NASA/TM—2019-220144

Real-Time Background Oriented Schlieren: Catching Up With Knife Edge Schlieren

Mark P. Wernet
Glenn Research Center, Cleveland, Ohio

April 2019

NASA/TM—2019-220144

Real-Time Background Oriented Schlieren: Catching Up With Knife Edge Schlieren

Mark P. Wernet
Glenn Research Center, Cleveland, Ohio

National Aeronautics and
Space Administration

Glenn Research Center
Cleveland, Ohio 44135

April 2019

Acknowledgments

The author would like to than Dr. Randy Locke and Garrett Clayo for their assistance in setting up the RT-BOS hardware in the different NASA Glenn Research Center (GRC) test facilities described in this work. This work was sponsored by the Transformative Tools and Technology Project of the Innovative Measurements Program.

Real-Time Background Oriented Schlieren: Catching Up With Knife Edge Schlieren

Mark P. Wernet
National Aeronautics and Space Administration
Glenn Research Center
Cleveland, Ohio 44135

Abstract

Background Oriented Schlieren (BOS) is a widely used technique that provides density gradient information in flow fields of interest, without imposing stringent optical quality requirements on the facility/experiment windows and/or optics used in the BOS setup. Typically, the BOS reference image is acquired before the test begins (flow off) and then the “live” image data are acquired during the actual testing/experiment (flow on). The raw BOS image data, while displayed in real-time as they are acquired from the camera, unfortunately provide little if any visual indication of the density gradients in the flow. Generally, the “live” images must be processed off-line after the testing is completed, providing no indication of the success of the BOS setup nor any feedback on the operational success of the test.

Advances in computer processing hardware enables the implementation of real-time processing and display of the BOS image data. Two different approaches to implementing the real-time BOS (RT-BOS) processing capability are described herein. First, a traditional multicore Central Processing Unit (CPU) based approach using scheduled parallel threads is used to build a RT-BOS processing engine. In the second approach, a Graphical Processing Unit (GPU) approach is used to construct a RT-BOS processing engine. Generally, high-core count CPU processors can provide a useful processing rate for RT-BOS. However, the GPU based approach exceeds the processing capability of the CPU approach, at a fraction of the cost. The GPU approach places no restrictions on the host PC processing capability, except that it be capable of acquiring the BOS image data from the camera in real-time.

1.0 Background

The Background Oriented Schlieren technique is a widely used tool for studying the density fields in complex flow fields. Typically, the flows are supersonic, where the flow features of interest are the location and shape of shocks in the flow. The BOS technique is similar to classic knife edge Schlieren in that the property of the flow that is measured is the derivative of the density. Classic Schlieren typically requires large diameter mirrors to collimate the illumination light and also the use of high quality optical windows on the walls of the wind tunnel (Ref. 1). However, unlike classic Schlieren, BOS uses a relatively simple optical setup and has much less demanding requirements on the quality of the optical glass along the imaged path (Ref. 2). A second distinction of BOS is that it provides both the x- and y-components of the derivative of the density field whereas classic knife edge Schlieren only provides a single component of the density field rate-of-change. The third distinction is that Classic Schlieren provides real time images of the density gradients in the flow, whereas BOS provides no on-line feedback from the acquired image data, hence the impetus for this work.

The BOS technique employs a digital camera to image a random dot or speckled background pattern. The speckled background must be illuminated in order for the camera to record images of the speckle pattern. Before the flow is turned on, a reference, or “wind-off” image is acquired by the camera. Next, a nonuniform density field is placed in between or the flow is turned on between the camera and the speckled background. The density disturbances along the imaged path yield distortions in the image of the speckled background. The acquired distorted images of the speckle pattern are then cross-correlation processed against the reference image using standard Particle Image Velocimetry (PIV) processing techniques to measure the density gradient in both the x- and y-directions. Improvements in the BOS processing have continued and the current state-of-the art uses a Least-Squares Matching (LSM) algorithm to measure the displacements (distortion) between the reference image and the "wind on" images (Ref. 3). The LSM approach can sometimes yield higher sensitivity than the cross-correlation approach, but is computationally much less efficient.

BOS has been used in both enclosed wind-tunnel flows and in external flows utilizing full scale rotorcraft and aircraft. In each of these applications, innovative backgrounds are used, such as leafy trees or the desert floor or even the surface of the Sun (Refs. 4 and 5). The main requirement is that the reference or speckled background remains unchanged during the duration of the measurement. For most indoor/wind tunnel applications of BOS, manmade speckle backgrounds are typically employed. Usually a white speckle pattern is sprayed on a dark background. The size and density of the speckle pattern affects the sensitivity of the BOS technique. The optimal size of the imaged "speckle" onto the CCD sensor should nominally span 2 to 3 pixels (Ref. 2). Hence a wide range of speckle patterns must typically be generated for each new experiment/field-of-view. Additionally, these speckle patterns must be sufficiently illuminated so that the camera can detect the speckle pattern with high contrast. Obtaining uniform illumination across the speckled background target and avoiding glare spots is a challenging task. Limited optical access ports in wind-tunnel facilities exacerbates the issue. In this work an innovative approach for generating the background speckle patterns was employed (Ref. 6). The speckle patterns required for the BOS measurement were displayed on a high definition 4K resolution monitor.

Real-time PIV acquisition and processing systems have been built previously using Field Programmable Gate Array (FPGA) technology (Refs. 7 to 9). Some of these systems were commercial products with very low frame-rate processing capability on relatively low-resolution images, however; they did open the door for real-time processing of PIV image data. FPGAs offer very fast signal processing capabilities, which continue to improve. Programming FPGAs requires extensive knowledge and experience with both the details of the hardware and the VHSIC Hardware Description Language (VHDL) used to program these devices. The main drawback of FPGA based signal processing is the lack of integration between the hardware and software, requiring an expert or team of experts to implement a processing solution.

Graphical Processing Units (GPU) are the main processing engine used in high end computer graphics boards providing realistic rendering and physics modelling in computer games. GPUs can also be used for general purpose computing. Several researchers have used GPUs to speedup data processing in PIV data reduction systems (Refs. 10 and 11). These previous GPU efforts were focused only on processing large volumes of PIV data off-line after the testing was completed. The GPU was used as a super-processing engine to expediently process large volumes of PIV image data. The software interface for programming GPUs is extremely well integrated and user friendly compared to the software tools available for FPGAs. The simplicity of the GPU interface made this project possible with only a novice level of experience in GPU programing. The real-time processing approach described in the work presented herein for BOS is equally applicable to PIV image data. Hence, although real-time BOS image processing is discussed, a real-time PIV image acquisition and processing package has also been developed in parallel. The BOS version is slightly simpler and faster, since there is only one reference image, whereas in PIV we are always acquiring a new pair of images for each data processing operation. The RT-PIV approach provides on-line monitoring of the seeding level concentration and the quality of the cross-correlated velocity vector maps.

2.0 Generalized Cross-Correlation Description of Serial BOS Processing

A generalized cross-correlation (GCC) processor is illustrated in Figure 1. There are two 2D spatial inputs to the GCC processor, identified as $s_1(x, y)$ and $s_2(x, y)$, which are each Fourier transformed. The lower input path also contains the complex conjugation operator, following the Fourier transform operator. The two Fourier transformed functions are multiplied together and then inverse Fourier transformed. The result is multiplied by its complex conjugate, yielding the real-valued cross-correlation of the two input signals. A peak detection operation is performed to determine the correlation peak location. The GCC processor just described and illustrated in Figure 1 is the standard data processing technique used to reduce PIV or BOS image data. In BOS, the reference image of the background speckle pattern with the "flow off" is recorded before the experiment/test begins. During the test, with the flow turned on, additional images of the background speckle pattern are acquired, which have been distorted by the density gradients in the flow. In the standard BOS cross-correlation process, the objective is to measure the distortion of the speckle pattern between the "wind on" and "wind off" conditions. The recorded image frames are segmented into small interrogation subregions, where $sr_1(x, y)$ is a subregion from the image 1 (reference image) and $sr_2(x, y)$ is a subregion from image 2 recorded with the flow on, as shown in Figure 2. In BOS processing, the interrogation subregions from the reference (wind off) and live images are the inputs to the GCC. The location of the correlation peak on the output plane identifies x- and y-components of the density gradients in the flow across the correlation subregion. This process is traditionally carried out in a serial manner using a for/do loop to iterate through all of the subregions across the acquired image pair in order to provide a mapping of the density gradients across the complete imaged flow field. The BOS processing yields the individual x- and y-components of the density gradient, enabling the user to select which component of the density gradient, or its magnitude is displayed.

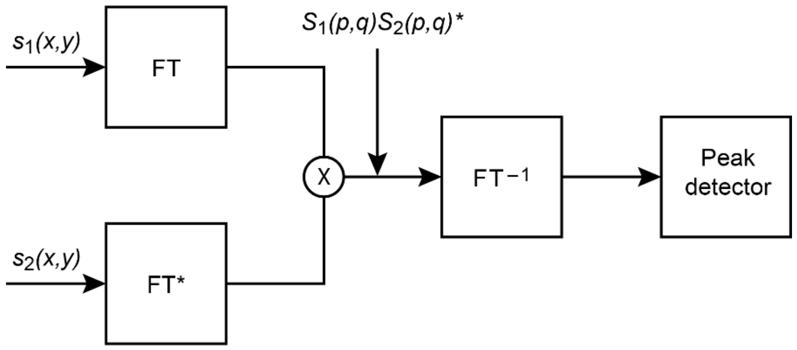


Figure 1.—Generalized Cross-Correlation processor flow diagram. Two input functions are Fourier transformed, multiplied, and then inverse Fourier transformed. A peak detection operation is then performed to locate the correlation peak.

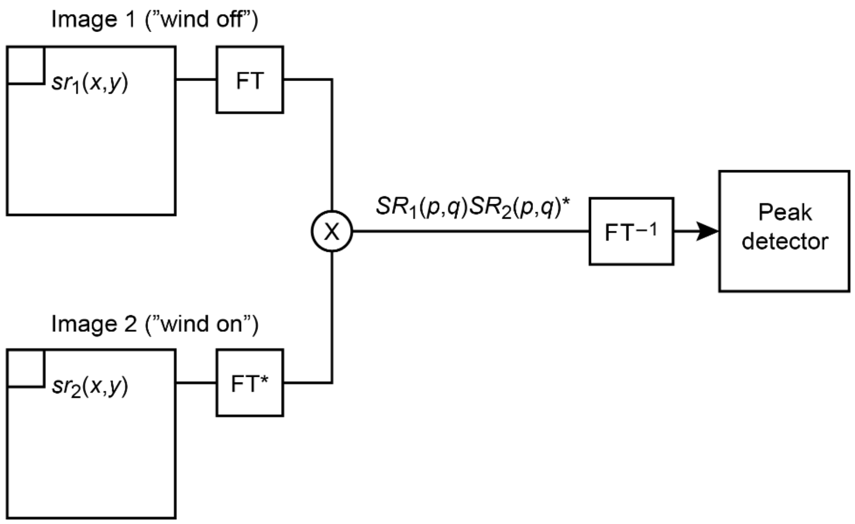


Figure 2.—BOS processing of an image pair, where each image is divided into subregions (small rectangle in the upper left corner of each image), which are cross-correlation processed. The location of the correlation peak from each cross-correlation operation yields the density gradient estimate for that subregion.

3.0 Parallel Processing Cross-Correlation Engines

Serial processing is the traditional approach for data processing on computer systems. The availability of multicore processors and the multithreaded software to make use of these processors opens up new possibilities in computationally intensive data processing operations. The ease in adopting these new technologies is directly proportional to both the benefits they provide and the efficiency of the tools available to the developer in order to implement the processing strategies on the hardware. While the CPU approach is more straightforward, real-time performance on a Microsoft Windows based operating system is difficult/impossible to achieve. For a modest increase in complexity, GPUs offer a processing platform independent of the host PC, where real-time processing can be achieved.

3.1 Intel Based Multithreaded Processing

In order to improve the processing efficiency of the standard BOS processing a parallel processing approach must be implemented. The objective is to use a multicore general-purpose processor and achieve a throughput improvement in proportion to the number of available processor cores. Modern compilers offer the option of automatic parallelization of the “For Loops” in the program code. Typically, these automatic parallelizations do not make efficient use of the available PC cores, sometimes resulting in inefficiencies of over 50%, meaning that less than half of the processing capability of the CPU is being realized. The loss of efficiency is due to the overhead of managing the pool of threads. Instead, the code is manually parallelized, where pre-knowledge of the geometry of the problem is incorporated into the launching of the parallel threads. By assigning each thread a substantial subsection of the processing job, the overhead of managing the threads is minimized. The multithreaded CPU based program is written in C++ and developed in the Microsoft Visual Studio 10. The CPU based processing version of the cross-correlation engine implemented here uses the Intel Math Kernel (MKL) Libraries’ optimized FFT routines which improves the processing speed by a factor of 3 over a standard Cooley-Tukey FFT function implemented in C or FORTRAN (Ref. 12).

The first step is to generate the processing grid of subregions for the BOS image using the user selected subregion size and spacing. The image is then divided into N_T horizontally banded regions, where N_T is equal to the number of available cores in the PC. The program manually launches N_T threads and then waits for all of the threads to complete their processing jobs. Each thread operates independently of all other threads, computing the cross-correlation and peak detection operations for each subregion within its “band” of the image, as depicted in Figure 3. The slowest thread ultimately limits the total throughput of the system. A best practice is to not use all of the available cores in the system, since a few must be left available for operating system maintenance chores. Hyperthreading is a feature of Intel processors, which effectively doubles the number of cores available for processing. For the RT-BOS testing in this work, Hyperthreading is enabled in BIOS and then N_T is limited to the actual physical number of cores in the system. Manually distributing the jobs reduces the overhead of managing the threads since the processing order is predetermined and avoids memory conflicts since the threads are working on different regions of the image. Using this approach, the increase in processing throughput can be nearly equal to the total number of cores allocated for the processing, N_T. If Hyperthreading is not enabled, then the number of cores allocated for processing must be reduced to less than the number of physical cores in the PC, in order to allow system maintenance operations. Unfortunately, the operating system is still in charge of scheduling the independently operating threads through the CPU execution core.

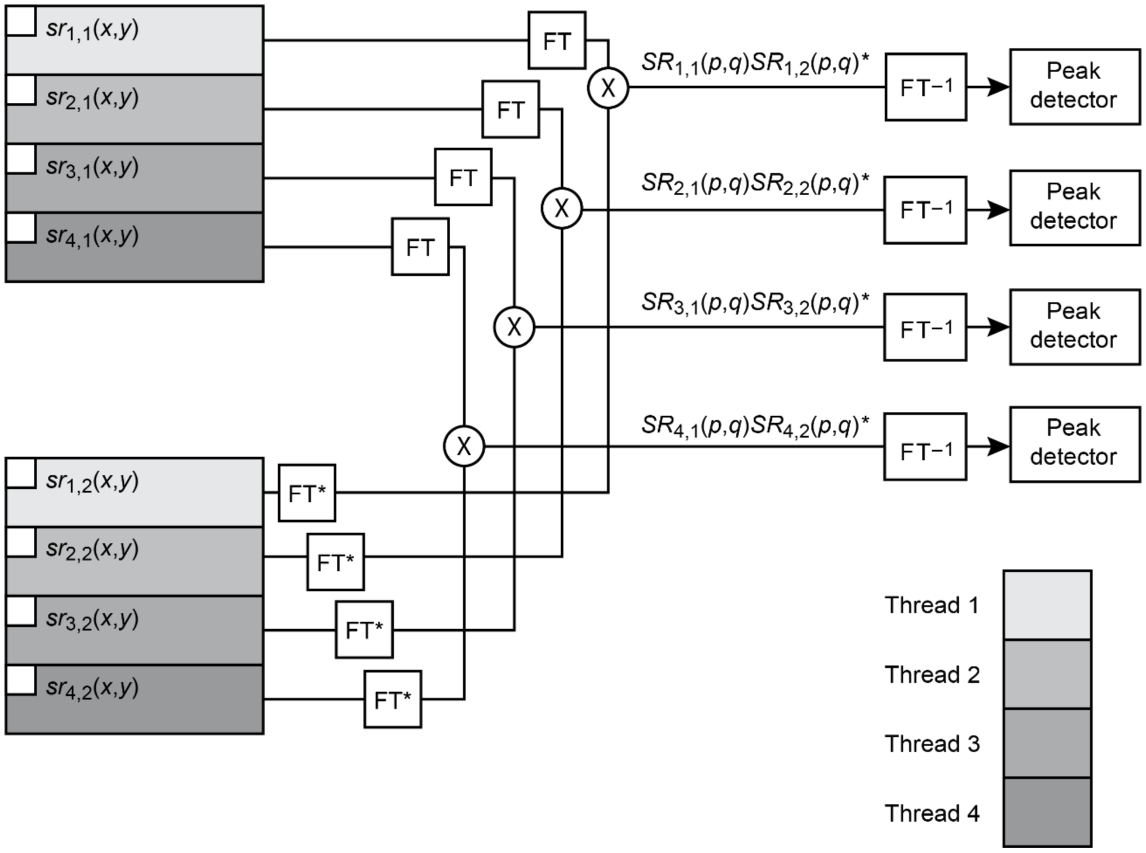


Figure 3.—Multicore PC processing based parallel processing implementation of the BOS cross-correlation processing engine. Here four threads are shown working on four different, independent regions of the input BOS images.

3.2 GPU Based Processing

Graphical Processing Units (GPUs) are the heart of all of the high performance video cards for computer gaming and high-end Computer Aided Design (CAD) engineering workstations. GPUs have continued to mature in their computational capability in order to provide more realistic physics based modeling (fire, smoke, and liquids, etc.) in high end gaming and virtual reality games. The high core count, coupled with high-speed memory, makes GPUs very powerful at certain types of engineering problems. The problems of interest achieve maximum performance improvement on GPUs when they can be computed in parallel and the computations are independent from each other. There is typically no communication between the threads that are launched on a GPU, each one works on its own small piece of the overall job. The approach to solving parallel problems on the GPU is very different from traditional programming languages, where the execution of a program was less physically intertwined with the hardware attributes of the computing platform. The Compute Unified Device Architecture (CUDA) is the interface that gives C, C++ and Fortran programmers access to the function level programming on the GPU. Writing efficient code to make full use of all of the GPU's capabilities requires extensive knowledge of the hardware and software resources available on the platform. Fortunately, the hardware vendors producing the GPU graphics boards have also built highly optimized software libraries to make efficient use of the GPU resources for less experienced CUDA programmers. Even with these highly optimized libraries, the user must still frequently write low-level CUDA code in order to achieve maximum performance for the particular programming problem of interest.

The implementation of the BOS cross-correlation processing engine on a GPU platform was built using Microsoft Visual Studio 10 with nVidia's Nsight integration into Visual Studio. The Nsight cross compiler takes the compiled C code and then integrates the nVidia CUDA function calls into the final executable code. The integration is generally seamless to the user once all of the nVidia library definitions are incorporated into the project build file. The nVidia platform was selected based on the wide availability of nVidia hardware and the wide range of nVidia CUDA libraries. The main library used to implement the cross-correlation engine is the cuFFT Library, which contains nVidia's highly optimized, proprietary FFT routines. Since FFTs are the most computationally intensive operation in implementing a cross-correlation engine, the cuFFT routines are a key element in efficiently implementing the cross-correlation engine on the GPU platform. Using the Nsight cross-compiler, the executable code can be generated to perform across a broad class of nVidia processing chip generations: Kepler, Maxwell, Pascal, and Turing.

GPUs operate independently from the host PC. Optimal processing throughput on the GPU is achieved when the amount of data that must be transferred between the GPU and the host PC is minimized. Hence, it is desirable to complete all of the processing on the GPU. GPU based processing is most efficient when a single function call (kernel launch on the GPU) can complete all of the processing required for a particular step in the processing. The cuFFT Library FFT function has a batch mode of operation, enabling it to process a series of FFTs in a single function call. The FFT function can perform 2D FFTs on a 2D array stored as a continuous block of data in the GPU memory. The stride variable then defines the step size or size of the 2D subregions to be processed. Hence, by rearranging the input image data into the proper format, the cuFFT Library routine can compute all of the FFTs of all of the subregions in an image in a single CUDA function call. For the tests performed here, single precision complex to complex FFTs were computed. The cuFFT Library routine was implemented to make optimal use of the compute capabilities of the installed nVidia board in the host PC. A flow diagram of the GPU processing engine is detailed in the in Figure 4. Briefly, the BOS images to be processed are uploaded to the GPU from the host as 8 bit/pixel images, the most compact format possible. The BOS images are then rearranged into a complex 1D array where each 2D subregion to be processed in the image is arranged as continuous complex elements in a linear array with their imaginary elements set to zero. For example, if the $N \times N$ subregion size is 32×32 pixels, then the 32×32 pixel region of the image is reordered as 1024 contiguous complex elements in the linear array. The next subregion to be processed occupies the next 1024 elements, and so on. The cuFFT function performs an in-place FFT so the data are output in the same array as the input, hence the need for the complex input format. The cuFFT routine is called for each of the images to be cross-correlated. The output complex vectors are then multiplied together element-wise in a custom CUDA routine in a single call. Then cuFFT Library FFT routine is called again to perform the inverse FFT, completing the cross-correlation operation. The output correlation plane only occupies the central $N/2 \times N/2$ region of the output plane. Another custom CUDA routine extracts the central $N/2 \times N/2$ region of each correlation plane, computes the complex magnitude and puts the result into a new linear array, which is 1/4 the size of the original array. Next, a custom CUDA routine finds the max element on each correlation plane. Another custom function call uses the max element locations to perform a 3×3 bilinear interpolation to estimate the correlation peak location to sub-pixel resolution. The correlation peak locations are the x- and y-density gradients in the fluid and are the final result from the processing. Only the x- and y-displacement real values need to be transferred back to the host PC.

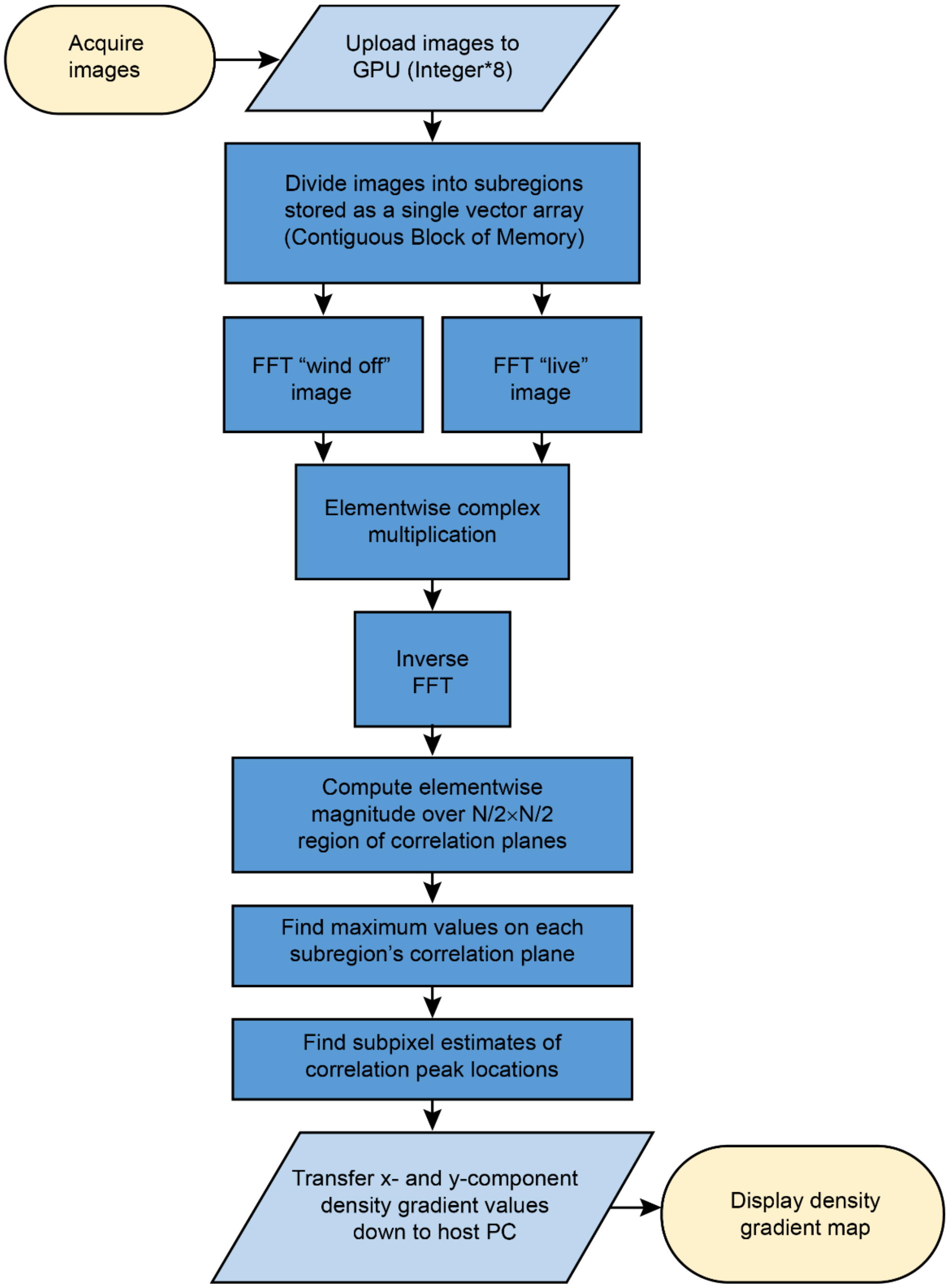

Acquire images
Upload images to GPU (Integer*8)
Divide images into subregions stored as a single vector array (Contiguous Block of Memory)
FFT “wind off” image
FFT “live” image
Elementwise complex multiplication
Inverse FFT
Compute elementwise magnitude over N/2×N/2 region of correlation planes
Find maximum values on each subregion’s correlation plane
Find subpixel estimates of correlation peak locations
Transfer x- and y-component density gradient values down to host PC
Display density gradient map

Figure 4.—Flow diagram of GPU cross-correlation engine processing steps.

3.3 Benchmarking: Intel Multicore Xeon Processor vs. nVidia GPUs

A program was written to benchmark the BOS cross-correlation engine on both the host PC and the installed GPU board with a single invocation. The evaluation images used were 8 bit/pixel, 4008×2672 pixel (11 MPixel) images. The nVidia CUDA 9.1 Tookit was used with an RTX-2080 ti board for these benchmarking tests. The host PC is an Intel XEON E5-2699-v4 based system running Microsoft Windows 7, 64 bit. The CUDA evaluation program calls three routines TEST1(), TEST2() and TEST3(), each of which processes the same pair of PIV images using the same processing parameters (subregion size and % overlap). TEST1 performs the processing entirely on the host PC, using the Intel Math Kernel Library (MKL) FFT routines for most efficient processing. The nominal test case uses 64×64 pixel subregions on a 16×16 pixel grid (75% overlap), yielding 40,098 subregions to be processed. TEST1 takes the input images and reorganizes them into a linear vector array just as is done on the GPU. The routine loops through all of the subregions in the linear array – the processing approach is as close to what the GPU processing is doing – except that the FFTs are being done serially, not in parallel as they are on the GPU. TEST1 provides a baseline BOS image pair processing time on a standard PC, without taking advantage of the multicore processing capabilities of the host PC. TEST2 is a parallel thread implementation of the BOS image processing on the host PC as described above. TEST2 provides a direct comparison of a parallel CPU implementation with the GPU processing engine. TEST3 loads the image files up to the GPU and then generates the linear arrays for feeding the data into the cuFFT routines. The time to transfer the images up to the GPU board is included in the overall GPU processing time. TEST3 represents the ultimate GPU processing strategy where all of the processing done on the GPU. Only the processed x- and y-density gradient values for each subregion are transferred back down to the host PC. For any of the processing modes, the resulting 2-component density gradient maps are written to files on the host PC to verify that the routines worked properly and yielded the same results.

Initially, the linear array of subregions was generated on the PC and then uploaded to the GPU, which facilitated the quick implementation of the algorithm on the GPU. However, the linear complex arrays were much larger than the original image files due to the complex data types and the redundancy of the overlapping subregions (4 Bytes/pixel*2 complex*64×64*40098 subregions = 1.25 GBytes/image), hence the transfer up to the GPU was slow. It was faster to upload the single byte integer image data up to the GPU (11 MBytes/image) and then generate the linear vector of complex subregions on the GPU using another custom CUDA kernel. As shown above, a 4008×2672 image processed using 64×64 pixel subregions at 75% overlap yields 40,098 subregions, requiring 1.25 GBytes/image of storage per image. There are two input images and hence two linear arrays to pass to the cuFFT Library routine. The BOS image data are processed using in-place FFTs so no additional large arrays are required. The size of the image that can be processed and the fidelity of the processing grid is limited by the available memory on the GPU. The 11 MPixel images used here require approximately 3.2 GBytes of GPU RAM storage space. GPUs with 8 GBytes installed are relatively inexpensive; however, models with 24 GBytes are available for much higher costs.

The results from the benchmarking are shown in Table 1. The baseline case, which is the traditional single threaded approach for processing the BOS image data requires 1904 ms to complete. Implementing a multithreaded version of the code yields over an order of magnitude reduction in the processing time: 149 ms. The reduction in processing time is not directly linear with the number of threads used to process the image. The realized reduction of nearly 13x in the processing time using 22 threads is disappointing. In a perfect implementation, the expected processing time would be reduced by 1/22, or 86 ms. The benchmarking program is capable of timing each individual thread on the host PC during the program execution. Some of the threads in the multithreaded CPU code finished in 74 ms, even better than expected based on the breakdown of the job by core count. Other threads required twice that amount of time, which leads to the overall processing time of 150 ms for the multithreaded CPU approach. Remember, a CPU is a generalized processor, controlled by the Windows operating system. The operating system places additional restrictions/requests on the processor during the time of program execution, which results in some of the threads in the program not getting equal access to the CPU, and hence an overall slower processing time. It appears that the operating system is still waiting to schedule some of the threads only after some of the initial threads are completed. The assumption that as long as the number of executing threads was less than the total number of available cores was not realized. Attempts to reduce the number of threads increased the workload on each thread and did not offer any performance improvement. For lower core count CPUs (core-i7 CPU family), the throughput increase was closer to the actual number of threads used. However, since the number of available cores was lower the effective system throughput was not increased over the Xeon based system implementation. Attempts to improve utilization by setting thread priority levels was ineffective. The TEST3 routine is fully implemented on the GPU. The processing time is 49 ms, a factor of 38 better than the single-threaded approach. The GPU is a dedicated processing engine, yielding very consistent processing times. The multithreaded CPU code yielded a range of performance values depending on the other system maintenance operations requested by the operating system. Note that the CPU based processing made use of the Intel MKL FFT library functions, which are 3x than a standard FFT algorithm. Compared to a standard PC using standard FFT algorithms, the GPU outperforms the PC by a factor of 114x.

The Performance Analysis section of the nVidia Nsight developer studio enables profiling the code to see how long each kernel launch requires to finish processing. For the RTX-2080 ti board, the benchmark test case profile is listed in Table 2. The breakdown of the processing time for the RTX-2080 ti benchmark shows that the majority of the time is spent computing the FFTs, as would be expected. Each FFT kernel launch requires 10.9 ms. The only two other significant processing steps are Step 2: the reorganization of the image data and Step 5: Complex elementwise multiplication. Both of these routines were written by the author. There are most certainly additional optimizations making better use of the nVidia hardware that could be done to the code by a more experienced developer. These additional enhancements would only result in a few millisecond reduction in the processing time of < 10%. However, the code developed here certainly approaches the best that could be expected for a GPU based cross-correlation engine. Although the nVidia GPU supports concurrent processing, attempts to implement concurrent processing were only effective for very simple CUDA kernels, which did not benefit the cross-correlation processing engine implementation in this work.

TABLE 1.—RESULTS FROM BENCHMARKING THE CROSS-CORRELATION ENGINE

Test	Subregion size on 16×16 grid	Number of subregions	Processing platform	Number of threads	Total processing time, ms
Test1[a]	64×64	40,098	PC	1	1904
Test2[b]	64×64	40,098	PC	22	149
Test3[c]	64×64	40,098	GPU	4352	49

[a]Single threaded CPU function.
[b]Multithreaded CPU function.
[c]Multithreaded GPU function.

TABLE 2.—TEST3 PROCESSING TIME FOR EACH STEP ON THE GPU

1) Transfer images up to GPU	1.4 ms
2) Convert images into LinVec of subregions	5.6 ms
3) Batch FFTs image 1	10.9 ms
4) Batch FFTs image 2	10.9 ms
5) Complex multiply	8.3 ms
6) Inverse batch FFT	10.9 ms
7) Complex multiply, N/2×N/2 reduce	1.15 ms
8) Correlation plane find max	1.3 ms
9) Compute 3×3 peak estimate	0.12 ms
10) Copy result to host	0.02 ms
Total:	49.17 ms

There are a wide array of GPU boards available on the market. Determining the GPU board features critical for implementing an efficient cross-correlation processing engine is an additional objective of this work. Additional benchmarking tests were performed across a series of nVidia GPU boards to determine which GPU board features affect the overall cross-correlation engine system throughput, as listed in Table 3. The same code used to perform the CPU versus GPU benchmarks above was again used on different nVidia GPU boards spanning the Kepler, Maxwell, Pascal, and Turing architectures. Many of the computing problems implemented on GPUs are performance limited by memory bandwidth, since accessing the data is the limiting step for all of the concurrent threads attempting to process the large data sets. There are thousands of threads available on the GPU to perform the data computations in parallel, however, accessing the data in the GPU memory in order to perform the computations is the fundamental limitation of these systems. Table 3 shows a general increase in memory speed and number of available cores from left to right. The performance analysis shows that less than 10% of the GPU resources are being utilized, hence many threads are left with nothing to do. Hence the processing throughput was not strongly dependent on the number of available cores. There is a strong correlation between memory bandwidth and processing time, as shown in Figure 5. A linear fit through the data illustrates the trend. An additional factor influencing the results is the Graphics Double Data Rate (GDDR) synchronous dynamic random-access memory architecture used on these differing generations of GPU boards: GDDR5, GDDR5X, and GDDR6. While there are many additional computational/functional improvements to the GPU capabilities with each new chip/board generation, our cross-correlation engine is not making use of these new features. These new enhancements certainly benefit graphical based computer games, which are the target for these new compute capabilities. Each new generation of GPU board does increase the memory bandwidth, which yields noticeable improvements in the overall cross-correlation engine performance.

As a side note, an attempt was made to use the Intel series of PHI processor boards in the development of a BOS cross-correlation engine, assuming that the PHI based approach would be a more natural extension of our traditional CPU based processing. However, the PHI boards require a Linix based operating system and are actually designed on the Message Passing Interface (MPI) Cluster style approach to High Performance Computing (HPC) problems. Unfortunately it was not feasible to devote the time and resources to implement the cross-correlation engine on the PHI boards. The nVidia Nsight approach provided a more direct and more user friendly approach for implementing the parallel processing approach.

TABLE 3.—COMPARISON OF GPU PROCESSING TIMES FOR VARIOUS nVidia GPU BOARDS
[The XEON CPU result is included for reference.]

	GTX-980 ti	GTX-1080	Quadro M6000	GTX-1080 ti	RTX-2080 ti	Xeon 2699-v4
Compute capability	5.2	6.1	5.2	6.1	7.5	--
Memory, GB	6	8	24	11	11	--
Memory type	GDDR5	GDDR5X	GDDR5	GDDR5X	GDDR6	--
Mem bandwidth, GB/s	336.5	320	317	484	616	--
Number of cores	2816	2560	3072	3584	4352	44
Base frequency, MHz	1000	1607	1506	1480	1350	--
Architecture	Maxwell	Pascal	Maxwell	Pascal	Turing	--
Processing time, ms	98.3	103.2	93.8	69	49.17	149

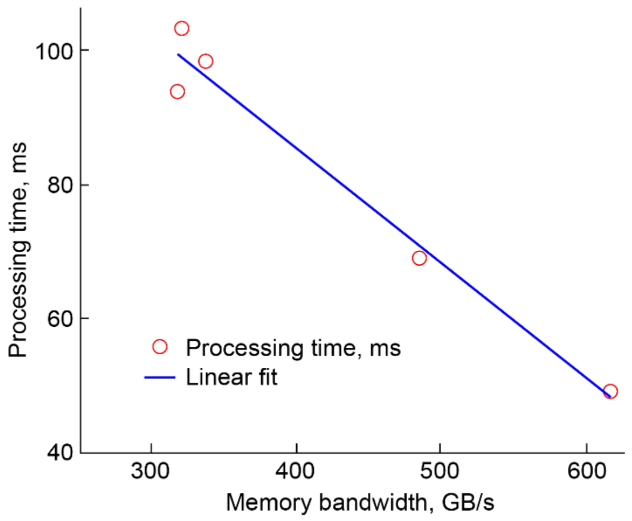


Figure 5.—GPU memory bandwidth versus processing time for the results from Table 3.

4.0 Real-Time BOS System

4.1 Self-Illuminating Background

A critical element in a BOS experiment is the background speckle pattern. Usually, a white speckle pattern is sprayed on a dark background. The size, distribution and contrast of the speckle pattern determines the BOS measurement system sensitivity to gradients in the flow. Speckle backgrounds must be generated for each new BOS implementation. Additionally, these speckle patterns must be sufficiently illuminated so that the camera can detect the speckle pattern with high contrast. Obtaining uniform illumination across the speckled background target and avoiding glare spots from the illumination source is a challenging task. Limited optical access ports in wind tunnel facilities can exacerbate the issue. Polarizers can be placed on the illumination sources and cross-polarizers on the camera lenses to reduce the glare from the surface, with a concomitant 25% reduction in the signal levels.

In this work, an innovative approach for generating the background speckle patterns was employed. The speckle patterns required for the BOS measurement were displayed on a high definition 4K resolution computer monitor (Ref. 6). Use of the 4K HD monitor to display the speckle patterns has four distinct advantages: (1) the speckle patterns can be generated on the computer and displayed directly on the monitor without having to physically construct the speckle pattern; (2) the scale of the speckle pattern can be readily changed to optimize the BOS system performance; (3) the speckle pattern is self-illuminating, which greatly simplifies implementing the technique in confined environments; and (4) the self-illuminated background enables the use of high f/number settings on the camera lens. Increasing the f/number on the camera lens minimizes the geometric blur in the acquired BOS image data (Ref. 2). Typically, operating a BOS system at high f/number is a challenge since it depends on being able to brightly illuminate the background. The self-illuminated background provides sufficient light so that the camera lens can be set to a high f/number, thereby enabling a more optimal configuration of the optical system.

A 4K monitor is used so that the highest resolution speckle pattern can be generated. The actual speckle pattern image to be displayed on the monitor must also be true 4K resolution (3840×2160 pixels) in order to achieve the highest resolution, highest contrast image. Background speckle pattern image files are readily generated using any of the publicly available PIV image simulation generating codes, where the size and concentration of particles determines the resulting speckle pattern size. Standard LCD and OLED type monitors have been used for generating the backgrounds. The best monitors have a rigid frame fully supporting the screen, which also facilitates mounting the monitor in a rigid frame to keep the display surface of the monitor from flexing or distorting. Although the heat generated by the monitor is not a significant issue, the use of a small fan to blow a steady stream of air across the surface of the monitor helps to cool the monitor and decorrelate any residual density fluctuations during the BOS image acquisitions.

4.2 Real-Time BOS Program

The Benchmarking comparison verified that the GPU approach to implementing the cross-correlation engine would provide performance benefits over traditional CPU based processing approaches. The program, referred to as RT_BOS_Cuda actually has both the multithreaded Intel processing code and the GPU based processing engine built into it. At program compilation the user can select which processing platform will be built in the executable code.

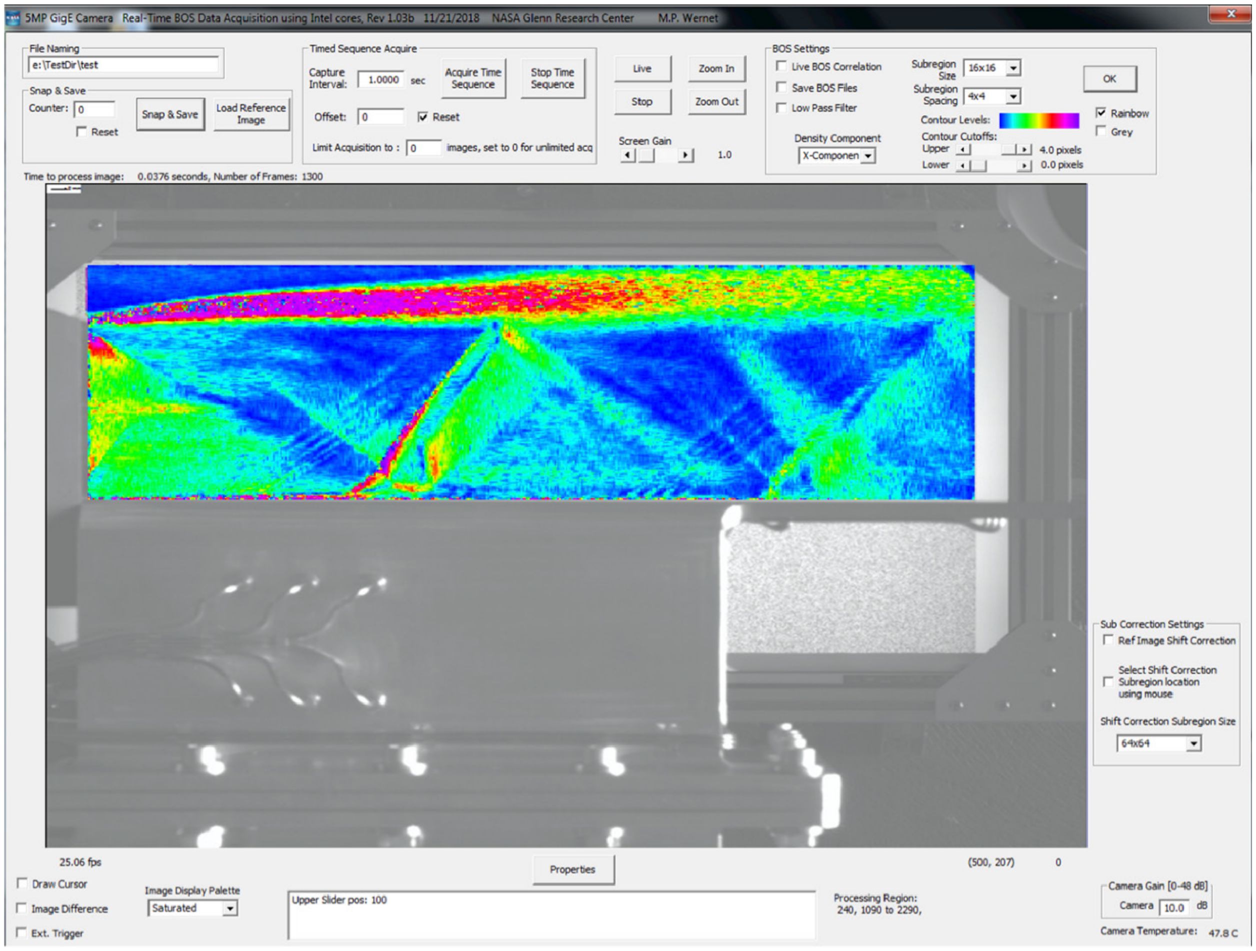
RT_BOS_Cuda can support either CameraLink or GigE based cameras. CameraLink support is provided through an EPIX framegrabber and XCLIB library of functions (Ref. 13). Support for the GigE cameras is obtained via the ActiveGigE SDK from A-B Software, which provides a very user friendly development path for supporting GigE cameras that comply with the GenIcam format (Ref. 14). GigE cameras are desirable for BOS due to their relatively small physical size, their high pixel count sensors, which operate at high frame rates and their ability to operate via Power of Ethernet (PoE) requiring a single interface cable for both power, control and data transfer. All of the camera operating features and settings are available via the software interface and the real-time acquisition and display of the camera images facilitates implementation of the Real-Time BOS system. A variety of GigE cameras have been used from FLIR and Pro Silica.

Once the BOS image data are acquired, they are processed following the flow chart of the cross-correlation engine described previously in Figure 4. RT_BOS_Cuda lets the user acquire a reference image before the testing begins. Then the system can go “live” and cross-correlate the reference image with all of the new incoming images from the camera. One simplification is possible for RT_BOS_Cuda compared to a real-time PIV implementation for the cross-correlation engine described above. Since there is only one reference image, the FFT of the reference image is computed and stored in GPU memory. Hence, only the incoming “live” image data need be FFT’d, reducing the computational load of the RT-BOS system.

RT_BOS_Cuda enables the user to select the subregion size and the spacing for the processing of the incoming images. The range of the density gradients that are measured scales with the subregion size. Smaller subregions are scaled to cover a smaller range of density gradients, which essentially scales the colormap across a smaller range, making the system appear to be more sensitive to smaller density gradients. The scale of the color maps can also be scaled to enhance the range of the density gradients that are being processed. The user also has control over the region of interest of the image that is processed. In some cases, the speckle background pattern may not fill the camera field of view, hence limiting the processing region reduces extraneous noise in the processed density gradient maps.

The goal of the RT_BOS_Cuda program is to acquire, process and display BOS image data in real time. Hence, the last step is the expedient display of the processed density gradient map on the user dialog. As the subregions are processed, the value of the density gradient determined for each subregion is written into a block of memory, which has the same dimensions as the display area on the RT_BOS_Cuda image display region. When in live correlation mode, the image data are overwritten with the processed density gradient maps. The fastest method for writing the image to the display is to use a BitBlt Windows API function call, where the block of memory holding the processed results is transferred to the computer graphical display buffer in the single API call.

All of the user controls for RT_BOS_Cuda are on the main dialog screen. The reference or live images are displayed in the main image display region. When live-correlation mode is selected, then the acquired images are overwritten by the processed density gradient color contour maps. Both color contour maps and gray scale contour maps of the density gradients are available. A screenshot of the RT_BOS_Cuda user screen is shown in Figure 6, where a processed image from an experiment where a



5MP GigE Camera   Real-Time BOS Data Acquisition using Intel cores, Rev 1.03b   11/21/2018   NASA Glenn Research Center   M.P. Wernet

File Naming: e:\TestDir\test

Counter: 0 ☐ Reset

Capture Interval: 1.0000 sec

Offset: 0 ☑ Reset

Limit Acquisition to : 0 images, set to 0 for unlimited acq

Screen Gain 1.0

Subregion Size 16x16

Subregion Spacing 4x4

Density Component X-Componen

Contour Cutoffs: Upper 4.0 pixels, Lower 0.0 pixels

☑ Rainbow

Time to process image: 0.0376 seconds, Number of Frames: 1300

Shift Correction Subregion Size 64x64

25.06 fps

(500, 207)   0

Image Display Palette Saturated

Upper Slider pos: 100

Processing Region: 240, 1090 to 2290,

Camera Gain [0-48 dB] Camera 10.0 dB

Camera Temperature: 47.8 C

Figure 6.—RT_BOS_Cuda screen shot showing the user dialog controls. A live image from an experiment of a supersonic film cooling flow is being captured in this screenshot, where a region-of-interest of the image is being processed. The background speckle pattern on the 4K monitor does not fill the entire 2448×2048 pixel camera field-of-view.

heated supersonic jet flow over a film cooled plate is shown. The density gradient from the free shear layer at the top of the nozzle is readily observed. Shocks form from the cooling holes in the plate surface. Shocks are also reflecting off of the top shear layer back down to the plate surface. The flow field is very complicated and changes dramatically depending on the amount of cooling flow delivered over the plate surface. Using RT_BOS_Cuda, these changes in the cooling film layer thickness and its impact on the supersonic flow over the plate can be observed in real-time, providing on-line feedback on the cooling flow/ambient flow interactions.

The RT_BOS_Cuda software has been used with the nVidia GTX-1080 ti board for acquiring and real-time processing of image data from several types of camera sensors. The best performance is now available via a 10 GigE camera interface. Camera images at 5 MP can be acquired via 10 GigE and processed at a frame rate of 67 frames/s using 64×64 pixel subregions on a 32×32 pixel grid. An RT_BOS_Cuda system connected to a FLIR 10 GigE camera with 12 MP resolution can achieve a sustained processing rate of 27 fps on the nVidia GTX-1080 ti board. The CameraLink version of the program has been used with an Imperx 29 MP camera operating at its full frame rate of 5 fps when processed on an nVidia Quadro M6000 board with 24 GB of on board RAM. All of these processed camera frame rates will continue to improve with memory bandwidth on GPU boards.

4.2.1 Active Error Correction

The ability to process and display the BOS image data in real time reveals any deficiencies in the optical system performance during the test. This is critical information that is sometimes not apparent until after the test, at which point there is no way to correct it. In most BOS systems, the camera and speckle pattern are rigidly mounted so that no vibration or movement occurs between the camera and the background speckle pattern from the time the reference image is acquired until the live image data are acquired. In some installations, there is still some movement between the camera and the background reference pattern due to either high vibration from the test facility and/or from other installation effects. For example, in one test facility using the RT-BOS system at the NASA Glenn Research Center, the BOS camera had to be mounted on a large traverse rig which also held a survey rake for profiling the flow field downstream of different mixing enhanced nozzle flows for aeroacoustics research work. The BOS reference image is acquired before the test begins with the traverse system in its nominal home position for the BOS measurements. However, sometimes a probe survey of the nozzle flow is required before BOS measurements are required, which means the traverse system must be moved from the original home location. After the surveys are completed the traverse is returned to the home position, however; the BOS camera system clearly shows that the home location has not been realized. The real-time processed BOS image data show a large DC shift across the image. The large DC shift substantially reduces the sensitivity of the system to small density gradients, or yields a total loss of correlation.

In order to mitigate this reference image shift error, an on-line image shift correlation-correction feature is implemented into RT_BOS_Cuda. Typically, there are regions in the BOS camera images where there are little or no density gradients within the camera field of view. These may be regions of the speckle pattern where there is no flow or parts of the facility rig or window frame that have no net flow, which means they have not changed relative to the reference image. The user can set the location and size of a single subregion in the stored reference image, which is cross-correlated with each new live image that is obtained from the camera. The size of the correlation subregion used is based on the magnitude of the anticipated image shift. Larger subregions are tolerant of larger image shifts. The computed correlation peak from this reference image check shows any net displacement of the live image relative to the reference. The shift (x_{shift}, y_{shift}) from the reference image is determined to subpixel accuracy. Since a single correlation subregion is being used, the computation overhead for determining this shift is negligible.

The computed live image shift relative to the reference image is then used to linearly interpolate the live image back to the location of the reference image, so that there is no net shift between the reference image and the live image when they are cross-correlation processed. The subpixel image shifting uses a bilinear interpolation using (x_{shift}, y_{shift}) values determined in the image registration cross-check. The bilinear interpolation is done in real-time as the images are acquired, before the subregion by subregion cross-correlation with the reference image to compute the density gradient map across the flow. Any DC offsets between the reference image and live image are essentially nulled out by this process. The raw data are unaffected, this is purely an on-line, visual correction for the user. Applying the subpixel image shift to the live images has a significant impact to the performance of the Intel CPU implementation of the RT_BOS_Cuda program. However, in the GPU version, the image shifting operation adds no discernable impact on the performance, using a single custom kernel to perform the image shift. The on-line adaptive image shifting feature makes the RT-BOS system robust against facility vibrations or other unintended misalignments between the camera and the reference speckle pattern after the reference image is acquired and testing has begun.

4.2.2 On-Line BOS Optimization

The need to generate and illuminate the speckle patterns in BOS is a formidable challenge in implementing the technique. The use of a self-illuminating background with an electronically generated speckle pattern greatly simplifies implementing the BOS technique. The performance and/or sensitivity of the BOS system is dependent on the resolution of the camera used to record the images, the size of the speckles on the background relative to their imaged size relative to the pixels on the camera and the magnitude of the density gradients under study. The use of an electronically generated speckle pattern and real-time processing enables the researcher to optimize the performance of the BOS system by dynamically changing the speckle pattern characteristics and observing their effect on the system sensitivity, yielding an optimally performing system.

A separate optimization program was written, which uses a database of generated speckle images of various speckle sizes, density and contrast to optimize a BOS system in the lab. The BOS optical system is preconfigured as it would be used in the test facility, then the full range of speckle patterns are displayed on the monitor and image data acquired and processed. The processed density gradient maps are analyzed to determine the configuration with the smallest background noise level, which yields the maximum system sensitivity.

The optimization is even possible while the RT-BOS system is under operation, increasing the sensitivity of the system in regions of low density gradients. The process is relatively simple. Multiple reference images are acquired prior to the start of the testing, where each reference image is from a preselected set of speckle images, with differing speckle size and concentration. Then while the test rig is in operation, image data are acquired using each of the selected background speckle patterns displayed on 4K monitor. The previously saved reference images can be loaded into the RT_BOS_Cuda software while the background image on the 4K monitor is loaded for the corresponding reference image. The quality of the real-time displayed density maps readily informs the user of the optimal background speckle pattern. This approach yields an *in-situ*, on-line optimization of the BOS technique for maximum data quality.

4.2.3 Post Processing BOS

RT-BOS provides on-line assessment of the experimental operating conditions and optimization of the BOS instrumentation itself. All of the data processing code developed for RT_BOS was designed for speed of data processing and display of the results. In addion to the real-time processing and display, RT_BOS_Cuda also streams the acquired image data to disk. The data collected using RT_BOS_Cuda can be easily be imported into commercial BOS data analysis packages after the test is complete. These commercial packages have many advanced data processing and averaging features to enhance the post-test data. LaVision's DaVis 8.4 StrainMaster software was used to post-process the BOS data collected in this work (Ref. 15).

5.0 Implementations of Self-Illuminated BOS

The RT_BOS_Cuda program has been used in several research programs at NASA Glenn to study a wide variety of flow fields including reaction control rockets inside a vacuum chamber, a turbine cascade rig with 3 in. thick Plexiglas windows, and various mixing enhanced nozzle flows in the AeroAcoustic Propulsion Lab (AAPL). Examples of two of these applications are shown in Figure 7, Figure 8, Figure 9, and Figure 10. BOS was required to study the shock patterns generated in a turbine cascade facility, where the cascade endwalls were made of 3 in. Plexiglas, see Figure 7. Standard knife edge Schlieren would not be feasible through such low quality windows. However, BOS was implemented in this test using a self-illuminated background on the far side of the test rig and the BOS camera mounted on the near side of the rig, as shown in Figure 7. A sequence of 400 BOS images were post–processed using the DaVis StrainMaster software package. The shock pattern generated through the turbine cascade is shown in Figure 8, for a NASA blade design. The profiles of the blades are discernable in the figure and the flow enters from right and exits the blade row to the upper left. Another example of the self-illuminated BOS approach is shown in Figure 9, where BOS was required to study reaction control rocket jet plumes inside a vacuum chamber with limited optical access in NASA's Altitude Combustion Stand (ACS). Both the camera and the self-illuminated speckle background had to be mounted inside a vacuum chamber. The use of the self-illuminated background and small BOS camera greatly simplified the implementation of BOS in this facility. A sample post-processed average density gradient map of the 0.105 in. diameter nozzle flow at plenum pressure of 1000 psia, evacuating into a 0.2 psia chamber is shown in Figure 10. The extent and shape of the nozzle plume is evident in these extremely low density vacuum chamber flows.

Using the RT_BOS_Cuda program in real-test environments reveals any deficiencies in the technique and enables incorporation of new enhancements, such as the real-time active image shift/error correction. Due to its simplicity of setup and operation, including self-illuminated speckle background, RT-BOS is now being used as a standard facility instrumentation in several rigs at NASA Glenn so that the operation of the test article and facility performance can be monitored in real time. An added advantage of the real-time live image is that any motion of the model is also captured in the BOS images.

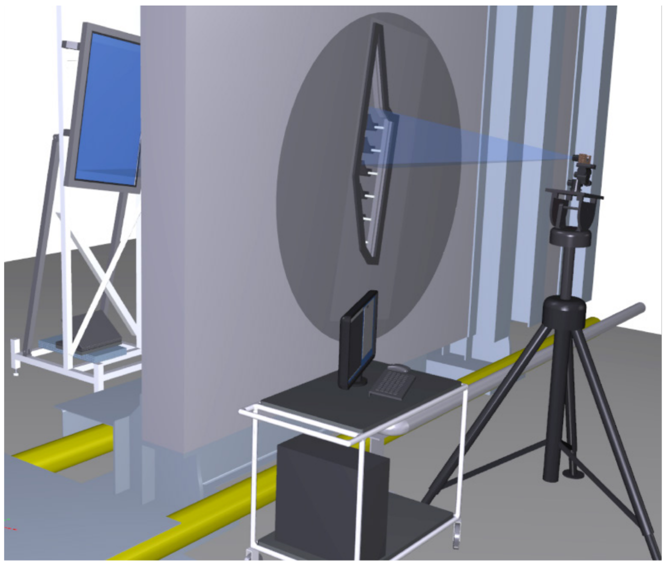


Figure 7.—BOS Installation in CW-22, camera views speckle image on monitor through blade row test section.

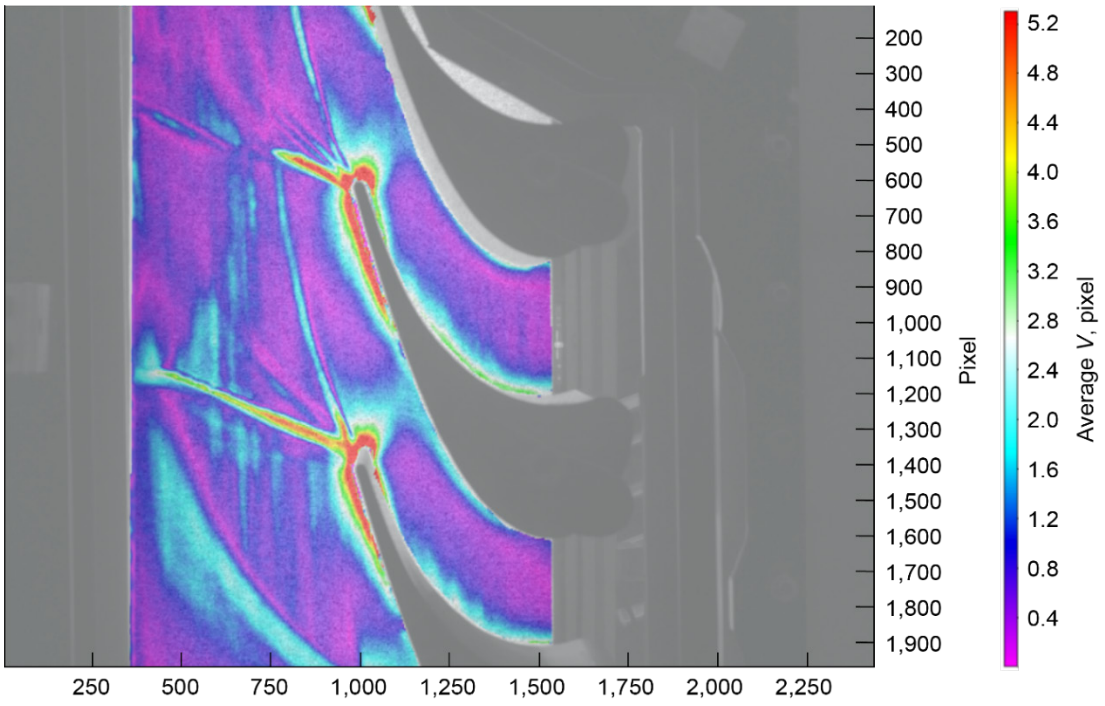


Figure 8.—Sample BOS processed image. Note the boat tail shocks at the trailing edge of the blade. (07Apr17zp_Strain_LSM_TimeSeries2D_subset=15_step=4_Avg_Stdev)

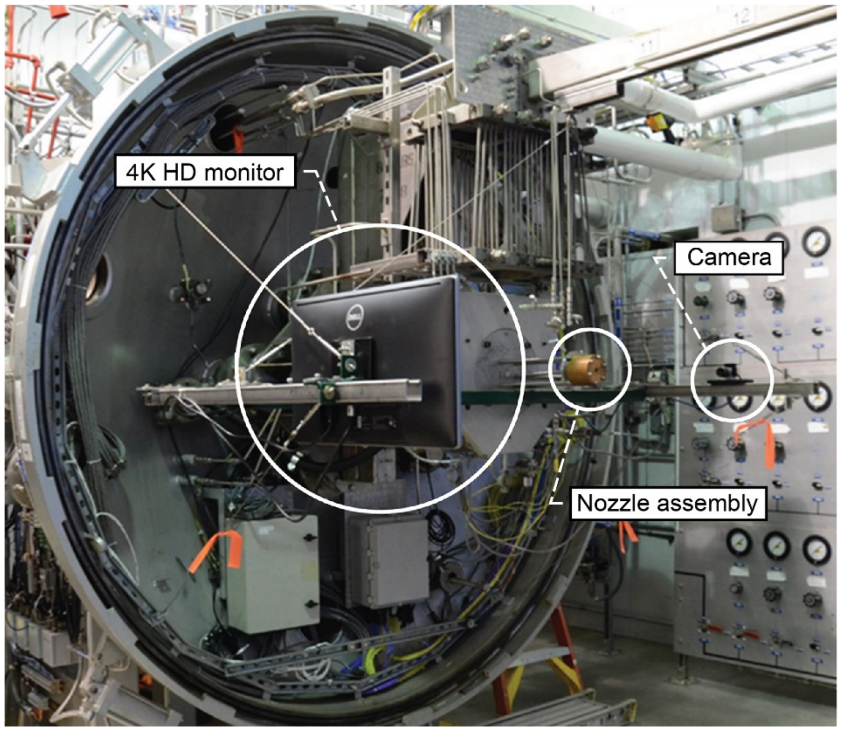


Figure 9.—BOS installation in ACS, the camera views the background image on the monitor through the nitrogen gas simulated rocket exhaust plume. In this photo the capsule is retracted downstream in order to provide access to the BOS hardware.

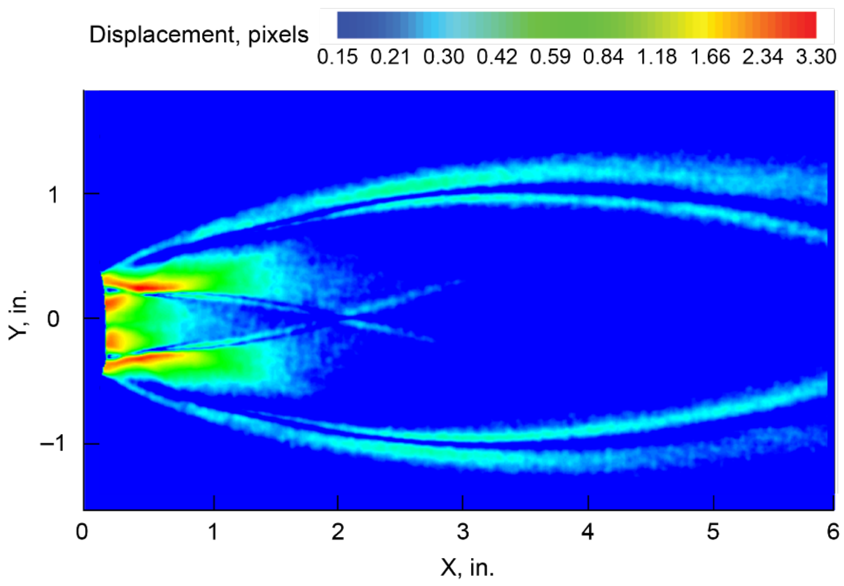


Figure 10.—Color contours of the average magnitude of the displacements due to the density gradients for the 0.105 in. diameter nozzle at a plenum pressure of 1000 psi.

6.0 Conclusion

The Background Oriented Schlieren technique is widely used across a range of engineering applications. Traditionally, the BOS image data are acquired during facility operation, but are not processed into density gradient fields until after the test is complete due to the computationally intensive data reduction required to process the images. A new Real-Time BOS system has been developed at NASA Glenn which enables the on-line monitoring of the density gradients in the flow fields of interest while the test is being conducted. The RT_BOS_Cuda software acquires, processes and displays BOS image data in real-time on the data acquisition computer. A cross-correlation engine application was implemented using both an Intel-multithreaded approach and a GPU based approach. The GPU approach provides the highest performance for RT_BOS and is the preferred approach. A new approach to generating the background patterns in BOS was implemented, where a 4K HD monitor was used to provide a self-illuminated speckled background. A single PC can be used to both display the 4K speckle background images on the monitor and run the RT_BOS_Cuda software for real-time acquisition and display of the BOS data. Direct computer control over the speckle pattern simplifies the generation of the background and facilitates optimization of the background for each experimental setup. An approach for on-line optimization of the RT-BOS system was described using the unique features of real-time processing and electronic background speckle patterns – all under computer control. An adaptive sub-pixel shift correction was also implemented to mitigate any shifts in the optical system, due to vibration, translation or thermal growth, making the RT-BOS system resilient in real-world facility installations. For the first time ever, BOS data can be acquired and processed in real-time and the BOS system performance optimized in real-time using the self-illuminating background. Examples of BOS applications at GRC using the self-illuminated background were presented, illustrating the simplicity of implementing the technique in challenging test environments.

Real-time processing, when combined with self-illuminating background makes a compact, easily configurable and easily deployable BOS system. Optimization of the backgrounds optimizes the system performance. All of these enhancements improve the ease of installation, operation and optimization of BOS in research testing facilities, further extending the applicability pf the BOS technique in aerospace simulation facilities. Now even a modest PC can provide RT-BOS if equipped with an nVidia GeForce GTX-1080ti board.

References

1. Settles, G.S., "Schlieren and Shadowgraph Techniques: Visualizing Phenomena in Transparent Media," Springer-Verlag, ISBN 3-540-66155-7, 1949.
2. Raffel, M., "Background Oriented Schlieren (BOS) Techniques," Exp. in Fluids 56:60, DOI 10.1007/s00348-015-1927-5, 2015.
3. Hild F., Roux S., "Digital image correlation: from displacement measurement to identification of elastic properties – a review". Strain 42 (2):69–80, 2010.
4. Hargather M.J. and Settles G.S., "Natural-background-oriented schlieren imaging," *Exp. Fluids* 48, pp. 59–68, 2010.
5. Heineck, J.T., Banks, D., Schairer, E.T., Haering, E.A., and Bean, P. "Background Oriented Schlieren (BOS) of a Supersonic Aircraft in Flight," AIAA Flight Testing Conference, AIAA AVIATION Forum, (AIAA–2016–3356).
6. Wernet, M.P., Stiegemeier, B.J., "Application of Background Oriented Schlieren for Altitude Testing of Rocket Engines," NASA/TM—2017-219578.
7. Arik, E.B. and Carr, J., "Digital Particle Image Velocimetry System for Real-time Wind Tunnel Measurements," ICIASF '97, Sept. 1997, pp. 267–277.
8. Fujiwara T., Fujimoto K., Maruyama T., "A Real-Time Visualization System for PIV," Field Programmable Logic and Application. FPL 2003. Lecture Notes in Computer Science, vol. 2778. Springer, Berlin, Heidelberg 2003. DOI https://doi.org/10.1007/978-3-540-45234-8_43

9. Yu, H., Leeser, M., Tadmor, G., Siegel, S., “Real-time particle image velocimetry for feedback loops using FPGA implementation,” Journal of Aerospace Computing, Information, and Communication (AIAA), vol. 3, pp. 52–57, Feb. 2006.
10. Tarashima, S., Tange, M., Someya, S., Okamoto, K “GPU accelerated direct cross-correlation PIV with window deformation,” 15th Int Symp on Applications of Laser Techniques to Fluid Mechanics, Lisbon, Portugal, 05–08 July, 2010.
11. Schiwietz, T., Westermann, R., “GPU-PIV,” *Proceedings of the vision, modeling, and visualization conference*, Girod B., Magnor, M.A., Seidel, H.P. (eds), Stanford, CA, pp. 151–158, 2004.
12. Cooley, J.W. and Tukey, J.W., “An Algorithm for the Machine Calculation of Complex Fourier Series,” Mathematics of Computation, Vol. 19, No. 90 (Apr., 1965), pp. 297–301.
13. EPIX, PIXCI-E8 Framegrabber and XCLib Library manual, http://www.epixinc.com
14. AB-Soft ActiveGigE Users Guide, http://www.ab-soft.com/ activegige.php
15. LaVision, Inc. DaVis 8.4 StrainMaster User manual, https://www.lavision.de/en/downloads/manuals/