is assumed to be small and is often therefore disregarded [24] (the impact of additive noise of various amplitudes on the CASI algorithm is shown later in this work, see Figs. 9 and 10). Furthermore, the spatial spectrum of the reference subimage is written as

$$H(\omega_x, \omega_y) = \iint_{\Delta} h_1(x, y) e^{-j2\pi(x\omega_x + y\omega_y)} dxdy \tag{4}$$

where Δ denotes the subimage area and ω_x and ω_y are spatial frequencies in the x and y directions, respectively. The spectral sum of the reference and displaced speckle patterns can be written as

$$F(\omega_x, \omega_y) = \iint_{\Delta} (h(x, y) + h(x - u, y - v)) e^{-j2\pi(x\omega_x + y\omega_y)} dxdy \tag{5}$$

or alternatively, using the Fourier shift theorem [25], as

$$F(\omega_x, \omega_y) = 2H(\omega_x, \omega_y) e^{-j\pi(u\omega_x + v\omega_y)} \cos\left(\pi\left(u\omega_x + v\omega_y\right)\right) \tag{6}$$

It can be shown [24] that the spectral amplitude of Eq. (6) can be approximated as

$$A_s(\omega_x, \omega_y) \approx \frac{4}{3\pi} A_h(\omega_x, \omega_y)\{1 + 4cos^2\left(\pi\left(u\omega_x + v\omega_y\right)\right)\} \tag{7}$$

where $A_h(\omega_x, \omega_y) = \left|H(\omega_x, \omega_y)\right|$ (often referred to as the diffraction halo). Inspection of Eq. (7) indicates that the resulting spectral amplitude consists of an overall diffraction halo bias modulated by cos^2 fringes i.e., Young's fringes resulting from the interference of the two speckle patterns. Chen and

Chiang [24] proposed processing this result via a second forward Fourier transform to estimate the peak position in the second frequency domain corresponding to the fundamental frequency of the fringes. Based on this description, implementation of CASI I involves the following sequence of processing steps:

Step 1. Extract a subimage (typically 64×64 or 32×32 pixels) from the reference image.
Step 2. Extract an equally sized subimage from the corresponding location in the deformed image.
Step 3. Add the subimages together to create the double exposure.
Step 4. Perform a forward Fourier transform of the double exposure and compute the magnitude of the result.
Step 5. Perform a forward Fourier transform of the result from step 4 and locate the crest of the correlation peak(s). This indicates the direction (with ± 180 degrees ambiguity) and magnitude of the shift between subimages.
Step 6. Shift subimage location and repeat sequence for entire image.

CASI II

Chen and Chiang further refined the method by proposing an alternative algorithm referred to as CASI II [26]. In this approach, corresponding subimages are extracted as in CASI I but processed independently rather than superimposed as a double exposure, and a new spectrum is formed which contains an array of phase differences. The direction of the deformation is indicated by the sign of the primary phase difference, and the magnitude of the deformation is indicated by the periodicity. Again, let the reference subimage be denoted as in Eq. (2) and the displaced subimage denoted as in Eq. (3), where u and v are the displacements in the x and y directions, respectively, and Δ indicates the subimage extraction region within the overall source image. The spectrum of the reference subimage is determined by

$$H_1(\omega_x, \omega_y) = \iint_{\Delta} h_1(x,y) e^{-j2\pi(x\omega_x + y\omega_y)} dxdy \tag{8}$$

Likewise, the spectrum of the displaced subimage is determined by

$$H_2(\omega_x, \omega_y) = \iint_{\Delta} h_2(x,y) e^{-j2\pi(x\omega_x + y\omega_y)} dxdy \tag{9}$$

A new spectrum, in the form of a normalized cross-power spectrum of the two subimage spectra, is obtained by

$$F(f_x, f_y) = \frac{H_1(f_x,f_y) H_2^*(f_x,f_y)}{\sqrt{\left| H_1(f_x,f_y) H_2(f_x,f_y) \right|}} \tag{10}$$

where $*$ indicates the complex conjugate. A second Fourier transform gives rise [26] to the secondary frequency domain (ξ,η) representation of this new spectrum:

$$G(\xi,\eta) = \iint_{\Delta_f} F(f_x,f_y) e^{-j2\pi(f_x\xi + f_y\eta)} df_x df_y \tag{11}$$

By computing the magnitude of $G(\xi,\eta)$, an expanded impulse function centered at the displacement point (u, v) in the second spectral domain is obtained. By locating the crest of this single impulse function, the direction and magnitude of the deformation between speckle patterns is uniquely determined. A detailed exposition of the derivation can be found in [3].

Based on the above description, implementation of CASI II involves the following sequence of steps:

Step 1. Extract a subimage (typically 32×32 pixels) from the reference image.
Step 2. Extract an equally sized subimage from the corresponding location in the deformed image.
Step 3. Perform a forward Fourier transform of the reference subimage.
Step 4. Perform a forward Fourier transform of the displaced subimage.
Step 5. Compute a normalized cross-power spectrum according to Eq. (10).
Step 6. Perform a forward Fourier transform on the complex result from step 5 and locate the crest of the magnitude impulse peak. This uniquely indicates the direction and magnitude of the shift between subimages.
Step 7. Increment subimage location and repeat sequence.

CASI II can uniquely determine the direction directly in the secondary frequency domain, whereas CASI I generates an interference fringe pattern which requires further processing to ascertain the nature of the underlying displacement. The penalty paid for this enhancement is greater computational complexity, primarily the need to compute an additional forward FFT in CASI II vs. CASI I.

Adaptation to GPU Architecture

GPU hardware is designed to efficiently process parallel workloads by scheduling many tens of thousands of independent threads of execution, each one running the same logical execution unit referred to as a "kernel". In most cases, threads execute the kernel method either independently of each other or with minimal synchronization. This

is generally the most efficient model for GPUs, as the flexibility it affords the thread scheduler allows for maximum occupancy. In the case of the CASI II algorithm, however, there exist two granular levels of parallelism: (1) data-level parallelism, and (2) task-level parallelism, as can be seen by examining the algorithmic sequence listed for CASI II. Fundamentally, each subimage pair is processed independently of any other - an example of task-level parallelism. Subimage processing (steps 3, 4 and 6 above) involves the application of multiple 2D FFTs, operations that can be parallelized at the data level. Lastly, Step 5 of CASI II is a pointwise operation over the entire subimage with maximum data-level parallelism. By inspection, CASI II is an algorithm that should respond well to GPU implementation provided that the dual-level parallel granularity can be efficiently mapped to GPU architecture.

Memory Access Optimization

One programmatic mechanism exposed by the C++ AMP standard is a data-partitioning strategy called "tiling". With this approach, the input data is partitioned into logical units processed by clusters of spatially associated threads whose execution can be synchronized, allowing for a degree of task-parallel granularity. Tiles can be up to rank order 3. The extent of the data partitioning (i.e., number of tiles in all dimensions) must be declared at the kernel invocation. In this case the tiles form a natural correspondence to individual subimages in CASI II and are of rank order 2. The declared dimension of the tiles defines the number of threads assigned to them and is equal to the product of the tile's dimensions (its area or volume). This value cannot currently exceed 1024 [19]. GPU threads within a tile group can synchronize among themselves but not among other tiles, and the sequence in which the tiles complete their processing is non-deterministic. Algorithms must be structured with these caveats in mind. Tiling enables the two-tiered parallel granularity required for CASI; the tile itself enforces the task-parallel nature of processing subimage pairs, while the thread group within each tile enforces the data-parallel nature of the actual processing operations. Following is an example of a two-dimensionally tiled kernel invocation using C++ AMP:

```
static const int TileSize = 32;
extent<2> tiledExtent(OutputDataRows * TileSize, OutputDataColumns * TileSize);
parallel_for_each(tiledExtent.tile<TileSize, TileSize>(), [=](tiled_index<TileSize, TileSize> tiledIndex) restrict(amp) {.}
```

In this invocation, *tiledExtent(.)* defines the degree to which the input data is apportioned threads. The user-defined variables *OutputDataRows* and *OutputDataColumns* correspond to the output dimensions (the total number of subimage processing positions within the input images) and are based upon the size of the input images, the subimage size and the shift size (i.e., the pixel offset between adjacent subimage locations).

Since a logical computation unit (tile) will be assigned for each subimage analysis position in the input image(s), the total extent of processing threads requested to process the entire image is the thread area per tile (1024 threads per 32×32 pixel subimage) multiplied by the total number of subimage processing locations in the parent image. Code within the parentheses is executed by each tile thread independently. The entire kernel declaration, beginning with "*parallel_for_each*" can be read as: *For the following, assign a collection of spatial tiles of 'tiledExtent" dimensions, partition each tile into a 2D cluster of threads such that each tile contains "TileSize" number of threads in the X and Y directions, enable each individual thread's global and tile-specific location to be identified by "tiled_index", and restrict execution to the C++ AMP runtime.*

Logical tile partitioning is beneficial not only because it enables task-level parallelism, but also because it enables the use of what is referred to as "tile-static" memory. This is a small region of programmable cache memory separate from the GPU's global memory bank. Similar to a CPU cache, its total capacity is much less than the main memory bank, but it exhibits far greater bandwidth and much lower latency. Unlike CPU cache, however, tile-static memory is not managed automatically and must be declared and managed by the programmer. Once declared within the kernel it is visible only to threads in a given tile (i.e., threads that have been instantiated for cooperative work on a specific task) and can be used as a way for them to share information while simultaneously partitioning a region of high-speed memory for exclusive use by that task's threads. Its use is crucially important when a cluster of threads is cooperatively processing a single task, as it enables thread synchronization and an encapsulated form of data sharing at the task-parallel level [19, 27]. Furthermore, from a performance standpoint, the low latency is highly beneficial if data residing in the global memory store needs to be accessed multiple times; in most cases this is the primary reason for adopting the greater complexity associated with tiling an algorithm. In the implementation described here, a tile size of 32×32 has been used as it provides excellent performance on the hardware tested while also providing for a conceptually straightforward mapping of the CASI algorithm to GPU kernel code. Figure 2 illustrates the spatial relationship between global and tile-static memory, and how individual tile threads reference their locations in tile domain and global data domain spaces. This is an example of computing a tile thread's data domain space coordinate:

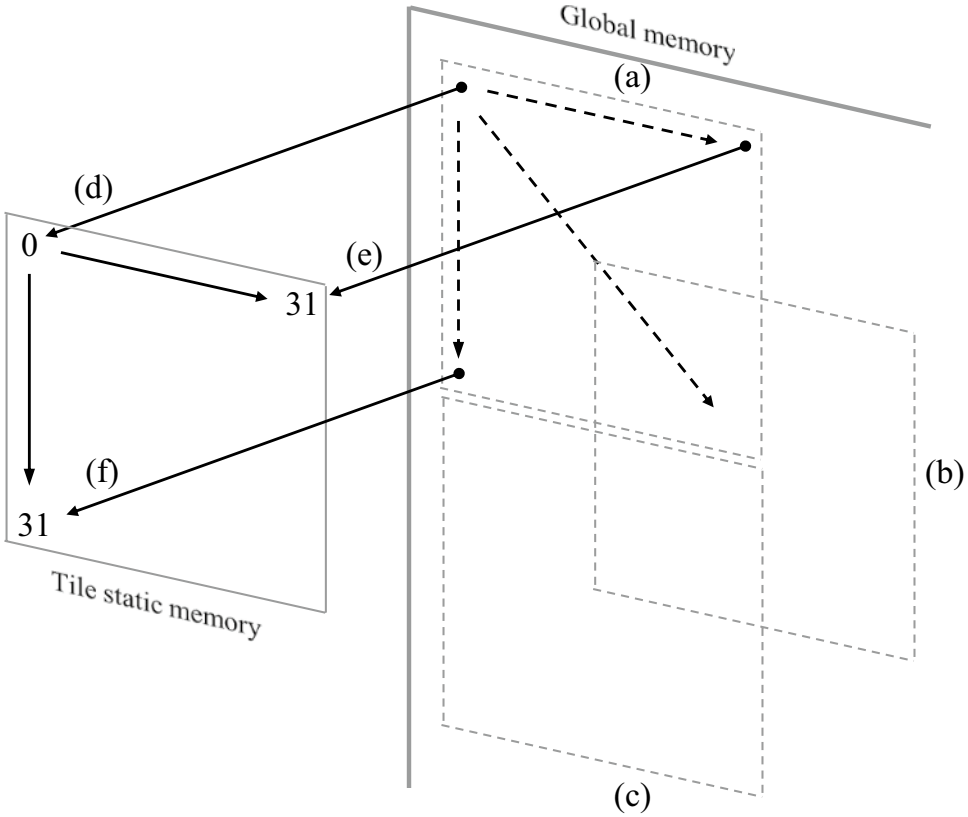


Fig. 2 Spatial relationship between tile-static memory containing subimage data and global memory containing entire image data. Assuming the user-defined 2D tile identifier "tiledIndex", row-column locations for individual tiles are: **a**) tiledIndex.tile[0] = 0, tiledIndex.tile[1] = 0, **b**) tiledIndex.tile[0] = 1, tiledIndex.tile[1] = 1, **c**) tiledIndex.tile[0] = 1, tiledIndex.tile[1] = 0, and individual thread locations within the tile are identified as: **d**) tiledIndex.local[0] = 0, tiledIndex.local[1] = 0, **e**) tiledIndex.local[0] = 0, tiledIndex.local[1] = 31, and **f**) tiledIndex.local[0] = 31, tiledIndex.local[1] = 0

int DataDomainRow = (*tiledIndex.tile*[0] ∗ *Shift*)
+ *tiledIndex.local*[0]; // *Global memory row*

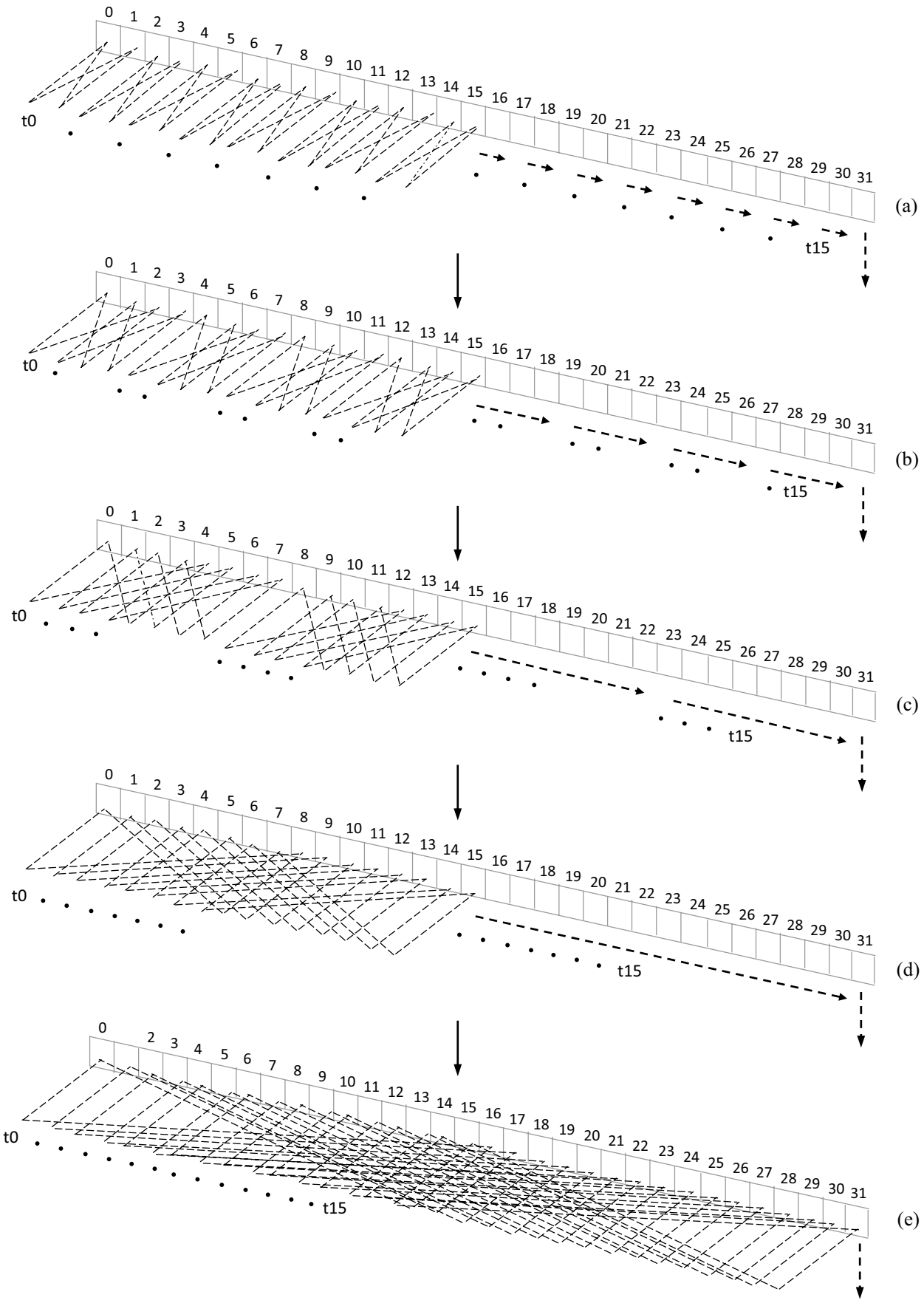
int DataDomainColumn = (*tiledIndex.tile*[1] ∗ *Shift*)
+ *tiledIndex.local*[1]; // *Global memory column*

where *tiledIndex* is the index identifier declared in the kernel invocation, and *Shift* is a user-defined integer value corresponding to the size of the spatial shift (in pixels) from one subimage analysis location to the next in the parent images.

Spectral Processing

An examination of the CASI II algorithm indicates the first major processing step in the sequence is a pair of 2D forward FFTs. This is an iterative operation where data is repeatedly read, shuffled, processed, and stored in-place, and thus makes optimal use of the cached tiling strategy previously described. Each forward FFT operation generates an equally sized array of complex results. Therefore, four 32×32 tile-static caches are needed to store the subimage spectra: two for each subimage FFT operation (one to store the real component and one to store the imaginary component of the complex result).

The FFT algorithm used for this work is a decimation-in-time radix-2 implementation with bit-reversed input ordering [28]. The bit-reversed input ordering requires that the input data be re-ordered prior to use, and this is typically done at runtime by bit-reversing the array input indices as they appear in their original order. Since the subimage dimensions are fixed in this implementation, the input ordering that would result from bit-reversal is predetermined and hard-coded in predefined *struct* of integers, simplifying implementation and improving readability. The input data is then reordered in row-wise fashion according to the index values in this *struct* in preparation for FFT processing. In similar fashion, the multiplicative "twiddle" factors needed for the FFT computation [29] are stored in predefined float structures and used during the computation rather than computed at runtime. Figure 3(a-e) shows the progressive signal flow of the radix-2 algorithm, as well as the topological progression of the classic trapezoidal "butterfly" data access pattern, at the tile-thread level for the first row of a subimage tile, from the first pass of the row-wise FFT (Fig. 3(a)) to the final pass (Fig. 3(e)). By inspection it can be seen that for a given butterfly pattern, two input values are required to generate two output values per pass of the algorithm, which are then stored in-place (this can be seen most clearly in Fig. 3(a)). The original input values must be known in order to generate the two output values of the butterfly pattern. Due to the use of in-place storage, if more than one thread were responsible for storing the intermittent results of each butterfly pattern (per pass), the values at those


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
t0
t15
(a)
(b)
(c)
(d)
(e)

◂**Fig. 3** Tile-threaded processing pattern for row-wise FFT of 32 x 32 subimage tile for **a**) first stage, to **e**) final stage. "t0..t15" denote thread 0 to thread 15 of a given tile row

memory locations at any moment become non-deterministic. The excessive thread synchronization required to avoid this state – known as a "race condition" – would negatively impact performance. Therefore, a single thread is assigned to each butterfly instance. Since each thread is responsible for generating two output values, only half of the available tile threads (threads are indicated in the figure by "t0" for the first tile thread, up to "t15" for the sixteenth tile thread) have been apportioned to process the subimage FFT. Since this results in a thread occupancy level of 50 % for the tile, maximum thread occupancy - and efficiency - is achieved by assigning the first 512 tile threads to the 2D FFT for the first (reference) subimage, and the other 512 threads in the tile to the 2D FFT of the second (displaced) subimage. Each cluster of tile threads therefore processes two FFTs simultaneously with results stored in-place, and the ideal thread occupancy level is achieved. This pattern assumes the input data has been re-ordered (bit-reversed). While only the first-row processing is shown in Fig. 3, all rows of the subimage tile are processed quasi-concurrently and in the identical pattern to that shown. Column-wise FFT processing proceeds in the same topological stages and with the same thread distribution.

Interference Spectrum and Parallel Reduction

Computing the interference spectrum according to Eq. (10) is a pointwise operation over the entire subimage tile with maximum thread occupancy. In other words, each tile thread generates one complex result corresponding to that thread's row-column location in the tile, with the operation occurring in parallel over the entire subimage. The spectrum of the reference subimage tile will be reused in a further processing step and is not overwritten by any following intermediate results. Instead, the results from the interference spectrum calculation are stored in the tile cache containing the forward FFT of the displaced subimage, overwriting the spectral data (it will no longer be needed after computing the interference spectrum) but preserving the reference subimage tile spectrum. This bypasses the need to instantiate additional scarce tile-static memory. A second 2D forward FFT is then applied to this result in the same manner as previously described, thus generating the 2nd complex spectral domain representation and storing it in-place in the displaced subimage tile-static cache. This is then processed in a pointwise manner with maximum thread occupancy (again, in-place) to compute the complex magnitude.

The magnitude field of the interference spectrum contains the peak indicating the displacement between the reference and displaced subimages, and so a reduction step is necessary to determine the row-column location of the peak. Tile-static cache consists of a series of interleaved memory banks of a certain width [19]. This is hardware independent but is most commonly 32-bits wide. Figure 4(a) shows the most likely configuration of the memory banks as they pertain to tile-static caches of the size being used here. All tile cache in the algorithm described in this work is declared as single precision floating point, with each column of the tile a single 32-bit wide memory bank. Memory access can only be parallelized by the thread scheduler if they are attempting to read/write different banks. Otherwise, they are serialized – a condition known as memory bank conflict. If this condition arises continuously, it has a negative impact on runtime performance. In this work, the parallel reduction of the tile cache data is performed in such a manner as to minimize memory bank conflicts. This is illustrated in Fig. 4(b). The tile cache is first scanned by the topmost row of tile threads in a vertical column-wise fashion, one thread per memory bank. Each thread determines the row number and magnitude of that column's maximum value. These results (row and magnitude) are stored in the first two rows of their respective column in the tile-static cache, and a second pass consisting of four tile threads is used to further reduce the previous results to four candidate values (Fig. 4(c)). Note that - due to the horizontal reduction operation of this second stage - the column number associated with the maximum value must also be stored alongside the row number. Lastly, a single thread determines the row and column coordinate associated with the maximum of these four values. This indicates the location of the impulse function, thus obtaining the displacement at the given subimage processing location.

CASI II Range Extension

Disambiguating spatial shift aliasing

In addition to its non-contact nature and tolerance of environmental noise, one of the benefits of the CASI II algorithm is its ability to resolve in-plane displacements that are large relative to the subimage dimension(s). This can be illustrated via numerical experiment. The method described by Goodman [7] for numerically simulating speckle formation by free space propagation is used to generate synthetic objective speckle patterns to evaluate CASI II performance characteristics. Goodman's method involves instantiating an array of zeros and then populating a given rectangular region of the array with a random phasor field. Computing the intensity of the array's frequency domain representation simulates the stochastic nature and appearance of an optical speckle pattern, with the initial size of the array governing the size of the resulting image and the size of the random phasor field

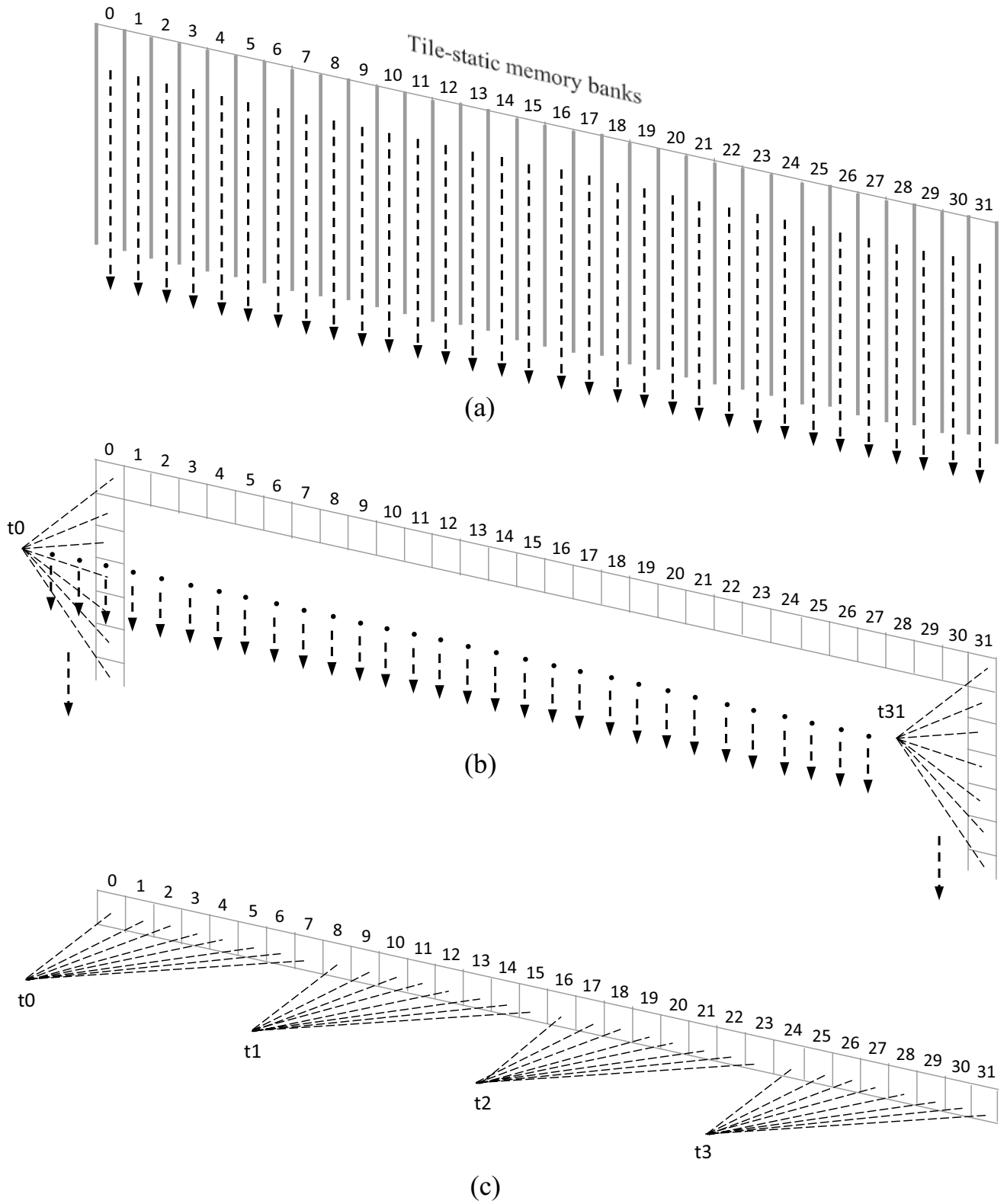


Fig. 4 Parallel reduction of 32 x 32 subimage tile. **a**) Memory bank configuration for tile-static cache used for subimages. Discreet banks are numbered 0 – 31 and correspond to subimage tile columns. **b**) First pass column-wise reduction using 32 tile threads. Each thread process a single column. Results from each column are stored in top two rows of associated column. **c**) Second pass horizontal reduction using four tile threads. A final pass with a single thread reduces these four results to the final one (not shown)

governing the average size of the speckle. Figure 5 shows three examples of synthetic speckle patterns - corresponding to three different speckle sizes - generated using this method. All synthetic speckle patterns used in this study are generated as sixteen-bit grayscale images.

To illustrate the resolvable noise-free range resolution, a synthetic speckle pattern with an average speckle size of 4 pixels was generated and then subjected to sequential rigid body integer translations of 0, 15, 30 and 40 pixels in the XY directions. This amounted to a trivial remapping of pixels which introduces no noise or resampling error. This sequence was paired with the initial untranslated image and CASI II is applied to a centrally extracted 64×64-pixel subimage pair. The results are shown in Fig. 6, which depicts the magnitude field of the second spectral domain (step 6 of the CASI II algorithm) resulting from each rigid body translation. The signal peak position translates in accordance with the relative rigid body displacement between the two subimages, while its magnitude decreases as initially correlated speckles move out of the subimage reference frame and new uncorrelated speckles enter. Even at a relatively large relative displacement of 40 pixels in both X and Y directions – corresponding to only a 14 % shared subimage area (Fig. 6(d)) – a peak is still resolvable among the background random peaks in the magnitude field. While useful, this large measurement range introduces an ambiguity when interpreting the result: provided the signal to noise ratio of the two speckle patterns is sufficient, when the relative displacement between subimages exceeds half the subimage dimension in either direction, the signal peak can be seen to wrap around to the opposite side of the magnitude field. As a result, while CASI II uniquely resolves the direction of the displacement when compared to CASI I, this is only true for displacements that do not exceed half of the subimage dimension in either direction. Displacements in the U and V directions exceeding this magnitude often manifest under the alias of a displacement in opposite direction to the true displacement, differing in magnitude by a factor equal to the subimage size. Using the engineering convention of positive U displacement being in the positive X direction and positive V displacement being in the negative Y direction (where positive Y is typically downwards from the standpoint of image coordinates), the aliased shifts can then be written as

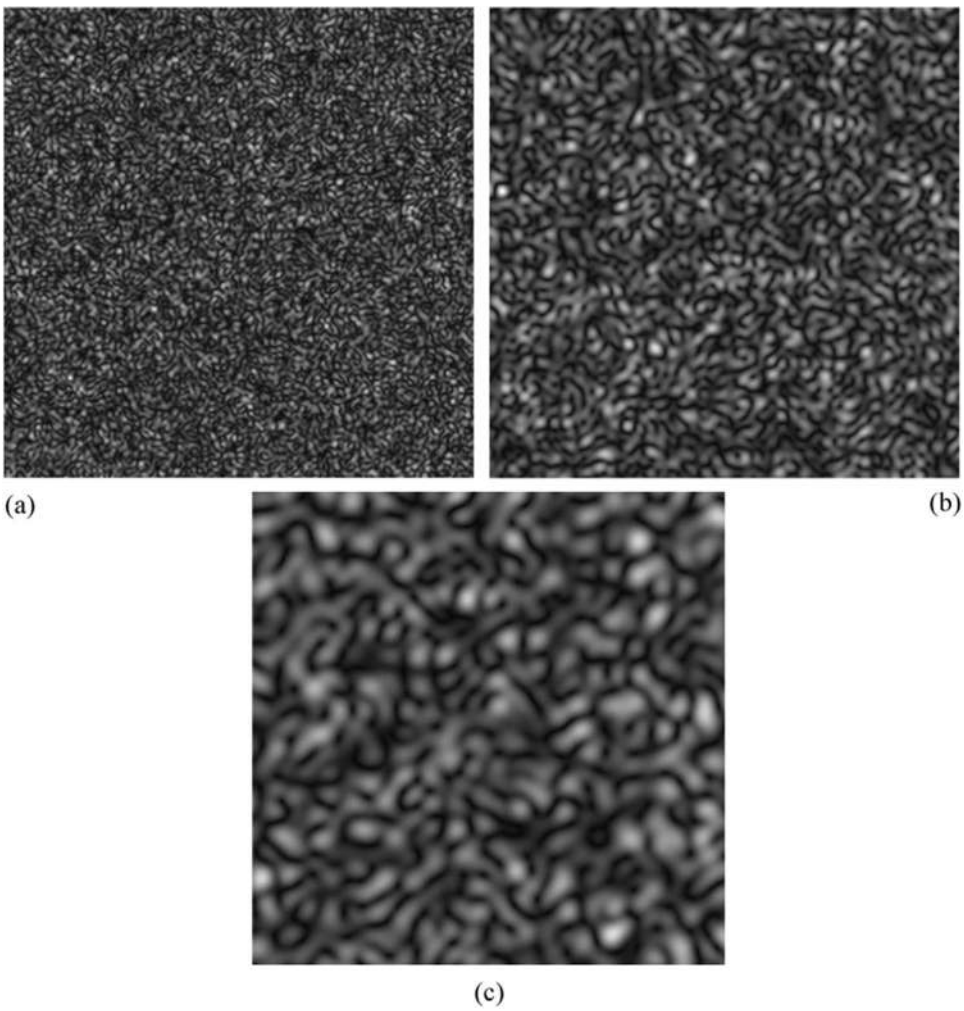


Fig. 5 Examples of numerically simulated far field speckle patterns for average sized speckles of **a**) 4 pixels per speckle, **b**) 8 pixels per speckle, and **c**) 16 pixels per speckle

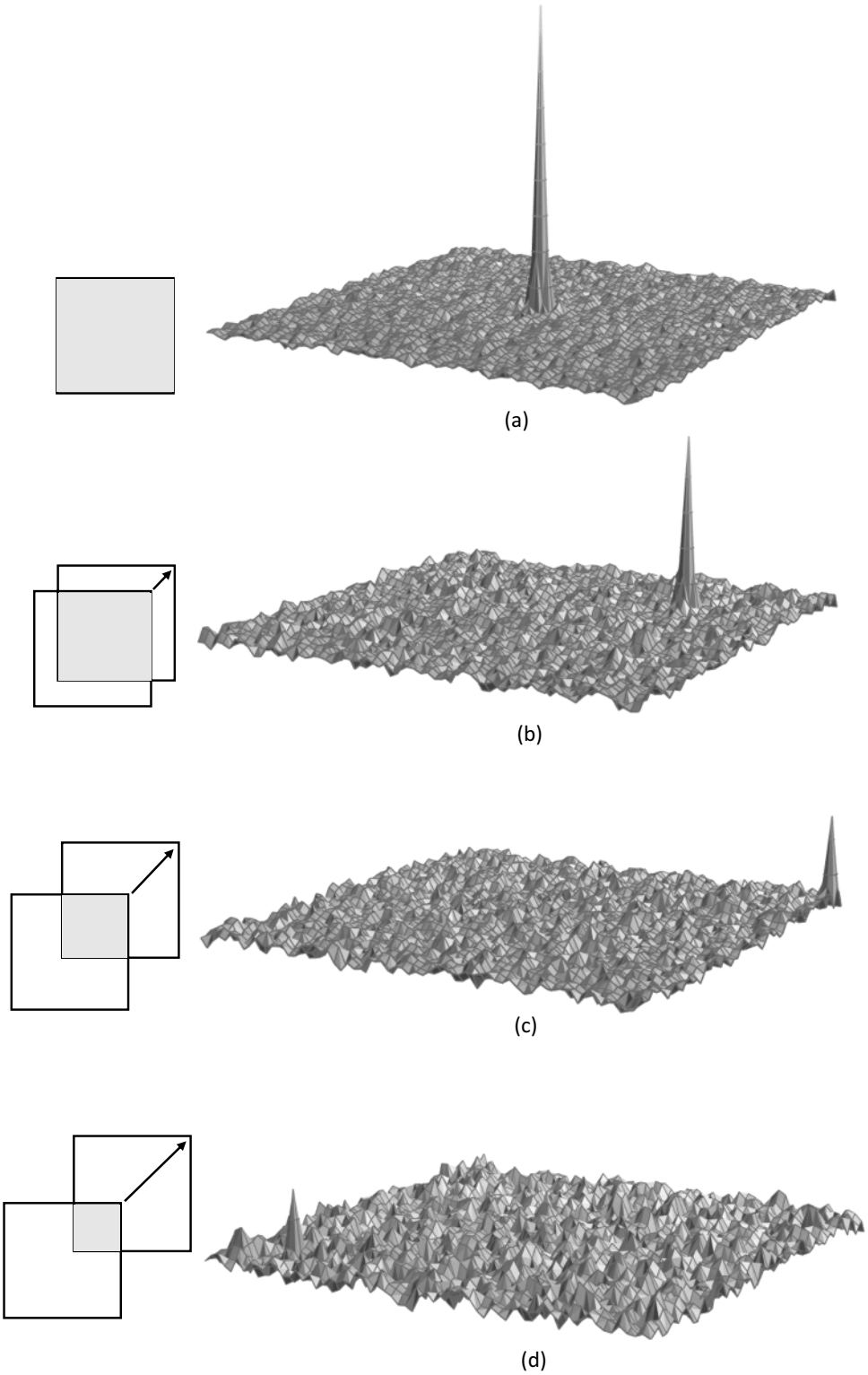


Fig. 6 Magnitude values resulting from application of CASI II to a 64x64 synthetic speckle pattern with sequentially imposed rigid-body UV shifts of **a**) 0, **b**) 15, **c**) 30 and **d**) 40 pixels. The gray region depicts the area of correspondence (the shared region) between the two equally sized subimages after one subimage has undergone a displacement relative to the other. As this region decreases, so too does the peak signal amplitude

$$U_a = \pm(U_d - S_s) \tag{12}$$

$$V_a = \pm(V_d - S_s) \tag{13}$$

Where U_a and V_a denote the aliased displacement in the U and V directions, U_d and V_d denote the determined U and V displacements resulting from the application of CASI II, and S_s is the size of the subimage (always a square) with

its sign being the negative of the associated determined displacements.

To disambiguate the determined displacements from their aliased counterparts and arrive at the true direction of the shift, a straightforward polling modification to the original algorithm is proposed: (1) CASI II is first implemented as described and the U and V displacements are determined. (2) If either one is found to exceed the threshold, Eq. (12) are used to estimate the aliased displacements. (3) CASI II is then repeated twice, first by pairing the reference subimage with a subimage extracted from the determined UV displacement location, and secondly, with a subimage extracted from the computed aliased displacement location (rounding to the nearest pixel). (4) The signal peak amplitude from both locations is then used to resolve the final displacement directions. A conservative threshold value that has been found to work well is 50 % of the determinable displacement for a given subimage i.e., for a 32×32 subimage, if the magnitude of the determined displacement in either direction is found to exceed 8, the polling modification just described is invoked.

This modification is illustrated in Fig. 7. A simulated subjective speckle pattern was generated using Goodman's algorithm [7] with width and height of 2048 and average speckle size of 4 pixels. This speckle pattern was then resampled in the horizontal direction using bicubic resampling to a new width of 2096 and then cropped to the original dimension to simulate an imposed strain of approximately 2.3 %. A 64×64-pixel subimage was then extracted starting from row 992 and column 1632 such that application of the CASI II algorithm results in a weak - but clearly detectable - signal peak residing at column 7 in the output magnitude array (Fig. 7(a)). The location of this signal peak erroneously indicates a relative displacement of -25 pixels between reference and displaced subimages (as originally written, the CASI II algorithm interprets this result as indicating a leftward shift of the subimage due to the location of the peak relative to the center of the magnitude field). The magnitude of this displacement exceeds the threshold convention used in this work; the thresholded region for reliable displacement determination is defined as a central portion of the output array with dimensions 32×32 for a 64×64 subimage. The described polling modification is then invoked to disambiguate the true displacement from its aliased counterpart. Figure 7(b and c) illustrate the results from application of CASI II for the determined displacement (b) and aliased displacement (c) as computed by Eq. (12a). These clearly indicate that the true displacement is in the positive X direction as expected from the resampling operation, and not the opposite direction as initially indicated in (a).

There is a penalty associated with this modification due to the additional processing of subimage pairs. Recall that 50 % of tile threads are used to perform a single 2D FFT in this implementation. Recognizing that the reference subimage spectrum has been preserved in the GPU cache - and is therefore available for reuse - full thread occupancy could, in principle, be achieved by targeting half of the tile threads to compute the spectrum of the subimage at the determined location, and the other half the spectrum of the subimage at the aliased location. This would, however, require the instantiation of an additional two tile-static arrays at the time of kernel initiation. For the implementation described in this work, the decision was made to favor cache-compactness over thread occupancy for two reasons: (1) GPU cache memory is currently much scarcer than global memory and should be used sparingly, and (2) while the additional cache memory would be allocated at the initiation of the kernel, there would be no guarantee of its actual need since it would only be called upon in the case where a determined displacement is found to exceed the predefined threshold. Therefore, the determined and aliased FFT analyses are performed in serial fashion and as separate steps by half (512) of the tile threads. As will be shown later, this approach demonstrates good runtime performance.

Sample-rate converted coarse-fine decomposition

A practice sometimes encountered in the application of digital full-field methods is to reduce the shift size between subimage analysis points in the belief that this will yield greater displacement field resolution. A shift size equal to (or, frequently, half of) the radius of the subimage is, for most practical purposes, sampling the displacement field at sufficient resolution relative to the subimage size. Uniform displacement is always assumed within the subimage; the only true method for increasing spatial resolution is to reduce the size of the subimage relative to the imaged surface. Maintaining the subimage size while reducing the shift size beyond the 50 % factor amounts to a form of oversampling: the approach yields greater information, but it is not useful. In this work, a different approach is taken. The spatial resolution of CASI II is maintained but the measurement range is increased even further beyond the previously described modification by incorporating a coarse-fine decomposition step into the original algorithm. This involves a two-step process wherein an initial coarse estimate of the displacement is generated and then further refined to the subpixel level. The initial coarse estimate is generated using CASI II with a larger subimage centered about the smaller subimage (which is used for fine displacement estimation). Due to its size, the larger subimage is capable of resolving larger displacements but with less spatial resolution, while the smaller subimage used in the second step resolves finer spatial resolution at the cost of decreased measurement range. By combining the two, the effective range can be increased without sacrificing spatial resolution. The process, as implemented here, involves extracting a 64×64 subimage centered at each

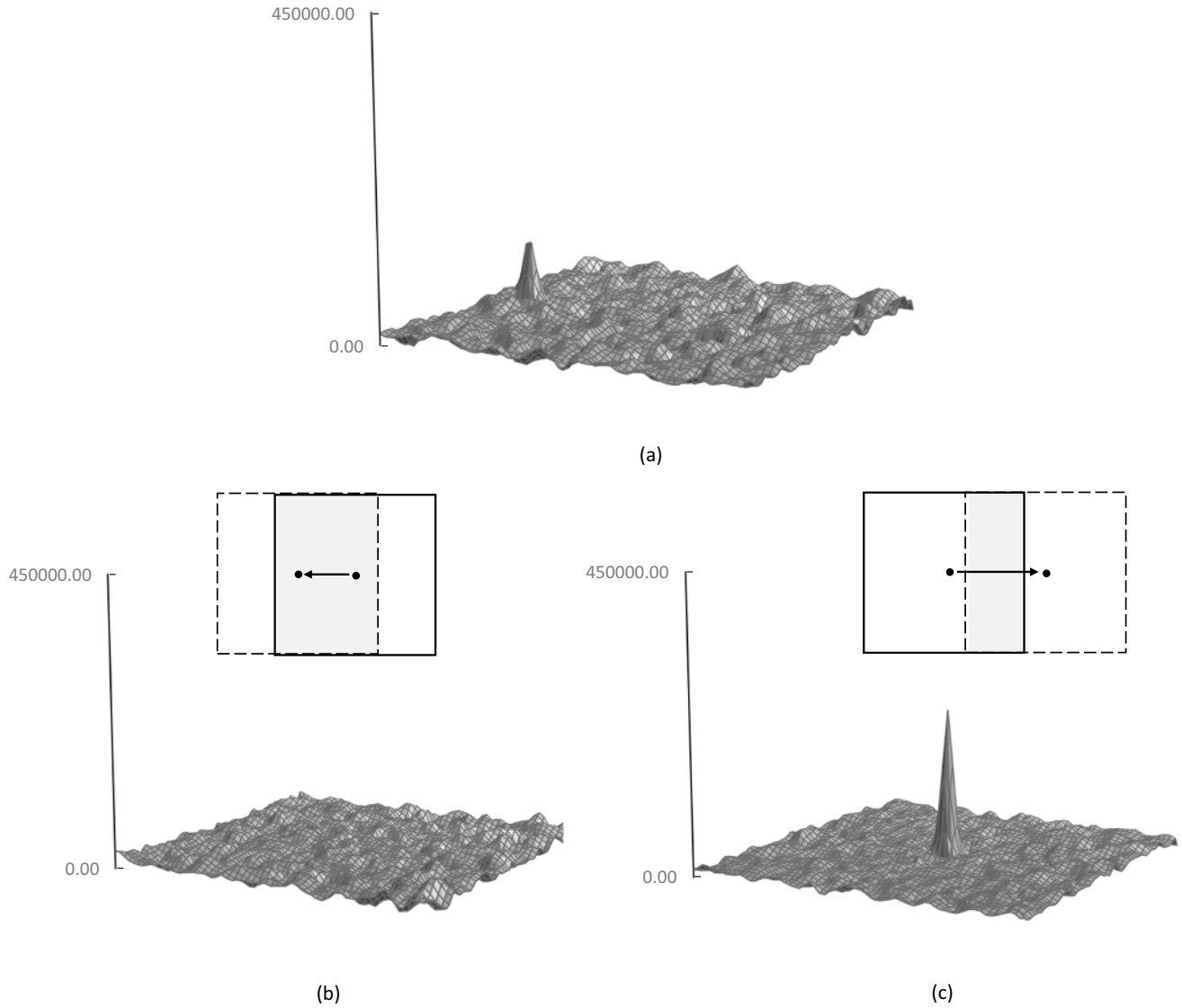


Fig. 7 (**a**) 64 x 64 magnitude output array from initial application of CASI II showing attenuated peak at column 7 row 16, indicating a shift in –X direction of 25 pixels. (**b**) CASI II output array resulting from pairing of reference subimage and subimage extracted at displacement determined from (**a**). (**c**) CASI II output array resulting from pairing of reference subimage with subimage extracted from computed aliased location. Strong signal peak is visible, indicating true displacement

analysis location and, using CASI II in combination with the aliased displacement step described previously, determine the pixel-level displacement. A 32×32 subimage centered at this displacement estimate is then extracted and processed with CASI II against a reference 32×32 subimage centered at the original analysis position. The final displacement is simply the sum of the determined displacements from both steps. The determined displacement from the coarse step should be thought of as a rough estimate since uniform displacement within the subimage is an implicit assumption in this technique. In most cases, the method is useful in bringing the smaller 32×32 subimage within sufficient proximity to the original reference location so that a signal peak of sufficient signal-to-noise ratio can be recovered. The aliasing correction step described previously is only applied to the larger 64×64 subimage. Here it is assumed that the coarse adjustment is sufficient to reduce the determined displacement of the smaller secondary subimage to within threshold limits, negating the need for a second application of the aliasing correction (and its associated computational overhead).

There are two drawbacks to this approach: (1) the additional computational burden involved in processing the larger subimage will negatively impact the runtime performance, and (2) the larger subimage requires a tile-static GPU cache of equivalent dimensions, which cannot be allocated (32×32 being the largest total size available as of this writing). To mitigate

these issues, a sample rate conversion step is proposed wherein the larger 64 × 64 subimage is downsampled to 32 × 32. This compact representation allows for the (re)use of 32 × 32 tile-static caches allocated for the implementation described up to this point, while also minimizing the impact of the additional computation on runtime performance. Care must be taken when downsampling, however; overlap can occur between spectral replications in the frequency domain, corrupting the higher frequencies and leading to unreliable results [29, 30]. Representing a 64 × 64 subimage in a 32 × 32 cache requires that the subimage be downsampled by a factor of 2×. Shannon sampling theorem [30] stipulates that the sampling rate must exceed the highest frequency component in the spectrum by 2 if aliasing is to be avoided. Therefore, it is proposed that the size of the speckles be adjusted such that they are oversampled by 2× when initially imaged onto the sensor. Downsampling this (oversampled) speckle pattern should, in principle, avoid aliasing.

To implement this strategy, the target average speckle size must be known *a priori*. The average subjective speckle size in the direction normal to the optical axis of the imaging system is given by [3]:

$$S = \frac{1.22\lambda q}{D} \tag{14}$$

where S is the speckle size, λ is the wavelength of the illumination, q is the distance from the imaging lens to the sensor focal plane, and D is the diameter of the imaging lens aperture. From Eq. (13) it can be seen that the size of a subjective speckle is inversely proportional to the imaging aperture. Therefore, it is a straightforward matter to alter the size of the subjective speckles by adjusting the aperture of the imaging lens such that they are oversampled by the camera sensor. The target size that brings about this degree of oversampling of the speckle must be determined. Chen and Chiang [3, 31] have applied Shannon sampling theorem to determine the optimal sampling rate for a subjective speckle pattern. They have shown the spectrum of a speckle pattern is bandlimited, i.e., non-zero only in a circular region defined as:

$$\sqrt{{\omega_x}^2 + {\omega_y}^2} = \Omega \tag{15}$$

where ω_x and ω_y are spectral frequency components in the X and Y directions, respectively, and

$$\Omega = \frac{D}{\lambda q} \tag{16}$$

From Shannon sampling theorem [30], a sampled function produces spectral replications at regular intervals of T^{-1} where T is the sampling interval. To avoid aliasing, the following condition relating T to the non-zero speckle pattern spectral region Ω must be satisfied:

$$\frac{1}{T} > 2\Omega \tag{17}$$

By substitution, the Nyquist sampling interval for a subjective speckle pattern can be written as

$$\frac{1}{T} = 2\left(\frac{D}{\lambda q}\right) \tag{18}$$

Rearrangement of Eq. (13) yields

$$D = \frac{1.22\lambda q}{S} \tag{19}$$

After substitution of Eq. (17) into Eq. (16), and rearrangement, an expression for the Nyquist sampling interval relative to speckle size is

$$T = 0.41S \tag{20}$$

Therefore, to avoid spectral corruption from the sample rate conversion step, the average speckle size must be at least 4-5 pixels (twice the optimal sampling interval given in Eq. (18)). As noted previously, this is accomplished via adjustment of the lens aperture prior to image collection.

To evaluate the effectiveness of sample-rate converted coarse-fine decomposition, two 2080 × 2080-pixel synthetic speckle patterns were generated: one with an average speckle size of 2 pixels (corresponding to an optimally sampled speckle pattern as per Eq. (18)) and one with an average speckle size of 4 pixels (corresponding to a speckle pattern oversampled by 2×). Both speckle patterns were resized in the X and Y directions to 2110 × 2110 pixels using bicubic resampling kernel, corresponding to approximately 1.5 % strain. These images were then processed with three iterations of CASI II. Figure 8 shows the results in the form of displacement magnitude i.e., the Euclidean distance arising from the determined U and V displacement fields. Figure 8(a) depicts the results from processing via the original CASI II algorithm utilizing a 32 × 32 subimage with the optimally sampled (speckle size 2) image pair. The aliased shift effect appears as mirroring in the displacement field when the magnitude of the displacement exceeds the subimage radius. Signal peak decorrelation is manifested as noise in the displacement field and dominates the larger displacement region. Figure 8(b) depicts the displacement magnitude output data generated using the algorithm enhancements described previously, but now applied to the same optimally sampled speckle patterns as those used in Fig. 8(a). The aliasing shift effect is corrected. However, due to the downsampling operation being applied to an optimally sampled pattern, the spectra of the subimages have been corrupted by aliasing-induced downsampling, resulting in a

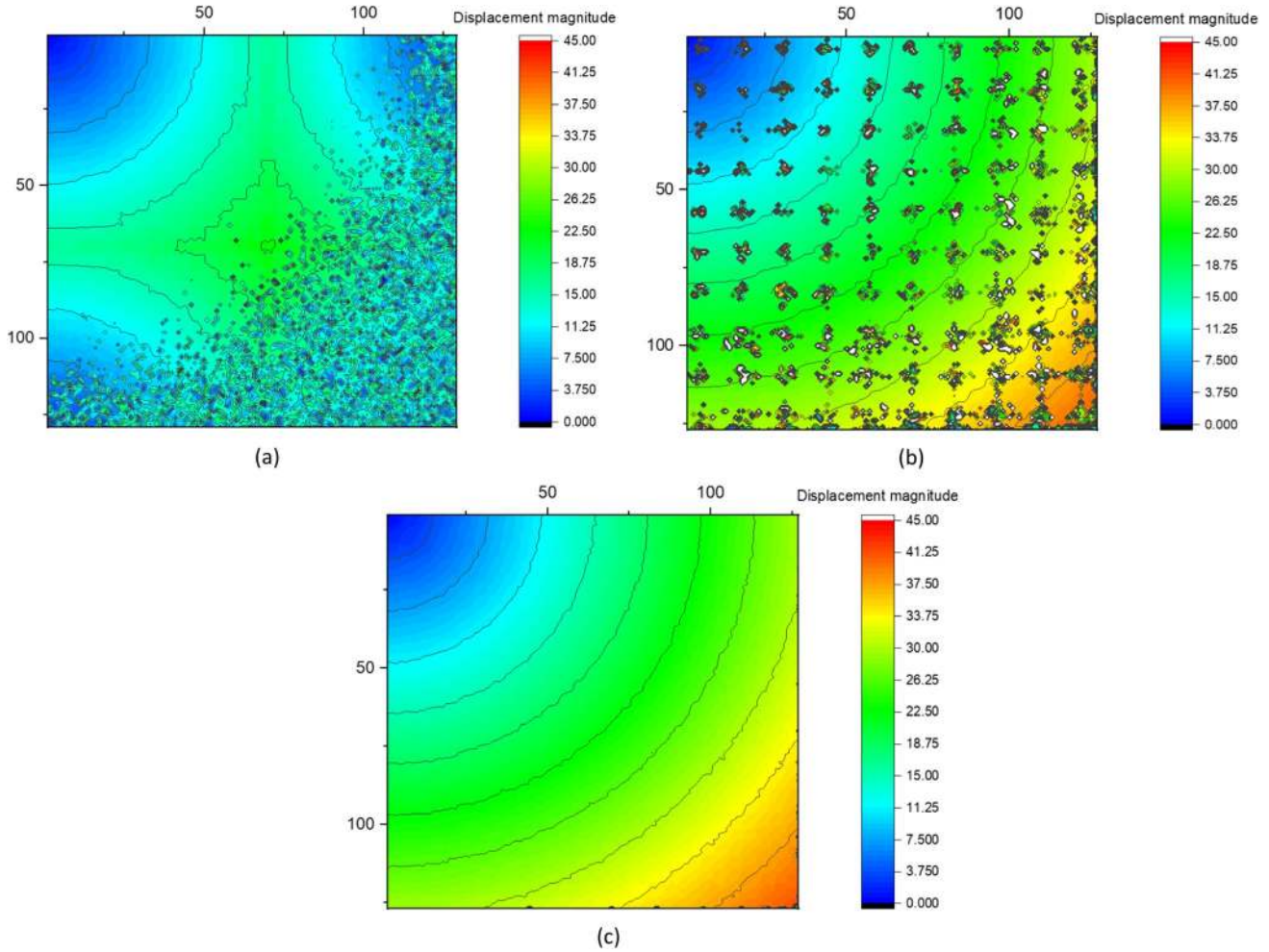


Fig. 8 Displacement magnitude fields using synthetically generated speckle patterns with 1.5% applied strain in X and Y directions. (**a**) Speckle size 2 (optimally sampled) original CASI II algorithm with spatial shift aliasing visible as mirroring in the output data as well as large-scale decorrelation as relative displacement increases. (**b**) Results from proposed algorithm manifesting spectral corruption due to sample aliased rate conversion of initial size 2 speckles from (**a**). (**c**) Results from proposed algorithm with size 4 speckle pattern (2X oversampled) and sample rate converted via 2X downsampling showing extended measurement range up to approx. 40 pixels displacement

quasi-periodic noise pattern in the output data. Figure 8(c) depicts the displacement output data generated using the algorithm enhancements described earlier but applied to the proposed 2× oversampled speckle patterns. Both the aliasing shift effect and the periodic output noise are eliminated, while the effective measurement range has been improved by a factor of approximately 2.5×.

An additional simulation was conducted to evaluate the noise tolerance of both implementations. The same simulated (noise-free) speckle patterns were used as in the previous example but corrupted by gaussian noise so as to model the effect of additive thermal-electronic noise that can arise in digital camera amplification circuits. Three levels of additive noise were used, defined as percentages of the image's full-scale dynamic range: 20 %, 40 % and 60 %. These values were then assigned to the fourth standard deviation of the gaussian curve to adjust the width of the noise distribution. Since the simulated test images were computed to sixteen-bit dynamic range with full scale of 0 – 65,535, the three additive noise levels of 20 %, 40 % and 60 % equated to 13,107, 26,214 and 39,321 at $4\sigma$of the gaussian. Separate noise fields were computed for each image in the sequence to simulate the temporal variation of the noise level in the speckle pattern during image capture. Examples are shown in Fig. 9. The images were then analyzed and the Euclidean distance resulting from the X and Y displacement fields was computed. Results from the analysis are shown in Fig. 10. At all noise levels, the proposed version demonstrates enhanced noise tolerance over the original.

While no changes have been made to the core algorithm of CASI II, the enhancements proposed in this work require larger speckles than would otherwise be considered

optimal [31], which may impact subpixel behavior of the algorithm. To evaluate sub-pixel performance, synthetically generated speckle patterns with dimensions of 2048 × 2048 pixels and average speckle size of 4 – 5 pixels were progressively shifted in the x direction via bicubic resampling in steps of 0.1 pixels through two ranges: (a) 0.1 – 1.0 pixels, and (b) 20.1 – 21.0 pixels. At each step, the enhanced CASI II algorithm was applied to the entire image using a shift value of 16 pixels. The average of all results in the computed displacement field at each step was then computed along with their standard deviation. Results are shown in Fig. 11. At the minimal displacement range (Fig. 11(a)) a slight bias can be seen wherein the results tend to underestimate the imposed displacements. At the larger range (Fig. 11(b)) this bias appears to be resolved. The average standard deviation found for the displacement range shown in Fig. 11(a) is 0.0351 pixels and for the range shown in Fig. 11(b) the average standard deviation is 0.051 pixels. The original CASI II paper [26] claimed a theoretical subpixel resolution on the order of 0.02 of the average speckle diameter. Assuming optimal sampling in that case, the claimed theoretical displacement resolution would be on the order of 0.04 pixels (optimal speckle size being approximately 2 pixels in the original publication). In the case of the modified version proposed in this work – and assuming an average speckle size of approximately 5 pixels – the theoretical displacement resolution would be on the order of 0.1 pixels. By inspection of the results shown in Fig. 11 and factoring in the uncertainty induced by the standard deviations, this appears to be the case.

The final proposed CASI II algorithm is illustrated in Fig. 12.

Computational Runtime Performance

Achieving interactive runtime performance for full-field analysis is a key objective of this work and, therefore, some performance metrics are now presented. Computational performance is, however, highly dependent on the data set

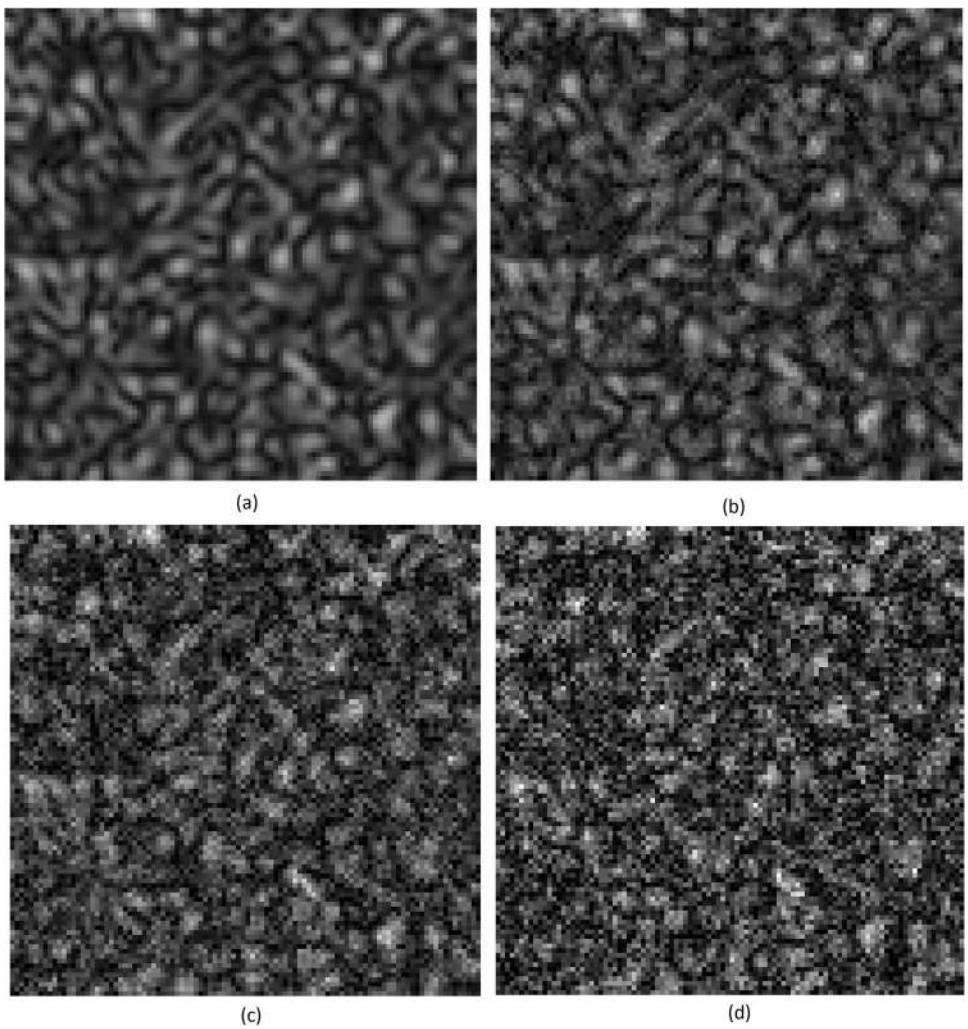


Fig. 9 Synthetic speckle pattern with varying levels of additive gaussian noise (specified as percent full-scale at 4th standard deviation of gaussian). (**a**) noise-free, (**b**) 20%, (**c**) 40%, (**d**) 60% full-scale additive gaussian noise

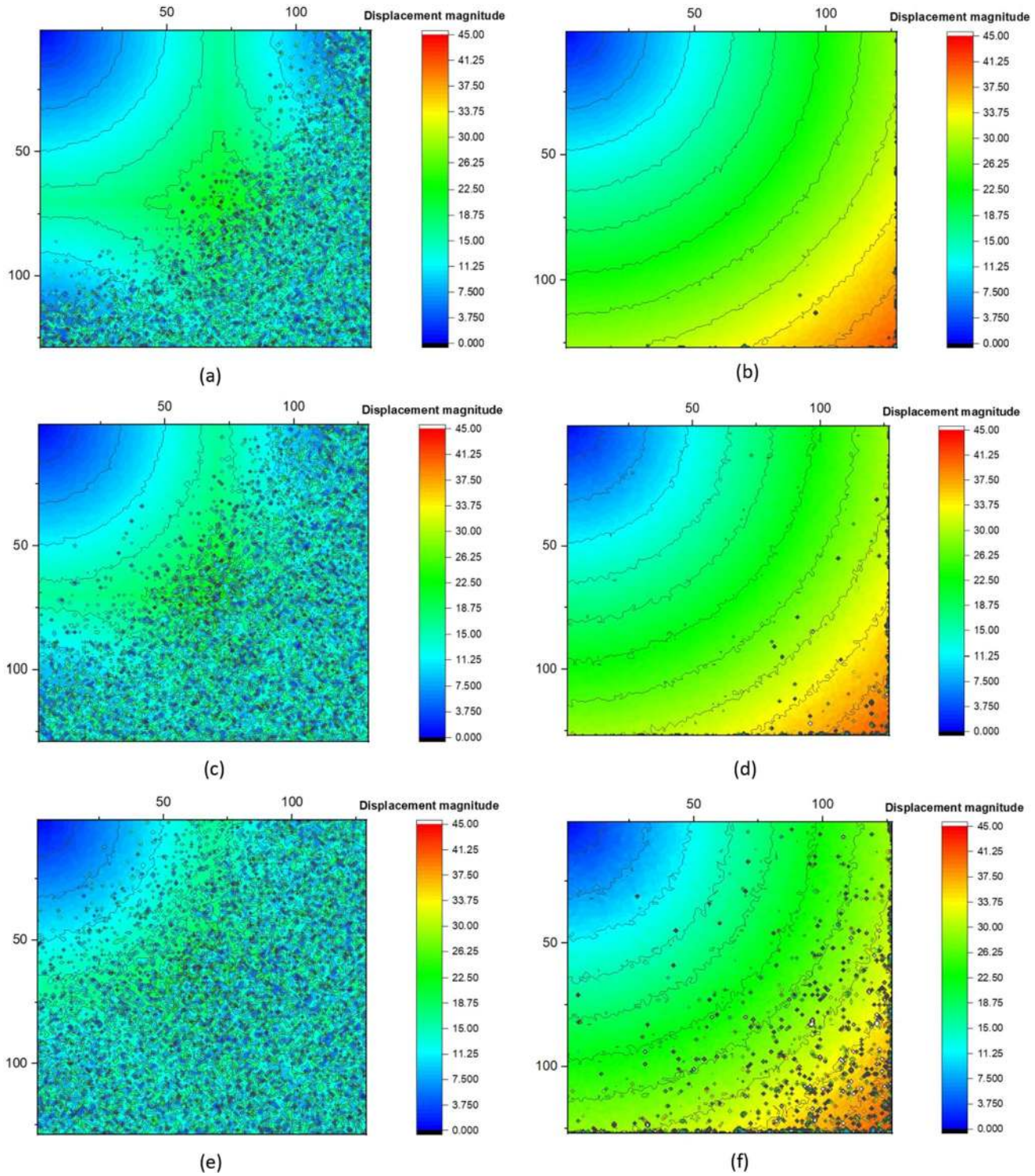


Fig. 10 Displacement magnitude fields (in units of pixels) using synthetically generated speckle patterns corrupted by additive gaussian noise model and with 1.5% imposed strain in X and Y directions, computed via (**a**) optimally sampled original algorithm with 20 % additive noise, (**b**)proposed algorithm with 20% additive noise, (**c**) optimally sampled original algorithm with 40% additive noise, (**d**) proposed algorithm with 40% additive noise, (**e**) optimally sampled original algorithm with 60% additive noise, (**f**) proposed algorithm with 60% additive noise

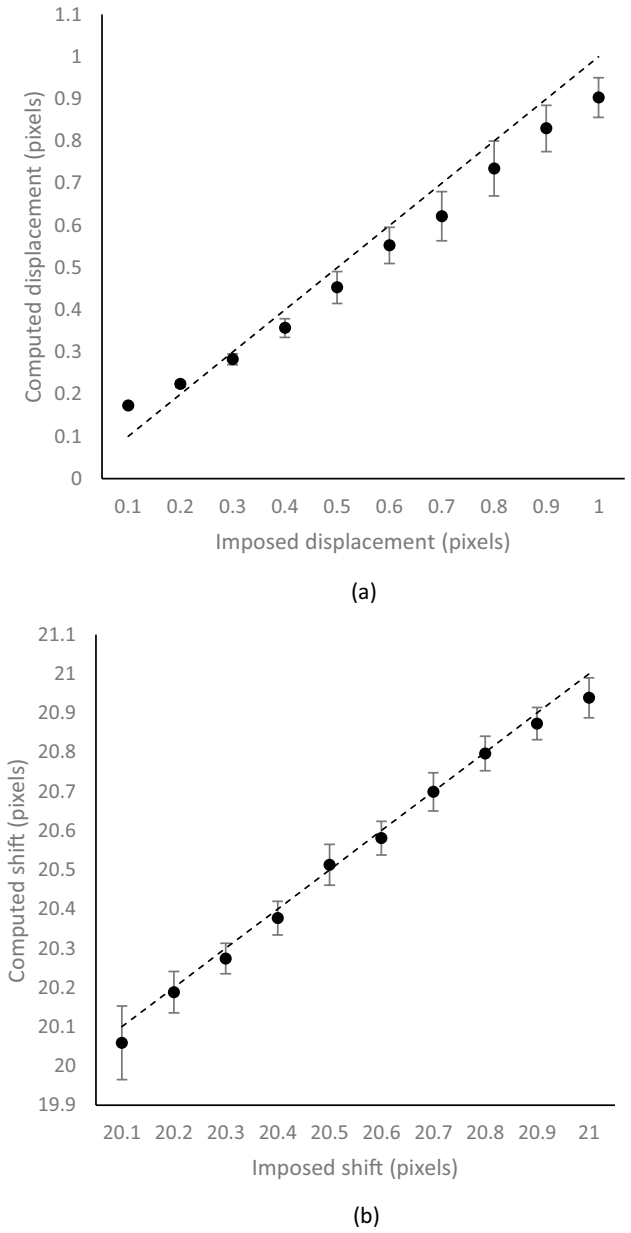


Fig. 11 Subpixel performance for progressively shifted synthetically generated speckle pattern. Dotted line indicates the imposed shift (via bicubic resampling) for shift range of a) 0.1 – 1.0 pixels in increments of 0.1, and b) shift range of 20.1 – 21.0 in increments of 0.1 pixels. Markers indicated mean computed shifts and their standard deviations

size and the hardware platform. In the case of the algorithm presented here, this is further complicated by the conditional branch governing whether or not the aliased shift disambiguation step is invoked (see Fig. 9). Therefore, runtime performance is dependent not only on data size and hardware platform, but also the strain field. To gauge the performance of this implementation, synthetic speckle patterns (average speckle size 4) with dimensions of 1024 × 1024 and 2048 × 2048 were generated as described earlier and 1.5 % strain applied in the X and Y directions via bicubic resampling. This value was chosen to trigger the conditional branch for approximately 75 % of the output data set such that its impact on performance is emphasized. Shift sizes of 8 and 16 pixels were used. The computer testbed consists of an 8-core Intel i9-9980HK with 16GB ram and an AMD Radeon Pro WX8200 discreet GPU. The average execution time for 10 iterations is shown. Runtimes include the peripheral processing required to invoke the GPU kernel; while the core of the algorithm is written in C++ to run on the AMP runtime, the decision was made (primarily for software modernization and ease of consumption) to encapsulate the kernel in a C# class library responsible for invoking it internally via the Platform Invocation services of the .NET runtime. Therefore, all timings include not only the time to execute the GPU kernel, but also the time required to marshal memory to and from both the managed (C#/.NET) to unmanaged (C++) runtimes as well as to-from the GPU global memory bank. Results are shown in Table 1. Even in the case of a 2048 × 2048 image analyzed with a shift of 8 (corresponding to a total output data set of 62,001 analysis points), interactive frame rates are obtained. Greater performance is easily achievable by using a more powerful GPU.

Discussion and Conclusion

Unlike traditional DIC, the method described in this work is a true interferometer and, therefore, limiting requirements are introduced. Primary among these is the need for a dense speckle pattern, most easily generated using a laser light source. Sparse speckle patterns of the type encountered in “white light” speckle or DIC/PIV applications are not expected to perform as well.

Secondly, the method requires accurate modulation of the size of the interference speckles to satisfy the requirements of the sample rate conversion scheme. This is accomplished most easily with a subjective speckle pattern by altering the imaging aperture. While it is possible to apply a white light speckle pattern with the approximate target statistical size distribution, in practice this is would be difficult to achieve reliably. Future work may include an automated analysis method that can be used to determine the average speckle size in an image. This could be used not only to provide feedback to the user when adjusting the lens aperture, but also as an automatic means to optimize the aperture for a motorized lens.

The effect of the sample-rate conversion on subpixel estimation accuracy has not been addressed in this work. The “fine” step of the “coarse-fine” decomposition utilizes a 32 × 32 subimage which produces the best results when applied to an optimally sampled speckle pattern. In this implementation, however, the smaller subimage is being applied to a

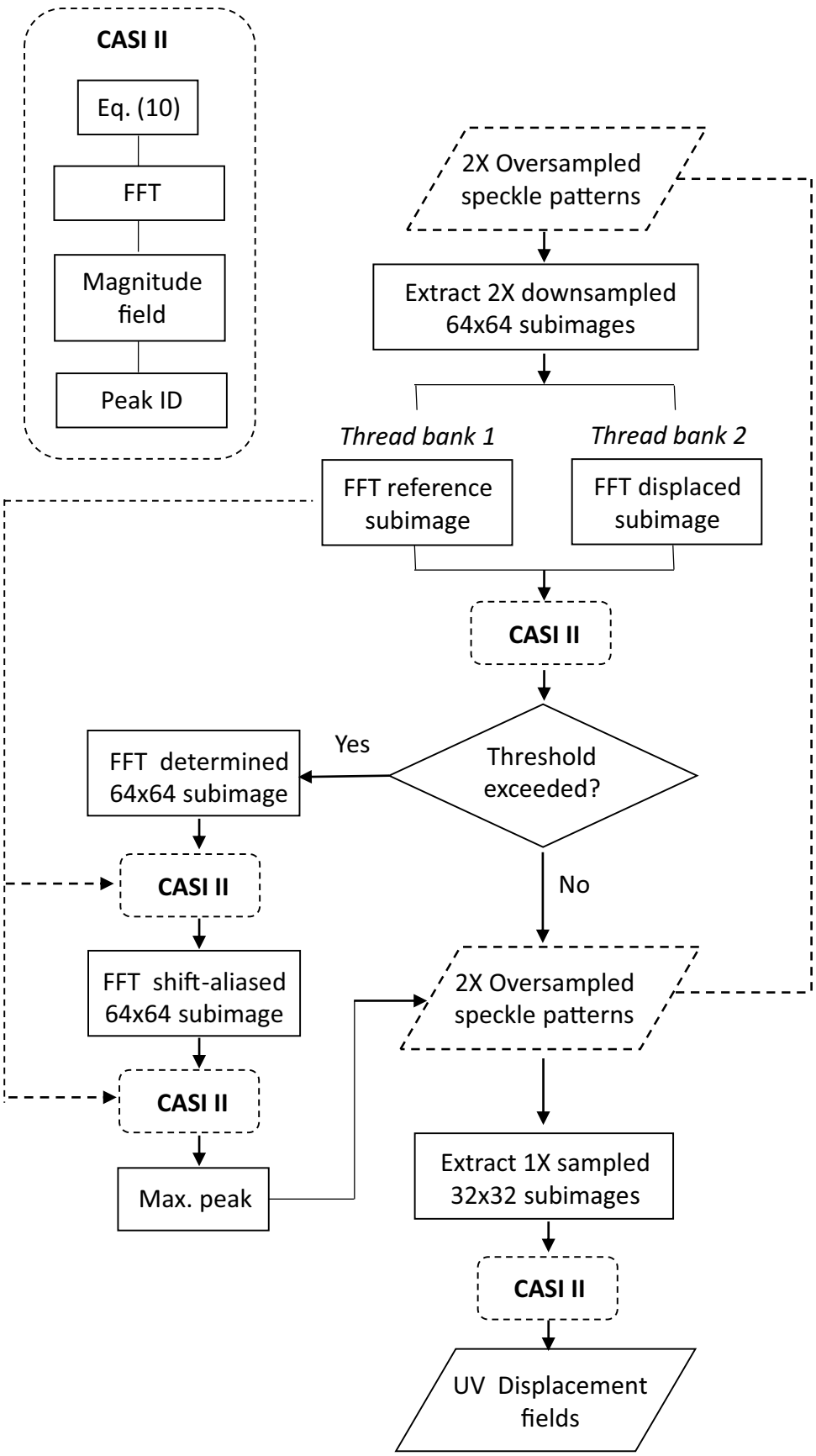


Fig. 12 Proposed algorithm for enhanced CASI II implementation showing processing flow for a given tile partition

Table 1 Runtime performance for GPU-based extended CASI II algorithm

Category	*1024 × 1024*		*2048 × 2048*	
	Shift 8	Shift 16	Shift 8	Shift 16
Full-field analysis points	14,641	3,721	62,001	15,625
Elapsed time	27.2 ms	13.3 ms	102.7 ms	46.6 ms

2× oversampled speckle pattern and some peak broadening can be expected. The effect this has on subpixel estimation has been shown to increase the measurement uncertainty to approximately 10 % of a pixel. While this was determined using resampled images, this places a strong emphasis on choice of type (and quality of implementation) of the resampling method used and should be thought of as, at best, a proxy for ground truth. In addition, noise levels in the image can be seen to affect subpixel estimations. This is clearly evident in Fig. 9, where the displacement contours become visibly progressively less smooth (when compared with the noise-free case shown in Fig. 8) as progressively greater levels of noise are introduced. Given that the implementation proposed here is targeted at real-time applications where noise may be more prevalent due to higher camera frame rates, less emphasis should be placed on the fidelity of subpixel accuracy. Perhaps more importantly, it should be noted that in most applications the true information being sought is strain. This requires a post-processing smoothing/denoising step to avoid overwhelming oscillatory artifacts in the calculated derivatives. Such filters can be expected to heavily alter the subpixel displacement estimations, calling into question the utility of super-fine subpixel detection. Nevertheless, for many applications the approach demonstrated here is likely to be useful and effective.

In conclusion, a series of enhancements to the original CASI II algorithm for simulating speckle interferometry have been detailed, the goal being to amplify both the usable range of the technique as well as its computational performance. Results for representative data sets have shown the efficacy of these modifications. This new version of CASI II generates high-resolution full-field displacement fields at common camera frame rates and is suitable for use in many non-contact industrial inspection applications, as well as classical experimental mechanics studies.

Acknowledgements The author wishes to acknowledge helpful technical discussions with Joseph Goodman (Professor Emeritus, Stanford University) and Fu-Pen Chiang (Distinguished Professor, Stony Brook University). Furthermore, the author would like to thank Joseph Tringe (Lawrence Livermore National Laboratory) for valuable feedback and comments during preparation of this manuscript. This work was supported by the US DOE LLNL-LDRD 20-SI-001 and was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Funding This work was supported by the US DOE LLNL-LDRD 20-SI-001 and was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Data Availability Not applicable.

Code Availability Custom class library implementation available for download.

Declarations

Ethics Approval Not applicable.

Consent to Participate Not applicable.

Conflict of Interest The author declares that he has no conflict of interest.

References

1. Chu TC, Ranson WF, Sutton MA (1985) Applications of digital-image-correlation techniques to experimental mechanics. Exp Mech 25:232–244. https://doi.org/10.1007/BF02325092
2. Shchepinov VP, Pisarev VS (1996) Strain and stress analysis by holographic and speckle interferometry. Meas Sci Technol 7. https://doi.org/10.1088/0957-0233/7/9/019
3. Chen DJ (1993) Computer Aided Speckle Interferometry (casi) and its application to strain analysis. PhD Thesis
4. Scarano F, Riethmuller ML (2000) Advances in iterative multigrid PIV image processing. Exp Fluids 29:S051–S060. https://doi.org/10.1007/s003480070007
5. Westerweel J (1993) Digital particle image velocimetry. Theory and application
6. Scarano F (2001) Iterative image deformation methods in PIV. Meas Sci Technol 13:R1–R19. https://doi.org/10.1088/0957-0233/13/1/201
7. Goodman J (2007) Speckle Phenomena In Optics, 1st edn. Roberts & Company Publishers, Englewood
8. Wang YV, Chen DJ, Chiang FP (1993) Material testing by computer aided speckle interferometry. Exp Tech 17:30–32. https://doi.org/10.1111/j.1747-1567.1993.tb00772.x
9. Rao M, Samuel R, Nair P et al (2001) Applications of holographic and electronic speckle interferometric techniques for NDE of spacecraft components. International Society for Optics and Photonics, Bellingham, pp 560–567

10. Leendertz J (1970) Interferometric displacement measurement on scattering surfaces utilizing speckle effect. J Phys E: Sci Instrum 3:214
11. Adams FD, Maddux GE (1973) On speckle diffraction interferometry for measuring whole field displacements and strains. Air Force Flight Dynamics Lab Wright-Patterson AFB OH
12. Zhu Z, Luo S, Feng Q et al (2020) A hybrid DIC–EFG method for strain field characterization and stress intensity factor evaluation of a fatigue crack. Measurement 154:107498. https://doi.org/10.1016/j.measurement.2020.107498
13. Chen Y, Avitabile P, Page C, Dodson J (2021) A polynomial based dynamic expansion and data consistency assessment and modification for cylindrical shell structures. Mech Syst Signal Process 154:107574. https://doi.org/10.1016/j.ymssp.2020.107574
14. Chen Y, Logan P, Avitabile P, Dodson J (2019) Non-model based expansion from limited points to an augmented set of points using chebyshev polynomials. Exp Tech 43:521–543. https://doi.org/10.1007/s40799-018-00300-0
15. Chen Y, Joffre D, Avitabile P (2018) Underwater dynamic response at limited points expanded to full-field strain response. J Vib Acoust 140. https://doi.org/10.1115/1.4039800
16. Gerber R, Bik JC, Smith A, Tian K X (2005) The software optimization cookbook: High performance recipes for IA-32 Platforms, 2nd edn. Intel Press, Santa Clara
17. Sanders J, Kandrot E (2010) CUDA by example: an introduction to general-purpose GPU programming, 1st edn. Addison-Wesley Professional, Boston
18. Kaeli D, Mistry P, Schaa D, Zhang DP (2015) Heterogeneous Computing with OpenCL 2.0, 3rd edn. Morgan Kaufmann, Burlington
19. Gregory K, Miller A (2012) C++ AMP - Accelerated massive parallelism with Microsoft visual C++, 1st edn. Microsoft Press, Redmond
20. Burch JM, Tokarski JMJ (1968) Production of multiple beam fringes from photographic scatterers. Optica Acta 15:101–111. https://doi.org/10.1080/713818071
21. Archbold E, Ennos AE (1972) Displacement measurement from double-exposure laser photographs. Optica Acta 19:253–271. https://doi.org/10.1080/713818559
22. Archbold E, Burch JM, Ennos AE (1970) Recording of in-plane surface displacement by double-exposure speckle photography. Optica Acta 17:883–898. https://doi.org/10.1080/713818270
23. Khetan RP, Chiang FP (1976) Strain analysis by one-beam laser speckle interferometry 1: Single aperture method. Appl Opt 15:2205. https://doi.org/10.1364/AO.15.002205
24. Chen DJ, Chiang FP (1993) Computer-aided speckle interferometry using spectral amplitude fringes. Appl Opt 32:225–236. https://doi.org/10.1364/AO.32.000225
25. Goodman J (2005) Introduction to fourier optics, 3rd ed. Roberts & Company Publishers, Greenwood Village
26. Chen DJ, Chiang FP, Tan Y, Don HS (1991) Computer-aided speckle interferometry: Part II--an alternative approach using spectral amplitude and phase information. Proc SPIE 1554:706–717
27. Moth D C++ AMP - Introduction to Tiling in C++ AMP. https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/april/c-amp-introduction-to-tiling-in-c-amp. Accessed 20 Mar 2018
28. Cooley JW, Tukey JW (1965) An algorithm for the machine calculation of complex Fourier series. Math Comp 19:297–301. https://doi.org/10.1090/S0025-5718-1965-0178586-1
29. Lyons RG (2004) Understanding digital signal processing, 2nd edn. Prentice Hall PTR, Hoboken
30. Shannon CE (1949) Communication in the presence of noise. Proc IRE 37:10–21. https://doi.org/10.1109/JRPROC.1949.232969
31. Chen DJ, Chiang FP (1992) Optimal sampling and range of measurement in displacement-only laser-speckle correlation. Exp Mech 32:145–153. https://doi.org/10.1007/BF02324726