TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Improving digital correlation algorithm for real time use

Study programme: P3901 – Applied Sciences in Engineering
Study branch: 3901V055 – Applied Sciences in Engineering

Author: **Ing. Petr Ječmen**
Supervisor: doc. RNDr. Pavel Satrapa, Ph.D.

Liberec 2017

Abstract

In this work, a set of improvements to DIC algorithm is presented. The result of these improvements should make the DIC algorithm more user friendly and better usable for common users.

First set of improvements focuses on implementing the DIC algorithm using OpenCL programming language. This allows to run the algorithm on wide range of available hardware, most notably on GPUs. As tests show, running DIC on GPU leads to significant speedup (reaching 30× compared to basic variant and 10x compared to threaded variant). Further improvements focus on optimizing the data size in order to lower the overhead of RAM to GPU transfers and a study on how the OpenCL implementation performs on integrated GPUs and CPUs.

Next improvement processes the input data in order to enhance the specimens texture to improve the quality of the correlations. The experiments show improvement of the quality of the results, but they are redeemed in increased computation time.

Last improvement is a design of an fully automatic algorithm that selects the best subset size to get the best results possible. The algorithm tries to find the optimal subset size to balance the systematic and random errors by monitoring the function of correlation quality versus subset size.

Contents

List of abbreviations

API	Application Programming Interface
CC	Cross-Correlation
CCS	Charge-Coupled Device
FFT	Fast Fourier Transform
CUDA	Compute Unified Device Architecture
DIC	Digital Image Correlation
DRAM	Dynamic Random Access Memory
GPGPU	General-purpose Computing on Graphics Processing Units
GPU	Graphics Processing Unit
GUI	Graphical User Interface
JOCL	Java bindings for OpenCL
NR	Newton Raphson
OS	Operating System
PIV	Particle Image Velocity
OpenCL	Open Computing Language
RAM	Random Access Memory
ROI	Region of interest
SEM	Society for Experimental Mechanics
SIMD	Single Instruction, Multiple Data
SIMT	Single Instruction, Multiple Threads
SSD	Sum-Squared Difference
VLIW	Very Long Instruction Word
ZNCC	Zero Normalized Cross-Correlation
ZNSSD	Zero Normalized Sum-Squared Difference

1 Introduction

1.1 Problem statement and motivation

Surface deformation measurement of materials and structures subjected to various loadings (e.g. mechanical loading or thermal loading) is an important task of experimental solid mechanics. There is a great need for precise results in order to be able to derive mechanical properties of the specimen. Traditional contact method using strain gauges offers good precision, but it does not offer a full field analysis (besides using multiple gauges, which is not practical). Two-dimensional digital image correlation (2D DIC) is now widely accepted and commonly used as a practical and effective tool for quantitative in-plane deformation measurement of a planar object surface. It directly provides full-field displacements to sub-pixel accuracy and full-field strains by comparing the digital images of a test object surface acquired before and after deformation.

While DIC is able to provide very precise and detailed results, the algorithm has several drawbacks that hinder it's use by general audience. First, the running time of the algorithm might be quite long if the the user wants most precise results possible, usually in the range of tens of minutes to hours. Another drawback is the sensitivity to input parameter settings. The DIC algorithm usually requires only one parameter, the window size. The wrong value leads to very imprecise results and there is no general rule of thumb by which the size can be picked. There are some algorithms that offer automatic size computation, but they usually increase the runtime considerably, thus trading one drawback for another. Lastly, typical user has little experience with setting up the experiment. The quality of results is of course dependent on the quality of the input data, so the user can unknowingly decrease the precision even before the experiment has started.

1.2 Goals of this dissertation

The main objective of this thesis is to improve the drawbacks of the DIC algorithm. The improvements can be split into 3 general areas.

First objective is to improve the running time of the algorithm, preferably in a way that can be directly used (or at least easily implemented)

by existing solvers. The improvement should not be limited to specialized hardware of software, because a typical user usually uses a basic setup without any special gear that allows fast numerical computations.

Another goal is to provide a way of automatic processing of input data to improve the quality of the results. The preprocessing however must ensure that the precision will be increased or kept same at worst for different data sets, because the user might not be able to tell if the result is incorrect or why the results is incorrect.

Last objective is to devise an algorithm that estimates optimal value of window size without the need of user input. The algorithm should be able to support different solvers (similarly to execution time improvement), because different solvers can offer better results for different scenarios, but all of them can benefit from optimal window size.

1.3 Organization

This work is split in four main chapters.

- Chapter 1 sums the motivation of this works along with the goals.
- Chapter 2 contains an introduction to DIC algorithm with theoretical background
- Chapter 3 presents the results of the research towards improvement of DIC algorithm in order to ease its use for general audience
- Chapter 4 concludes the reached goals and proposes ideas for future research

2 Theory and Background

Traditional surface deformation measurement of materials and structures uses point-wise strain gauge technique, but various full-field non-contact optical methods exist [Rastogi, 2000], including both interferometric techniques, such as holography interferometry, speckle interferometry and moire interferometry, and non-interferometric techniques, such as the grid method [Sirkis and Lim, 1991, Goldrein et al., 1995] and digital image correlation (DIC), have been developed and applied for this purpose.

Interferometric metrologies require a coherent light source, and the measurements are normally conducted in a vibration-isolated optical platform in the laboratory. Interferometric techniques measure the deformation by recording the phase difference of the scattered light wave from the test object surface before and after deformation. The measurement results are often presented in the form of fringe patterns; thus, further fringe processing and phase analysis techniques are required. Non-interferometric techniques determine the surface deformation by comparing the gray intensity changes of the object surface before and after deformation, and generally have less stringent requirements under experimental conditions.

As a representative non-interferometric optical technique, the DIC method has been widely accepted and commonly used as a powerful and flexible tool for the surface deformation measurement in the field of experimental solid mechanics. It directly provides full-field displacements and strains by comparing the digital images of the specimen surface in the un-deformed (or reference) and deformed states respectively. In principle, DIC is an optical metrology based on digital image processing and numerical computing.

2.1 Fundamentals of 2D DIC

In general, the implementation of the 2D DIC method comprises the following three consecutive steps, namely (1) specimen and experimental preparations; (2) recording images of the planar specimen surface before and after loading; (3) processing the acquired images using a computer program to obtain the desired displacement and strain information. In this section, issues on specimen preparation and image capture are introduced first. Then, the basic principles and concepts of 2D DIC are described.

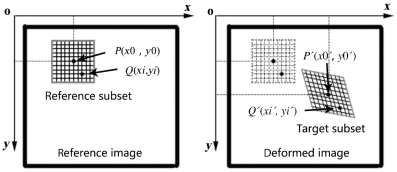


Figure 2.1: Illustration of deformation

2.1.1 Basic principles and concepts

After recording the digital images of the specimen surface before and after deformation, the DIC computes the motion of each image point by comparing the digital images of the test object surface in different states. In the following, the basic principles and concepts involved in 2D DIC are introduced.

Basic principles

In routine implementation of the 2D DIC method, the calculation area (i.e. region of interest, ROI) in the reference image should be specified or defined at first, which is further divided into evenly spaced virtual grids. The displacements are computed at each point of the virtual grids to obtain the full-field deformation.

The basic principle of 2D DIC is the tracking (or matching) of the same points (or pixels) between the two images recorded before and after deformation as schematically illustrated in figure 2.1. In order to compute the displacements of point P, a square reference subset of (2M + 1) x (2M + 1) pixels centered at point $P(x_0, y_0)$ from the reference image is chosen and used to track its corresponding location in the deformed image. The reason why a square subset, rather than an individual pixel, is selected for matching is that the subset comprising a wider variation in gray levels will distinguish itself from other subsets, and can therefore be more uniquely identified in the deformed image.

To evaluate the similarity degree between the reference subset and the deformed subset, a cross-correlation (CC) criterion or sum-squared difference (SSD) correlation criterion must be predefined. The matching

procedure is completed through searching the peak position of the distribution of correlation coefficient. Once the correlation coefficient extremum is detected, the position of the deformed subset is determined. The differences in the positions of the reference subset center and the target subset center yield the in-plane displacement vector at point P, as illustrated in figure 2.1.

Shape function/displacement mapping function

It is reasonable to assume that the shape of the reference square subset is changed in the deformed image. However, based on the assumption of deformation continuity of a deformed solid object, a set of neighboring points in a reference subset remains as neighboring points in the target subset. Thus, as schematically shown in figure 2.1, the coordinates of point $Q(x_i, y_j)$ around the subset center $P(x_0, y_0)$ in the reference subset can be mapped to point $Q'(x'_i, y'_j)$ in the target subset according to the so-called shape function [Schreier and Sutton, 2002] or displacement mapping function [Lu and Cary, 2000]:

$$\begin{aligned} x'_i &= x_i + \xi(x_i, y_j) \\ y'_j &= y_j + \eta(x_i, y_j) \qquad (i, j = -M : M) \end{aligned} \tag{2.1}$$

Correlation criterion

As already mentioned, to evaluate the similarity degree between the reference and deformed subsets, a correlation criterion should be defined in advance before correlation analysis. Although different definitions of correlation criteria can be found in the literature, these correlation criteria can be categorized into two groups, namely CC criteria and SSD correlation criteria [Giachetti, 2000, Tong, 2005].

Interpolation scheme

As can be seen from equation 2.1, the coordinates of point (x'_i, y'_j) in the deformed subset may locate between pixels (i.e. sub-pixel location). Before evaluating the similarity between reference and deformed subsets using the correlation criterion, the intensity of these points with subpixel locations

must be provided. Thus, a certain subpixel interpolation scheme should be utilized. In the literature, various sub-pixel interpolation schemes including bilinear interpolation, bicubic interpolation, bicubic B-spline interpolation, biquintic B-spline interpolation and bicubic spline interpolation have been used. The detailed algorithms of these interpolation schemes can be found in numerical computing books [Press et al., 2007]. However, a high-order interpolation scheme (e.g. bicubic spline interpolation or biquintic spline interpolation) is highly recommended by Schreier et al [Schreier et al., 2000] and [Knauss et al., 2003] since they provide higher registration accuracy and better convergence character of the algorithm than the simple interpolation schemes do.

2.2 Displacement field measurement

Due to the discrete nature of the digital image, the integer displacements with 1 pixel accuracy can readily be computed. To further improve displacement measurement accuracy, certain sub-pixel registration algorithms should be used [Bing et al., 2006]. Generally, to achieve sub-pixel accuracy, the implementation of 2D DIC comprises two consecutive steps, namely initial deformation estimation and sub-pixel displacement measurement. In other words, the 2D DIC method normally requires an accurate initial guess of the deformation before achieving sub-pixel accuracy. For example, for the most commonly used iterative spatial cross-correlation algorithm (e.g. the Newton Raphson method), it only converges when an accurate initial guess is provided (the radius of convergence is estimated to be smaller than 7 pixels in the evaluation tests performed by Vendroux and Knauss [Vendroux and Knauss, 1998]). As for the coarse fine algorithm and the peak-finding algorithm, an integer displacement of 1 pixel resolution must be provided before further sub-pixel displacement registration.

2.3 Strain field estimation

Now, we can get the full-field displacements to sub-pixel accuracy using the algorithms described above. However, as in many tasks of experimental solid mechanics such as mechanical testing of material and structure stress

analysis, full-field strain distributions are more important and desirable. Regrettably, less work has been devoted on the reliable estimation of strain fields. Presumably, this can be attributed to the fact that the displacement gradients (i.e. strains) can be directly calculated using the NR, quasi-NR, LM or genetic algorithm. Alternatively, the strains can be computed as a numerical differentiation process of the estimated displacement.

2.4 DIC parallelism

It is quite surprising that a lot of effort has been given to improve the quality of results but there are very few works devoted to increase the speed of computation of the algorithm using available hardware. It is probably due to the age of the method, because first implementations of DIC algorithm are dated around 1988, when processors had only one core and GPUs (graphics processing units) were unusable for mathematical operations. As can be seen from equation 2.1 and tables **??** and **??**, the computation for one facet and deformation is isolated from others. The synchronization point, where all workers must meet, is after the correlation is computed, so the best result can be picked. With this in mind, we can create parallel version of the task quite easily in terms of task splitting. Problem with this approach can be in memory management, because every worker needs different data, so it might be problematic to transfer the data to all workers in time. This can be overcome using grouping workers according to some common parameter (e.g. same subset or deformation), which can save a lot of memory bus bandwidth.

Parallel computation can be achieved in multiple ways. We can use multiple computers to compute the task, we can split the task between multiple processors / processor cores or we can use GPU to compute the task. Every approach has its advantages and disadvantages. Our main goal is to present a user-oriented approach, which does not require any complex configuration or time-consuming setup. This leaves out multi-PC approach, because it is often quite cumbersome and difficult for a common user to setup computers to work as a cluster. CPU parallel approaches are quite common in today's world, so we won't focus much on the parallel implementation using CPU, however we will use results from this approach

as the baseline for comparison and as a fail-safe approach when no GPU is available for computation.

2.4.1 Task splitting

In DIC case, splitting the task is quite easy. The task is to compute correlation of original facet with a facet after some deformation. We have to compute multiple deformations for each facet and the task usually consists of multiple facets. So we need to perform $N_F \times N_D$ correlations, where N_F is the number of facets and N_D is the number of deformations. From earlier we know, that computation of each sub-task (one deformation of one facet) is independent from others, they all meet only when the best result is selected. We can see, that we can split the task facet-wise or deformation-wise. We tested both variants, because the flow of the computation, memory access pattern and merging of sub-task results are different for each of mentioned ways and may provide different results.

2.4.2 Parallelism on GPU

The nature of DIC algorithm makes it an ideal candidate for GPU computation - large number of computations, that can be run in parallel. Our work differs from other papers in two factors; first the implementation is done using OpenCL (as opposed to CUDA for others), which offers inter-platform compatibility. As far as we know, no other work inspects this approach. The second difference is that our paper focuses on maximal performance optimization and we present a thorough analysis on how to improve performance of DIC algorithm using OpenCL.

3 Research and Results

This chapter sums all results in order to improve the DIC algorithm for real-time use. Chapter 3.1 is a study of practical implementation of DIC algorithm using OpenCL programming language. It presents a step-by-step study on optimal kernel creation. Chapter 3.2 further extends the effort of improving the performance by studying the influence of input data format on algorithm performance. The aim of the performance evaluation is to decide if it is better to generate the input data on CPU beforehand and transfer it to GPU or if it is better to generate the data on GPU. Following chapter 3.3 presents a study on OpenCL's ability to run one kernel on different device types (such as GPU, CPU etc.) in context of DIC algorithm. The goal of the study is to determine if it is better to create just one kernel and use it on all device types or to spend more time programming in order to create several kernels, each optimal for a subset of device types. Last two chapters (3.4 and 3.5) aim to improve the quality of the results without the need of user input. Chapter 3.4 aims to improve the quality of the results by pre-processing the image using filters. Several filters are evaluated to see if they can somehow improve the result's quality without big performance overhead. Last chapter 3.5 aims to remove the need of user input of subset size by automatically determining the size using the correlation quality.

3.1 OpenCL on GPU

While GPU proved to be a good choice for DIC algorithm for improving performance, there is no work performing a detailed analysis which part of the algorithm is the bottleneck and is slowing the rest of the computation. Also all the works use CUDA as the language of implementation, thus reducing the target audience by half (at least). We want to change that and have performed a detailed study on how the performance improves when using different implementations of different parts of the DIC algorithm. We have also utilized OpenCL as implementation language allowing everybody with supported HW (almost all consumer grade graphic cards) to benefit from our research.

We have used Java ([Oracle, 2017]) as main programming language. OpenCL offers API in form of C language headers, so we had to use some

kind of wrapper, which will allow us to call methods of the API via Java method calls. We decided to use JOCL library ([JogAmp, 2017]), because it has low overhead due to direct Java code generation from C API.

For testing we used correlation part of DIC algorithm (given set of facets and deformations, compute correlation value for each combination of facet and deformation coefficients), facets were square shaped and as order deformations we chose first order (6 coefficients). First order of deformations provides good accuracy while keeping running time relatively low. In our test we have used 9 data combinations - 200 / 5000 / 100000 deformations, facets of sizes 7 / 20/ 40 and ROI size 88 x 240 px (resulting of facet counts of 408 / 48 / 12). This allowed us to observe how the variants behave in different task sizes so we don't end up with an algorithm that does well for big tasks but is very bad for smaller cases.

We measured not only total time of execution, we also measured time spent in „preparation" of task (data copying, changing data order etc.) so we could better see where are the pitfalls of given variant. For time of kernel execution, OpenCL offers functions to obtain time needed for execution, other times (total time and time for pure Java implementation) were measured using Java built-in method *System.nanoTime()*, which offers good precision without the need of external libraries.

We have run the test on multiple devices to see how the algorithm scales and how it behaves on different device types. The testings devices were following:

- Intel Core i7 3610QM
- Intel Core i7 4700MQ
- NVIDIA Geforce GT650M
- NVIDIA Geforce GTX765M
- Intel HD4000

As can be seen, we used devices present in modern notebooks. We did this because of two reasons; one is that notebook is quite common as a main device for work and thus the results will be more real-world oriented, second is that if the variants behave better on notebook hardware, they will behave in same way (or most probably better) on desktop hardware. In this paper, we will present numerical results from GT650M along with notes on how the variants behave on other devices.

3.1.1 CPU implementation

Given complete test set, we can choose if we want to compute all deformations for one facet first (we call it per-facet) or try one deformation for all facets (per-deformation). In case of per-facet variant, threading can

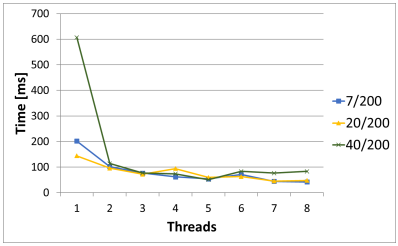


Figure 3.1: Computation time vs. thread count

help a lot (cutting time by as much as 75%), but the improvement won't get much better after using more threads than 4, in some cases the result with 8 thread was slightly worse. The per-deformation variant behaves much better in terms of speed improvement with multiple threads. In comparison of per-facet and per-deformation is no clear winner. As the conclusion of Java implementation of correlation calculation, it is good to use threads and as a silver bullet we can use as much of them as we can (ideally one thread per physical core, not virtual core) and select variant according to ratio of facets to deformations.

3.1.2 GPU porting

The code for computation on GPU has two parts; first is in Java using JOCL library, which consist of data preparation and loading to / from GPU, second part is the kernel written in OpenCL language. In previous section there was virtually no data preparation required (we generated data beforehand to minimize error in time measurement due to background OS operations), but for GPU you need to transfer the input data to GPU and results back from GPU which can take a lot of time compared to computation time.

Results graph

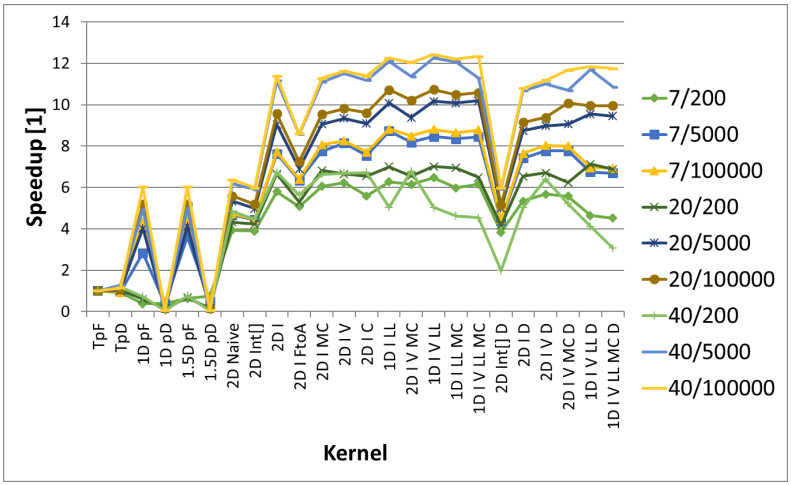


Figure 3.2: Speedup of kernels compared to base variant

Figure 3.2 presents complete results of our optimization process. For each optimization, we have picked the best result in terms of absolute computation time. The graph presents speedup against base variant, which was Java per facet variant. The kernel names are created as abbreviations of section titles. Last 6 kernels are the driven variants (D at the end of the name).

We have implemented and tested following variants:

- Direct code conversion to OpenCL
- 1.5D kernel
- 2D kernel
- image2D_t
- Store facet to private memory

- Memory coalescing
- Vector data types
- Using constant memory
- Using local memory
- Variant combinations

3.1.3 OpenCL platform limitations

Reaching the best performance of kernel is not the only pitfall of OpenCL programming. Another problem is kernel running time limit. Every OS has a watchdog built-in which is trying to prevent GPU from freezing by monitoring execution time of kernel (usually a few seconds). The solution is to split the task, but now the data can be kept all on GPU until the total end computation, we only need to launch small portions of task to comply with the time limit. This is a task of feedback loop control, where we monitor the time of execution and increase or decrease the task size accordingly.

The results clearly show that the performance for driven kernels (time limited ones) is worse by as much as 5% in worst cases. But there is almost nothing we can do about it, the only improvement could be smarter task size management, but our tests have shown that even with almost maximal task size from the beginning the performance is worse by approximately 3-4%.

3.1.4 GPU vs CPU vs iGPU

The results presented so far have been all from computation on Nvidia GT650M. We tried the same bulk of tests on other devices also. We won't present the numerical results here because the direct comparison of times wouldn't show something interesting. We will try to present a comparison for all tested devices in means of impact of optimization on performance gains.

Second device we tested was Nvidia GTX765M. The behavior of kernels was quite the same, only the absolute performance if better as expected.

Next we switched to Intel CPU's to see how they behave. The results clearly present different architecture of CPU and GPU. CPU's can't handle image2d_t and have limited or none local memory. The best variant for CPU was 1D per-facet, which performed as the best in dominant part of data configurations. But OpenCL variants were still faster than Java variants. As expected, Intel Core i7 4700MQ was faster that i7 3610QM and behaved comparably.

Last device we have tried was Intel HD4000, an integrated graphics card. The performance itself wasn't bad, but the behavior of the card was quite random. In some cases the performance plummeted almost to a halt with no reasonable explanation why did it happen. From the results we could obtain it is clear that driven variants behave quite the same as on CPU.

3.1.5 Summary

To conclude all our findings, we can say that using GPU and OpenCL to speed up the computation of DIC algorithm is a very good approach. The time of best OpenCL variant was always around 30 times faster than basic Java single thread implementation and round 9 times faster than the threaded one. We saw that vectorization and local memory usage was beneficiary in almost all cases wile using memory coalescing not so much. Next good thing apart from speed improvement is that the CPU can perform other tasks (such as result post-processing, presentation etc) while the computation continues in the background on GPU so user can browse „live“ results without slowing down the computation itself.

3.2 Reducing size of DIC task data

One of the main bottlenecks observed when converting the algorithm from CPU to GPU implementation is transferring data to and from RAM to GPU. In case of GPU computation we need to transfer data each round in order to be able to present the results and possibly make further computations (such as strain analysis). In this work we will omit static cases, where data (both subset and deformation) remain the same for whole computation. In normal case, the ROI of computation changes according

to material deformations and analyzed coefficients of shape function also change according to solver needs.

Typical parameter set of OpenCL kernel for DIC computation consists of data (subsets and deformations) and parameters (subset size, item counts etc.). The memory size of parameters is usually very small (most often a couple of integers), so they can utilize constant memory and due to their size they do not pose a concern for memory bandwidth. Subset and deformation data however can get quite big for larger and more precise tasks. As mentioned before, the size of subset is calculated as $(2M+1)\times(2M+1)$, so the area of subset grows quite quickly even for small values of M. Typical task consists of tens or hundreds of subsets so the size of subset data can get quite big (we have $(2M+1)\times(2M+1)\times 2\times N$ integer values, where N is the count of subsets).

Size of deformation data is determined by two factors - solver type and order of shape function. Shape function order determines count of coefficients - 2 for zero order, 6 for first order and 12 for second order. Type of solver determines total count of tested coefficient tuples and its structure (if they are somehow ordered). Total count of deformations is also determined by count of subset, because in usual cases, the displacement field is not homogeneous and thus we need different deformations for different subsets.

Data generation

For now the data are pre-generated on CPU and then transferred to GPU, where the computation takes place. First we will focus on subset data generation, specifically generation of square subsets. Before we can generate concrete values of position for each point in subset, we need to generate subset centers. The size of subsets center data is small (2 values) compared to full subset points position data. This is the first data size optimization we tried - transfer only subset centers and generate subset point position on GPU. With typical subset size of 20 pixels, this optimization can shrink the subset data to size from 1681 values to only 2.

Generating deformations can be done in multiple ways depending on chosen solver. In most cases however, the coefficients can be ordered some way and the step is constant. This allows us to convert set of deformations to limits describing minimal, maximal and step values. In terms of data

size, limits have the same size as 3 deformations (minimal value, maximal value and steps have all same number of coefficients as deformation). So in terms of data size reduction, we can achieve pretty significant reduction, only limitation is ordering of deformations.

By using these "limits" we can effectively reduce the size of data we need to transfer to GPU. The drawback of this approach is that we need to generate the data on GPU, which can be slower than CPU data generation and also OpenCL language does not offer as good programmer comfort as traditional programming languages such as Java or C++.

3.2.1 Scenarios

OpenCL is very well known for unstable performance along different data, kernel and device. To overcome this we tested 5 kernels and 94 data combinations on GPU and CPU. Kernel types were following ones (abbreviations will be used further in text for naming the kernels):

2D Int 2D kernel using integer arrays for storing image data

2D Im 2D kernel using image2d object for storing image data

2D Im V Extension of 2D Im with vector instructions

1D Im L 1D version of kernel with local memory to reduce data loading

1D Im V L Extension of 1D Im L with vector instructions

For data variants we chose typical scenarios and their combinations. Image sizes are in pixels, subset multiplier simulates subset overlapping by generating n times more subsets.

Image size 88x240, 240x240

Deformation count 10, 50, 200, 1000

Subset size 7, 21, 41, 61

Subset count multiplier 1, 2, 4

We have tested all kernels and data variants with different work sizes and then we have picked the best result. This applies also for data generation,

where we ran benchmark for each data set to find the best work size. Testing GPU was NVidia Geforce GT 650M, testing CPU was Intel Core i7 3610QM @ 2.30GHz.

3.2.2 Results

Sample results can be found in Figures 3.3 and 3.4, results for other data configurations were very similar. After inspecting all results we came to conclusion that kernel with integer arrays behaves very differently from the rest of kernels, which use image2d class to store image data (this behavior can be clearly seen in Figure 3.4). Horizontal axis denotes used data generation variant - NO for CPU data generation, LFD are limits for both subsets and deformation and LD / LF are limits only for deformations / subsets. GPU in name means off-line GPU data generation, limit variants without GPU means on-the-fly data generation.

In general, kernel 2D Int performed best for data generated on CPU, because it was the most stable variant. All other variants (with any or both data generated off-line on GPU) performed in some cases better than original, but in most cases were slightly worse (the speedup or slowdown was always around 5%).

Kernels with image2d objects can gain a lot of performance by introducing limits. Using only subset limits (LF) the performance drops significantly. But for deformation (LD) and both limits (LFD) the performance can improve as much as 2.5 times. Off-line GPU data generation was always worse that on-the-fly data generation. Another factor for testing was scaling of our solution for larger counts of deformation and / or subsets. Effect of larger deformation count can be seen in Figures 3.3 and 3.4, all variants behave quite the same, and only the speedup curve looks cleaner. Effect of subset count on performance can be seen in Figure 3.5. We can see that the task scales almost linearly with subset count. This is due to sequential computation of subset results, only deformations are computed in parallel. We have also performed tests on CPU (also using OpenCL, but choosing Intel platform). The results clearly showed that using off-line GPU generation is pointless, the performance dropped significantly. For image2d variant on-the-fly data generation performed similarly to CPU data generation.

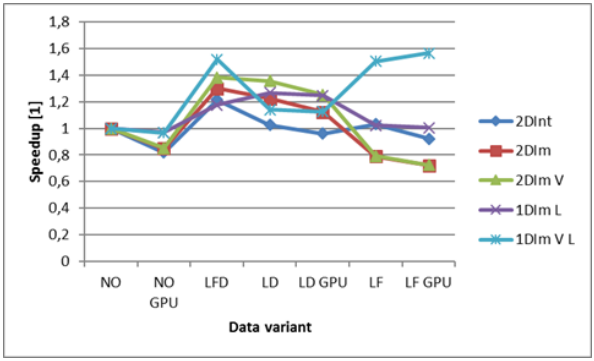


Figure 3.3: Subset size = 20, Subset multiplier = 1, Image size = 88x240, Deformation count = 50

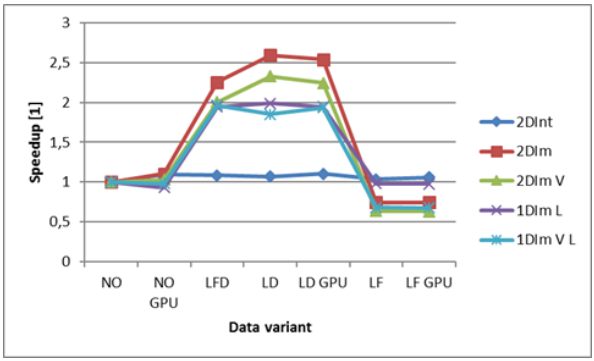


Figure 3.4: Subset size = 20, Subset multiplier = 1, Image size = 88x240, Deformation count = 1000

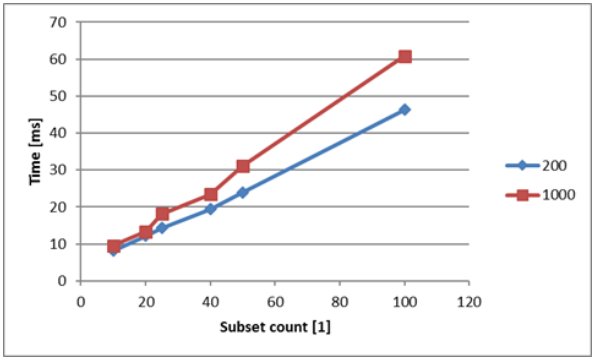


Figure 3.5: Time needed for computation with different subset counts

3.2.3 Summary

We devised two approaches for GPU data generation - off-line and on-the-fly and performed test of all possible data generation scenarios on multiple data configurations and devices. Results clearly show improvement for kernels with image2d objects using on-the-fly data generation, while off-line GPU data generation proved ineffective.

Computation using OpenCL on CPU showed that off-line data generation is pointless for this type of devices, but on-the-fly data generation variant offers same performance as basic variant. To conclude all this, we can say that on-the-fly data generation provides good performance for all device types and data configurations, so it is a good approach for speeding up the computation of digital image correlation algorithm.

3.3 DIC on different device types

While running the computation on GPU using OpenCL and generating data online can provide very good performance, the performance of the solution can vary a lot depending on hardware used. It is well known that solution that works well on NVIDIA GPUs can be quite slow on AMD cards and vice-versa. But OpenCL also allows the computation to be run

on integrated GPUs and also CPU and its architectures are different from standard GPUs, thus the performance might be also bad.

One solution to this problem is to have multiple kernels, each one optimized for different conditions (computation device, input data configuration). The downside of this approach is the count of kernels we would need to create (using optimizations found in [Ječmen and Satrapa, 2015a] and [Ječmen and Satrapa, 2015b]). In worst case scenario we would end up with 72 different kernels - input data in form of array / image2d_t, ZNCC / ZNSSD / WZNSSD correlation criterion (W meaning weighed), 1D / 1.5D / 2D kernel, use of memory coalescing and lastly use of online data generation. While possible, it would be extremely difficult to create and maintain this code base.

We have noticed that source code differences between kernel types are not big so it could be possible to generate kernel types dynamically. The user (or some kind of computation driver) will provide a kernel type configuration (input data form, correlation criterion, kernel dimension and memory coalescing and online data generation usage) and a function will generate a valid OpenCL kernel source code.

With that in mind we have created a solution, which can generate valid OpenCL kernel given a configuration. Along with it we have created a test suite, which executes different kernel configurations with various input data configurations and measures the execution time.

3.3.1 Results

To evaluate the performance of all kernels, we have measured the total time of execution, both host and device code. The execution time of device code is not enough to properly assess the performance of the kernel, because different configurations are launched a bit differently and the data generation in Java code is also different. In order to further improve the precision of the results, we launch a couple of kernels before the test in order to allow the device to initialize and load appropriate drivers.

As mentioned earlier, the performance of the kernel can vary a lot for different device, so we ran the test on multiple devices, the list follows:

- AMD Radeon HD 8750
- NVIDIA GeForce GT650M

- NVIDIA GeForce GTX765M
- Intel HD Graphics 4600 on HP ProBook 650
- Intel Core i7-3610QM

The results will be presented in the form of graphs, where horizontal axis will enumerate input data configurations and vertical axis will denote execution time in milliseconds. The input data configurations will be described using a shortcut consisting of (subset count: deformation order - deformation count). For example the configuration "02:Z-169" denotes a configurations with 2 subsets and 169 zero-order deformations. Each line in the graphs represents a kernel configurations. The kernel configurations are also described by a shortcut consisting of (kernel dimension; input data type; correlation criterion; memory coalescing; online data generation). We will use AMD Radeon HD 8750M as a baseline and compare other results to this device.

Kernel dimension

First kernel parameter we have tested was kernel dimension. The dimension determines if we let OpenCL handle data indexing or if we provide some kind of control using Java. Further details on kernel dimension can be found in [Ječmen and Satrapa, 2015a].

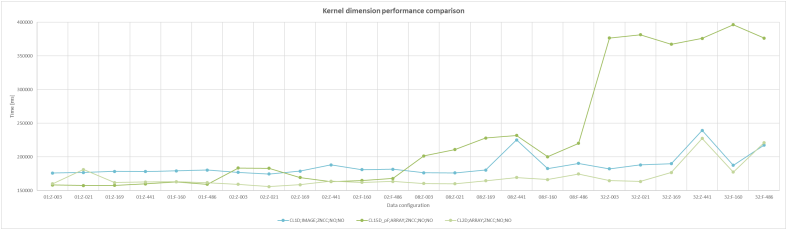


Figure 3.6: Kernel dimension performance comparison

Figure 3.6 presents comparison of best performing kernels for each dimension. The fastest is 2D kernel for almost all data configurations. The 1.5D kernel provides comparable performance as 2D kernel for data

configurations with lower subset counts, but with higher subset counts the performance plummets. The 1D kernel's performance trend is same as for 2D kernel, but it is always 10% slower.

Best 2D kernel

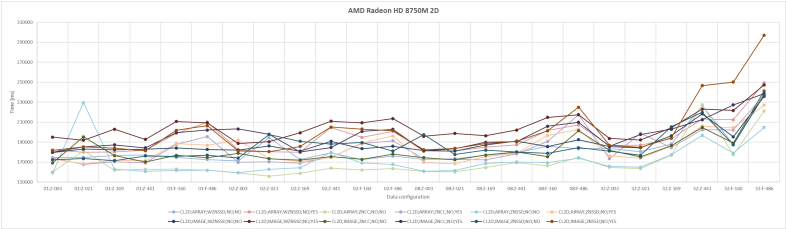


Figure 3.7: 2D kernel comparison for AMD Radeon HD 8750M

Figure 3.7 presents full results for 2D kernels launched on AMD Radeon HD 8750M. The trends for all kernels are comparable, none of the kernels behaves differently when compared to others. The best kernels are the ones without memory coalescing and online data generation. In terms of correlation criterion, ZNCC is probably a bit better, but for larger data sets, it might be good to use ZNSSD. WZNSSD is always worse than other two options simply because the criterion is more complex and thus harder to compute. We use it only for testing purposes to see, if it somehow changes the behavior of the trend and we can see that it doesn't.

NVIDIA GPU

While AMD GPU provides good performance for all data configurations, we are more interested if one kernel can provide good performance on other devices as well. We ran the same tests on NVIDIA GPUs, namely models Geforce GT 650M and Geforce GTX 765M. The results are presented if Figures 3.8 and 3.9.

From the trend we can see that the performance of all kernels is almost the same. The absolute performance is much better in comparison with AMD card. The Geforce GT 650M card should be a little bit better than

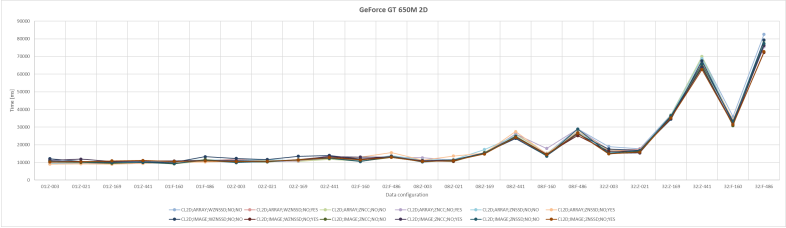


Figure 3.8: 2D kernel comparison for NVIDIA GT 650M

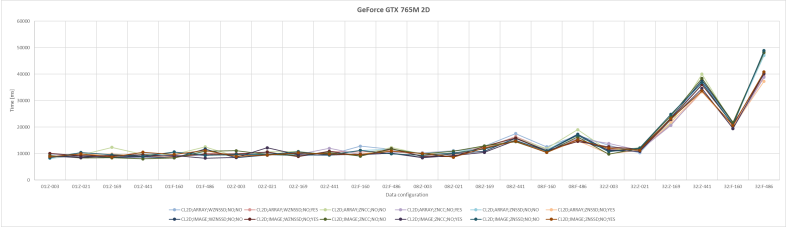


Figure 3.9: 2D kernel comparison for NVIDIA GTX 765M

AMD Radeon 8750M in terms of performance, but in our test the Geforce cards dominated by factor of 10 or more in all cases. Only for larger data sets the AMD card's performance gets better. The absolute performance still isn't better, but the performance drop is smaller for AMD card than for NVIDIA cards.

Integrated GPU

The devices we have tested so far have all been dedicated graphic cards. A lot of users have laptop as main working station and usually they don't have dedicated graphics card, in most cases the graphics is handled by on-board Intel HD graphics card (or similar integrated solution from other vendors). These graphics cards have inherently different architecture and usually they don't have any dedicated video memory, they use system

memory (RAM) instead. We ran our test on Intel HD 4600 (Haswell architecture) graphics card to see how it behaves.

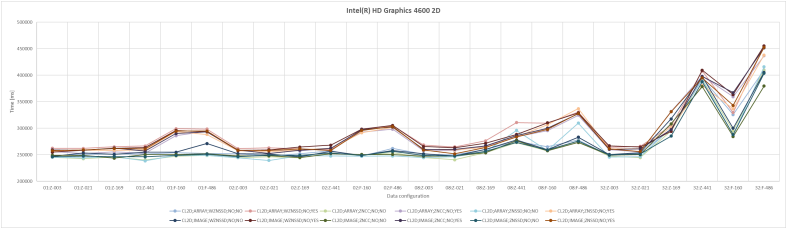


Figure 3.10: 2D kernel comparison for Intel HD 4600

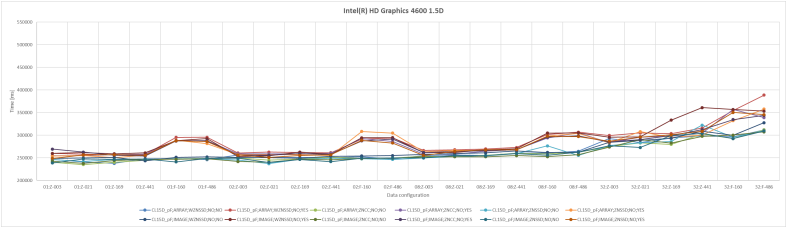


Figure 3.11: 1.5D kernel comparison for Intel HD 4600

First surprise was that 2D kernel's (Figure 3.10) performance was about 5% worse than performance of 1.5D kernel (3.11). 1D kernel was at least 10% slower for smaller data sets and for larger ones the 1D kernel was almost 2 times slower than 1.5D kernel. Figure 3.11 also shows good performance even for larger data sets and for the biggest one (32 subsets with 486 first order deformations) the performance is almost the same as for AMD Radeon HD 8750M.

CPU

Last option to run OpenCL is to run it on CPU. CPUs generally are not good candidates for OpenCL computation, mainly because of a bit different

parallelism model. CPU model uses less workers to solve larger portions of work as opposed to GPU, where there is a large count of workers (from hundreds to thousands) solving small portion of problem. To see how the CPU performs in solving DIC using OpenCL, we ran the test on Intel Core i7-3610QM CPU.

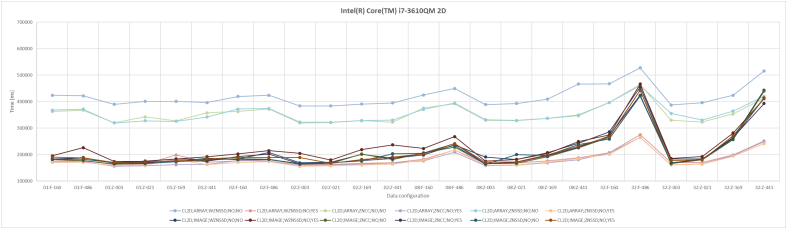


Figure 3.12: 2D kernel comparison for Intel Core i7-3610QM

The CPU performs best with 2D kernel. For small input data configurations, the CPU outperforms the integrated GPU, for larger configurations the performance drops significantly and behaves much like integrated GPUs. Dedicated GPUs outperform both CPUs and integrated GPUs in all cases.

3.3.2 Summary

During our tests we came to several conclusions. It is appropriate to use different kernel depending on used device. While 2D kernel offers good performance on all devices, on integrated GPUs it is good to use 1.5D kernel, because it handles larger data sets better.

Next we have compared all kernel configurations (input data type, use of memory coalescing and use of online data generation) in order to determine, if there is the best kernel for all correlation criteria and data configurations. For both dedicated and integrated GPUs, the best variant was without memory coalescing and without online data generation. The use of image as input data type isn't always an advantage, mainly for smaller data configurations the use of array proved to be better choice. For CPU, the use of online data generation showed significant performance improvement, in some cases up to 10%. The results confirm, that the use of image as input data type is counter-productive. This is due to the fact,

that CPUs don't have dedicated pipeline for texture processing so they process the image same way as an array, plus the conversion time.

We also wanted to find out, if it is viable to use different kernels for different input data configurations. After inspecting all obtained results, we can conclude that it is not necessary to switch kernels. If we ignore the worst kernel configurations, we see that the performance trends of all configurations are comparable, so there is no need to use different kernels for different data configurations.

Our main focus and question was wherever it is beneficial to dynamically generate the kernel according to input data configuration and computation device. Dynamic kernel generation can provide the best performance for all scenarios, even for new and unreleased devices, which support OpenCL. However dynamic kernel generation has several drawbacks, such us as the lack of ability to use OpenCL code editor to check the code for errors before compilation and profilers for performance tuning. From all this we can conclude that dynamic kernel generation, while being harder to program, is a good choice when performance is the main concern.

3.4 Image preprocessing

The basic idea of DIC algorithm is point intensity matching. From this it is clear than changes in points color due to noise can introduce systematic error, which can corrupt the result significantly. In order to reduce such error, we have tried several image filters in order to both increase result correlation quality and decrease resulting displacement errors. Zhou et al. [Zhou et al., 2015] proved, that filtering can very effectively lower the error introduced by noise. We want to extend this study by using more complex filters, which can hurt algorithms performance, but we hope the loss of performance will be compensated in result quality gain.

3.4.1 Pre-filtering setup

For testing we have selected two groups of image sets. First is a selection of image sets from Society for Experimental Mechanics (SEM, [SEM, 2017]). Image sets we used contain only shifts both in vertical and horizontal dimensions and the resulting deformations are well described, because they

serve as testing scenarios for researchers from around the world in order to be able to test and compare their solvers.

Second group of image sets are real-world stress tests captured with high-speed camera with non-ideal lightning. Examples can be seen in figure 3.16 in next chapter. The tested material is plastic and the specimen is being pulled downwards. The resulting deformations are not only shift, but also stretches, so the result analysis is not as straightforward as in case of SEM image sets.

Because the image sets have different dimensions, we generate 20 subsets for each image set in the region around the image center. We have tested 3 subset sizes - 11x11, 21x21 and 31x31, the correlation criterion was ZNCC and shape function was zero order for SEM image sets and first-order for real stress tests. We have tested 2 kinds of solvers, first was Newton-Rhapson (baseline for DIC solvers), second one was implementation of Coarse-Fine scheme. While CF does not support higher order shape functions, it can reach quite good result quality even for more complex deformations.

As [Schreier et al., 2000] and [Zhou et al., 2015] mentioned, filtering can improve DIC analysis result quality. They have tested general low-pass filters and also a Wiener filter, which also showed good results. We have selected and tested several other filters to see, how they behave and if they can perform better than the ones in [Zhou et al., 2015]. We have tested following filters:

- Histogram equalization
- CLAHE
- Gaussian
- Binomial
- Wiener
- Lucy-Richardson
- Median
- Bilateral

3.4.2 Results

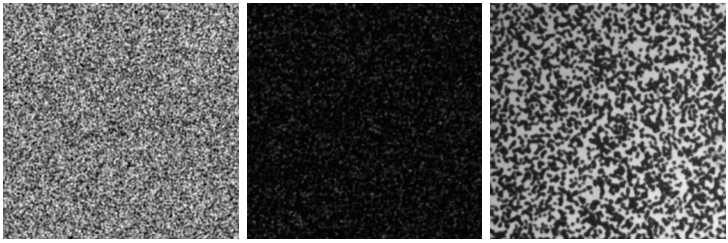
As mentioned in previous chapter, we have multiple combinations of image sets, filters and subset sizes. We will present only a portion of numerical results along with discussion about the behavior of filter in other data combinations and overall filter quality. This section will be divided in three chapters - real data results, SEM data results and discussion about overall filter performance in tests.

SEM data

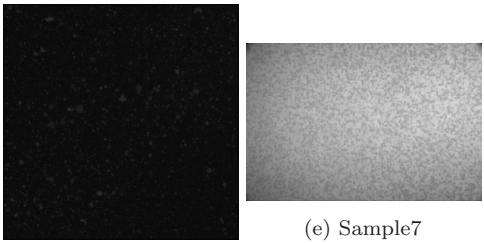
Society for Experimental Mechanics offers test data ([SEM, 2017]) to allow testing and comparison of DIC solvers. We have picked only data sets with shifts and no strain, because it allowed us better result inspection. The data sets we have chosen are following ones: Sample1, Sample2, Sample3, Sample4 and Sample7. The samples we have chosen offer a wide mix of disturbances - noise, contrast, shifts, so the results we obtain will provide good analysis of filter performance in most cases. Figure 3.13 shows tested samples. It can be clearly seen that the data have been artificially generated, the sources are TexGen ([The University of Nottingham, 2017]), FFT shifting and binning. SEM also provides information about image contrast, added noise and shifts for all data sets.

Figure 3.14 presents the error of displacement for SEM data using subset size of 5 (thus the subset is 11 pixels wide and tall). First thing we noticed are quite bad results for Sample 2. By looking into SEM notes about the data, we discovered that Sample 2 is the worst test data set in terms of noise and contrast, thus the results vary so much. There is no clear winner here, all filters provide a change in results precision, but typically they improve results in one direction and worsen the other one. Next image set in terms of image quality is Sample 7. For this set, histogram based filters are the winners. Other filters performed almost as well as unfiltered version. In the rest of the data sets filters performed similarly. Bilateral, Gaussian and Binomial filters performed the best, while the rest slightly worse, only Wiener filter handled Sample 4 data set poorly.

The quality results in figure 3.15 show a little bit different results than displacement error analysis. Some filters work very well in terms on correlation quality, while others not. Bilateral filter performs the best in



(a) Sample1 (b) Sample2 (c) Sample3



(e) Sample7

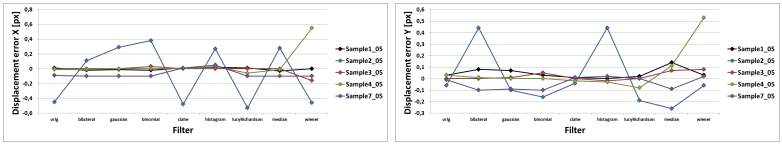
(d) Sample4

Figure 3.13: SEM test data examples

terms of correlation quality, followed closely by Gaussian and Binomial filters. Median filter provides good result quality, but the precision of the results drops significantly (as stated before), so Median filter is a poor choice. Wiener filter is also a good choice, but the selection of good input parameters is not as easy as for other filters.

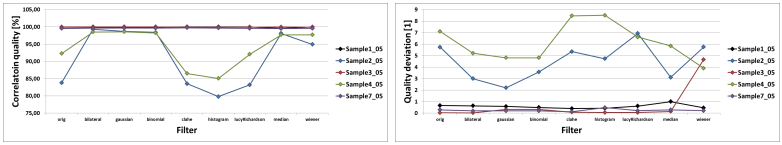
We have also analyzed the deviance of resulting values, as can be seen in Figure 3.15b. The deviance results confirm the findings from absolute value analysis, that Bilateral, Gaussian, Binomial and in most cases Wiener filters improve the quality of results.

Results for three data sets - Samples 1, 3 and 7, are good even for non-filtered image. We have looked in data information from SEM and we have discovered that all three sets have low image noise level.



(a) X direction (b) Y direction

Figure 3.14: Displacement error for SEM data, subset size 5px

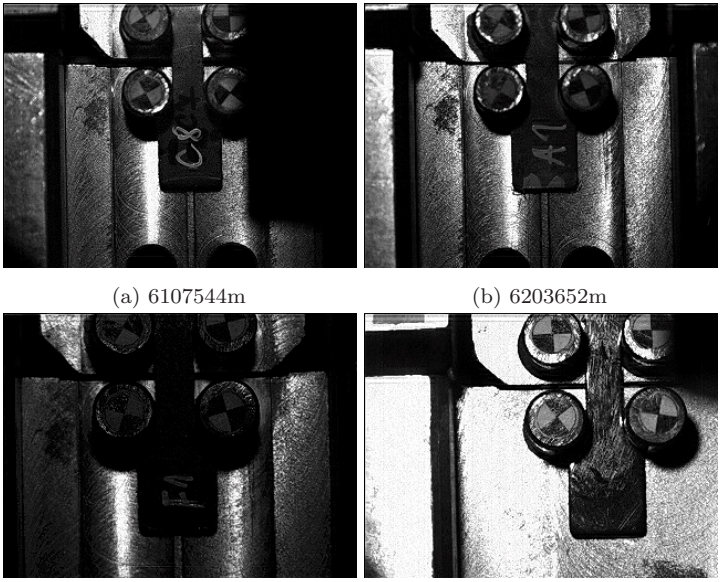


(a) Median value (b) Deviance

Figure 3.15: Correlation quality results for SEM data, subset size 5px

Real world data

While SEM data offer good testing images with variety of errors, but they do not provide same properties as real world data. In real world data all errors are dynamic in general, meaning that for example contrast might vary a lot in different areas of image. This adds another level of problems for solvers and we wanted to see how pre-filtering reacts to real data. Figure 3.16 shows examples from 4 sets of data we have used. The recordings are from mechanical stress test, the specimen is being pulled downwards until it breaks apart. The specimen is made out of layered plastic, so the resulting deformation will be quite complex, including both shifts and shears. This made result analysis a lot harder, because we don't know precise result for each pixel / subset. Because of this, we have decided to analyze only the quality of results via correlation value. In most cases, this approach can describe the performance of the filter, there are cases however, where it fails. Because of this, we have inspected all deformation results and compared them against original analysis to rule out false positives. We have also performed visual inspection of recordings and guessed pixel deformation by eye to rule out further false positives.



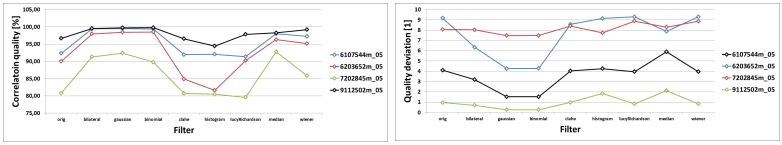
(a) 6107544m (b) 6203652m

(c) 7202845m (d) 9112502m

Figure 3.16: Real world test data examples

The DIC analysis was performed only on the specimen, so the count of subsets is limited compared to SEM data, but in all cases there were at least 10 subsets available for all data configurations.

Figure 3.17 shows the results of our DIC analysis before and after pre-filtering. In total correlation quality results, we can see a clear trend for all filters. Bilateral, Gaussian and Binomial are performing very well for all input images. Median filter also performs well, but after inspecting the displacement results, we came to conclusion, that the results are wrong, thus it is not advisable to use median filter at all. Wiener filter provides some improvement, but it was quite hard to find the best parameter combination. Other filters performed on-par as unfiltered image, sometimes the results were even worse.



(a) Median value (b) Deviance

Figure 3.17: Correlation quality results for real data, subset size 5px

The graph of result quality values deviation (Figure 3.17b) confirms the behavior seen in absolute values, best filters lower the value deviation, while the others keeps it same or make it bigger.

3.4.3 Summary

From out results we can conclude, that pre-filtering can be an effective way to improve DIC result quality, but this has already proved Zhou et al. in [Zhou et al., 2015].

We have extended this test with other types of filters to see, if they can help in some way. Histogram based filters (Histogram equalization and CLAHE filter) showed no improvement. Lucy-Richardson filter performed the worst of all tested filters. The result quality plummeted along with precision. Good in terms of correlation quality, bad in precision was Median filter, so it is not advisable to use it, because the obtained results will be wrong. We have confirmed results from [Zhou et al., 2015] for all 3 filters - Gaussian, Binomial and Wiener. Bilateral filter showed very good results, sometimes outperforming Gaussian filter, but at the cost of computation complexity.

To conclude all our data, it is advisable to use pre-filtering. Gaussian and Binomial filters offer very good performance and quality improvement and it is quite easy to pick correct size of the filter. Wiener and Bilateral filters can offer better quality improvement for heavier noise, but is harder to pick correct filter parameters, because the noise-to-signal ratio must be somehow estimated beforehand.

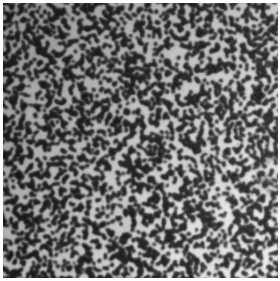
3.5 Automatic subset size

In subset-based DIC, the users must manually select a subset size varying from several pixels to more than a hundred pixels before the DIC analysis. Since the subset size directly determines the area of the subset being used to track the displacements between the reference and target subsets, it is found to be critical to the accuracy of the measured displacements. To achieve a reliable correlation analysis in DIC, the size of a subset should be large enough so that there is a sufficiently distinctive intensity pattern contained in the subset to distinguish itself from other subsets. On the other hand, however, it is noted that the underlying deformation field of a small subset can readily and accurately be approximated by a first-order or second-order subset shape function, whereas a larger subset size normally leads to larger errors in the approximation of the underlying deformations. For this reason, to guarantee a reliable displacement measurement, a small subset size is preferable in DIC. These two conflicting demands imply that there is a trade-off between using large and small subset sizes.

Subset size vs result quality

[Huang et al., 2013] and [Yuan et al., 2014] showed that the value of the correlation criterion can be used as a reliable way of selecting the optimal subset size. Both papers, however, need an extra parameter added to the state vector, thus extending the computation time. This can be tolerable in „offline“ computations, where quality is the main preference, but for „online“ or more complex computation the computation time might play a significant role.

We have performed several tests on data from DIC Challenge dataset supplied by Society for Experimental Mechanics ([SEM, 2017]). We chose dataset „Sample 3“ (Figure 3.18a) and „Sample 10“ (Figure 3.18b) for demonstration in this paper. Sample 3 dataset presents a test case with a constant shift thorough the image, while Sample 10 dataset contains sinusoidal shift. Both datasets resemble real world scenarios by added noise and limited contrast (details on added noise and contrast limitation can be found at the DIC Challenge website).



(a) Dataset "Sample 03"

(b) Dataset "Sample 10"

Figure 3.18: DIC Challenge datasets

3.5.1 Theoretical framework and algorithm

One type

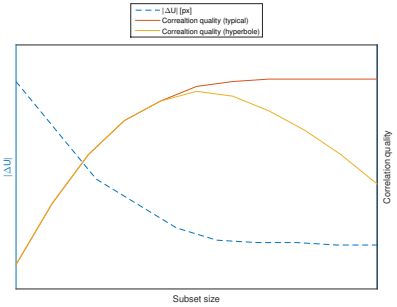


Figure 3.19: Illustration of typical displacement error function of subset size vs correlation quality function

of curve of standard deviation of the displacement error vs. subset size. can be found in [Yaofeng and Pang, 2007] (illustrated red line in Figure 3.19). After reaching the lowest point the function usually starts to grow again, mimicking the hyperbole. Other type of the curve can be found in [Pan et al., 2008] (yellow line in Figure 3.19). The function is shaped like

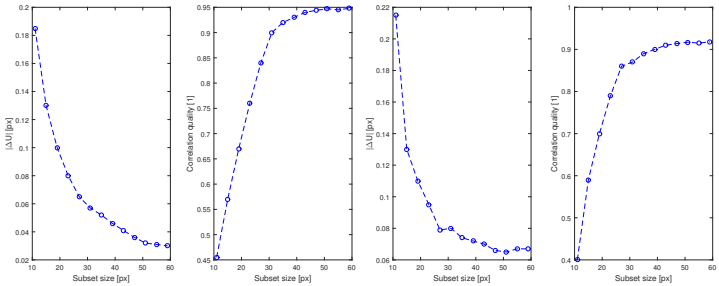
a logarithm function, after quick initial growth the function reached some maximal value and remains constant. If we overlay the displacement error function and correlation quality function (Figure 3.19), we can see that the shapes of both functions are very similar for the log shaped function. And for the hyperbolic shaped function the lowest point is located in the area where the correlation function reaches the maximal value. Using this knowledge, we have designed two algorithms that can estimate the ideal subset size, which will provide most precise results.

Iterative approximation

One way of approximation is to use a line to get an estimate of the optimal value. For first approximation we need at least three points; one for larger subset size to estimate the maximal value of correlation function (y_{max}) and two points for small subset size to approximate the initial growth. New subset size is calculated as the intersection of two lines. The first line is horizontal with y equal to y_{max}, the second line is created from the two points computed for small subset sizes (line equation computation can be seen in the equation 3.1). The value of new subset size is then computed using equation 3.2. Next subset size is always calculated from last two correlation values. We can view the iterative process as an optimization task and thus we can reuse the termination functions used in optimization. One possible condition of termination is defined by required precision, typically expressed as percentage of maximal precision, in Figure 3.21 we have used 95%. The second way to define the termination condition is to define required slope of the initial growth approximation line. Next possibility is to stop the iterative process when the improvement of the precision is low enough.

$$\begin{aligned} y &= m \times x + b \\ m &= \frac{y_2 - y_1}{x_2 - x_1} \\ b &= y - m \times x \end{aligned} \tag{3.1}$$

$$x_{new} = \frac{y_{new} - b}{m} \tag{3.2}$$



(a) Dataset "Sample 3" (b) Dataset "Sample 10"

Figure 3.20: One step approximation displacement error and correlation quality

Spline approximation

The advantage of line approximation is the simplicity. However the precision is limited in general and the noise in the image can adversely affect the resulting value of optimal subset size. More precise would be to use some higher order approximation curve. We have chosen spline approximation, because it provides good accuracy with relatively low computational cost. We used third-order polynomials, resulting in cubic spline interpolation (details can be found in [Bartels et al., 1998]).

3.5.2 Results

In this chapter we will first present the application of approximations on single step of digital image correlation algorithm. Then we will present numerical results comparing the results of classical NR solver with predefined subset size against a NR solver coupled with approximation methods.

Approximation on one step

Figures 3.20a and 3.20b present results (displacement error and correlation quality) obtained by running the DIC algorithm for different subset sizes. We can see that there is a good correlation of functions for correlation

quality and absolute displacement error. The result of iterative approx-

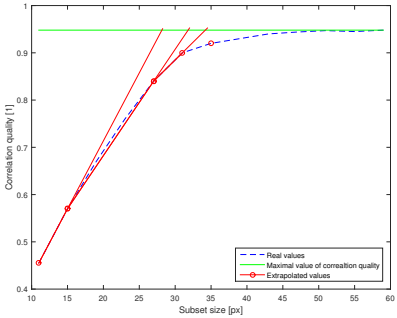
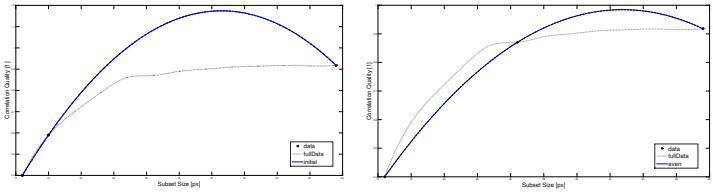


Figure 3.21: Iterative linear approximation

imation can be seen in Figure 3.21. We can see that the estimate is far from optimal, but after just 3 approximation we have already reached a good precision (97% of maximal correlation quality). The implementation of this iterative scheme into a typical digital image correlation engine is quite straightforward, the only requirement is the ability of the engine to compute each round with different subset size. One way of implementation is to repeat the computation for the same image pair over, which will yield the most accurate result. The second way is to run the computation in classic fashion going through the image pairs one after another and change the subset size according to the computed optimal subset size.

Next we ran the test with spline approximation. First we have tried to approximate the function on three points. The results are presented in Figure 3.22. The first sub-figure (3.22a) uses the same data points as the initial step in iterative approximation (two values for smallest subset sizes and one for largest). The dashed line shows the full data for comparison. It is clear that the approximation is very bad and practically unusable. Sub-figure 3.22b presents the approximation with three points evenly spaced (11, 35 and 59 pixels). While the approximation is slightly better, it is still not suitable for practical use.

Spline approximation requires a different approach for point selection. If we provide a value in between the minimal and maximal value, we remove the hill effect (in our case we have chosen the subset size of 31



(a) Same points as iterative approximation

(b) Three evenly spaced points

Figure 3.22: Approximation with cubic spline on three points

pixels). The results of such approximation can be seen in Figure 3.22. The error of the approximation is under 5%, which is enough to allow very precise estimation of optimal subset size.

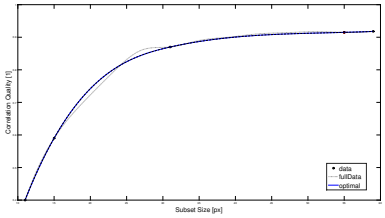
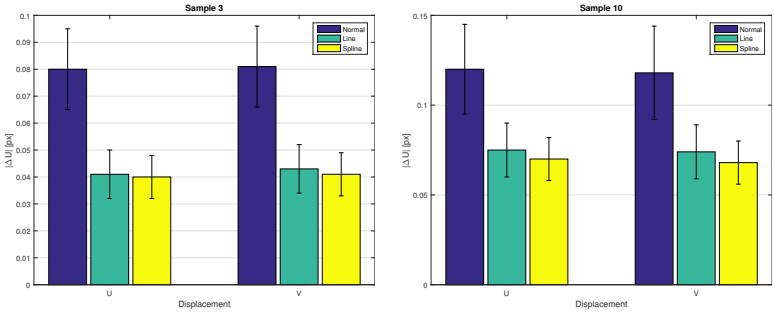


Figure 3.23: Approximation with cubic spline on five points

After computing the interpolated spline, we can use the same constraints as for iterative approach. The only downside of the spline interpolation is the requirement to compute the values in advance before we can use the interpolation, but as we presented the spline approach requires only 5 points in traditional case to achieve high precision. The iterative approach might require only 4 points, but the number of points needed strongly depends on the termination criterion and the exact shape of the function.

3.5.3 Numerical results



(a) Dataset "Sample 3" (b) Dataset "Sample 10"

Figure 3.24: Comparison of statistical errors (mean and standard deviation)

Figures 3.24a and 3.24b present the numerical results. We ran the computation on both datasets using sub-pixel registration algorithm based on ZNSSD criterion with Newton-Raphson solver. First we have used the solver with preset subset sizes, then we repeated the computation with variable subset size controlled by our algorithms. Both figures clearly show good improvement both in absolute error value and also for standard deviation. If we look at Figures 3.20a and 3.20b we can see that the performance of our algorithms is very close to best values possible, but they also haven't reached high values of subset size, thus reaching almost optimal values in terms of precision and noise suppression ability.

3.5.4 Summary

In this chapter, two new subset size selection algorithms have been presented. By means of line approximation the algorithm can predict a proper subset size, which will maximize result precision while keeping the subset size as small as reasonable. The cubic spline interpolation approach presents a more static approach to optimal subset size selection, where we use means cubic spline interpolation on a small set of points to obtain a very good approximation of the correlation quality function shape.

The main contribution is the use of the results of the solver in order to determine the best size, because while the image statistics can provide a good initial estimate, they have no way to take the information about the deformation and solver configuration into account. This way we can balance the influence the systematic and random errors, in order to obtain the most precise result possible for given recording.

Recently, [Hassan et al., 2016, Zhao, 2016] presented similar algorithms that dynamically adjust the window size to obtain the best results possible removing the need of user choosing the size and hoping that it is the best one. Due to recent release date we have not been able to compare our algorithm against the other two.

3.6 GPU-DIC application

As part of the dissertation an application has been developed ([Ječmen, Petr, 2017]). It serves both as testing environment for new research but also should serve for general public to allow easy analysis of deformations happening in the image.

3.6.1 Comparison versus other available systems

We have tested available solutions to compare their speed and precision against our solution. We have downloaded and tested following solutions: NCORR v1.2.1, Optistics MOIRE v0.959, GOM Correlate 2016 and VIC 2D v6. For testing we have picked an image from SEM Test Data Suite called „Sample 3“ (Figure 3.18a). The subset spacing was set to 5 pixels with subset size of 21 x 21 pixels. We ran the test on laptop MSI GE70 with Intel Core i7 4700MQ Haswell, 8GB of RAM and NVIDIA Geforce GTX 765M. The test was executed 5 times in order to filter out background task influence on execution time.

From the table we can see that our software is the fastest one, but also the least precise one. The poor precision of our solution compared to other ones is due to the fact that we were focusing on the raw computation power but didn't give enough attention to the solver. By implementing a better solver our solution could be more precise and perhaps even faster than it is now.

Software	Execution time [ms]	Displacement error [px]
GPU-DIC	38 ± 4	$0,05 \pm 0,014$
NCORR	82 ± 10	$0,01 \pm 0,006$
Opticist	103 ± 10	$0,035 \pm 0,011$
GOM	70 ± 12	$0,008 \pm 0,005$
VIC	56 ± 7	$0,05 \pm 0,006$

Table 3.1: Comparison of performance of various DIC solutions

The evaluation was performed on a limited dataset thus it should not be used as a baseline for picking the right solution. It was performed in order to make a rough estimate on how our software compares to other solutions in general. The detailed comparison should cover far more data configurations and should also inspect the GUI of the solutions, because it is a crucial component of the software from the consumers point of view.

4 Conclusions and future work

Summary

In this work the author is presenting several improvements made to DIC algorithm in order to make it faster and more robust for unexperienced users.

It has been shown that OpenCL offers a good way of improving the performance of the algorithm for all platforms capable of running OpenCL code. First the general assumption of performance gain by implementing the algorithm using OpenCL computing language has been confirmed. The basic idea has been extended by running the test on multiple types of devices (including CPUs, dedicated GPUs and integrated graphics cards) . These tests showed that by using the OpenCL the performance is improved for all types of devices, but each device type requires a bit different computation kernel. Further performance improvement has been achieved by reducing the size of the input data by generating the deformations on GPU.

Next research focused on image preprocessing in order to improve the stability and reliability of the algorithm. Several different approaches for the preprocessing have been tested and it has been shown that the preprocessing can help to improve the result quality, but the improvement comes with reduced performance. The tests also revealed that some algorithms (especially the median filter) may improve the value of the correlation, but the resulting values of deformation are inaccurate at best, so it is not good to use these algorithms for DIC as preprocessors.

Next improvement consists of automatic optimal subset size selection, which also allows the use of DIC algorithm by common user without any knowledge on how to choose optimal subset size to obtain best results possible. The size adjusting algorithm can balance the influence of systematic and random errors in order to reach better results quality.

All mentioned improvements combined can greatly improve the user experience with DIC algorithm. Automatic subset size and improved image preprocessing can prevent the algorithm from computing wrong or inaccurate results after providing images with poor quality and / or using wrong subset size. In cases, where automatic approaches fail, OpenCL implementation can lower the user's frustration after several repeated

computations, because the results are available much faster, so the user does not have to wait so long to see, if the results are correct.

Future works

Main focus of future should be concentrated on improving the precision while not degrading the computation speed. Proposed solution has been tested on integer displacement solver and basic NR solver. Other solvers have different requirements on computed correlations (both in terms of count and locality) so it would be interesting to test how the proposed GPU solution behaves with different solvers.

Another way to improve performance of the algorithm is to reduce the count of subsets. While increasing the spacing between the subset improves the performance, it can seriously hit the precision of the result. Another option of lowering the count of subsets is to choose the ROI as small as possible. In today's software solution, the ROI is selected by user and is usually much larger than needed. Also typically the applications offer basic shapes of ROI while the shape of the specimen could be more complex, which means the user is forced to use the larger ROI. So it could be interesting to design and test an algorithm that analyzes the ROI and searches for area with no movement and mark them as background. The main challenges for the algorithm would be the development of cracks in the specimen and required precision.

Author publications

[1] Josef Novák and Petr Ječmen. Zpracování digitálních záznamů rychlých dějů. souhrnná výzkumná zpráva, Technická univerzita v Liberci, 2014.

[2] Petr Ječmen and Pavel Satrapa. Javaccl - library for simple computation on cluster of workstations. In *QUAERE 2015*, pages 1425 – 1433, Hradec Králové, Česká republika, 2015. Magnanimitas.

[3] P. Ječmen and P. Satrapa. Improving result quality of digital image correlation by image processing. In *Proceedings of the International Scientific Conference on MMK 2015 International Masaryk Conference for Ph.D. Students and young Researchers*, pages 2184 – 2193, Hradec Králové, The Czech Republic, 2015. Magnanimitas.

[4] Petr Ječmen and Pavel Satrapa. Improving speed of digital image correlation algorithm using opencl. In *Proceedings — Research Track of the 4th Biannual CER Comparative European Research Conference*, pages 125 – 129, London, Great Britain, 2015. Sciemcee Publishing, London.

[5] Petr Ječmen and Pavel Satrapa. Reducing memory requirements of digital image correlation algorithm running on gpu. In *Proceedings of the International Scientific Conference on MMK 2015 International Masaryk Conference for Ph.D. Students and young Researchers*, pages 2123 – 2131, Hradec Králové, Česká republika, 2015. Magnanimitas.

[6] Petr Ječmen and Pavel Satrapa. Optimal opencl kernel creation for digital image correlation algorithm. In *CER Comparative European Research 2016 Proceedings*, pages 107 – 111, London, 2016. Sciemcee Publishing.

[7] Petr Jecmen, Frederic Lerasle, and Alhayat Ali Mekonnen. Trade-off between gpgpu based implementations of multi object tracking particle filter. In *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 6: VISAPP, (VISIGRAPP 2017)*, pages 123–131, 2017.

[8] Petr Ječmen and Pavel Satrapa. Real-time optimal subset size selection in digital image correlation. In *CER Comparative European Research 2017 Proceedings*, London, 2017. Sciemcee Publishing.

Bibliography

[Bartels et al., 1998] Bartels, R. H., Beatty, J. C., and Barsky, B. A. (1998). *Hermite and Cubic Spline Interpolation*. Morgan Kaufmann.

[Bing et al., 2006] Bing, P., Hui-Min, X., Bo-Qin, X., and Fu-Long, D. (2006). Performance of sub-pixel registration algorithms in digital image correlation. *Measurement Science and Technology*, 17(6):1615.

[Giachetti, 2000] Giachetti, A. (2000). Matching techniques to compute image motion. *Image and Vision Computing*, 18(3):247–260.

[Goldrein et al., 1995] Goldrein, H. T., Palmer, S. J. P., and Huntley, J. M. (1995). Automated fine grid technique for measurement of large-strain deformation maps. *Optics and Lasers in Engineering*, 23:305–318.

[Hassan et al., 2016] Hassan, G. M., MacNish, C., Dyskin, A., and Shufrin, I. (2016). Digital image correlation with dynamic subset selection. *Optics and Lasers in Engineering*, 84:1–9.

[Huang et al., 2013] Huang, J., Pan, X., Peng, X., Yuan, Y., Xiong, C., Fang, J., and Yuan, F. (2013). Digital image correlation with self-adaptive gaussian windows. *Experimental Mechanics*, 53(3):505–512.

[Ječmen and Satrapa, 2015a] Ječmen, P. and Satrapa, P. (2015a). Improving speed of digital image correlation algorithm using opencl. In *Proceedings — Research Track of the 4th Biannual CER Comparative European Research Conference*, pages 125 – 129, London, Great Britain. Sciemcee Publishing, London.

[Ječmen and Satrapa, 2015b] Ječmen, P. and Satrapa, P. (2015b). Reducing memory requirements of digital image correlation algorithm running on gpu. In *Proceedings of the International Scientific Conference on MMK 2015 International Masaryk Conference for Ph.D. Students and young Researchers*, pages 2123 – 2131, Hradec Králové, Česká republika. Magnanimitas.

[Ječmen, Petr, 2017] Ječmen, Petr (2017). GPU-DIC application. `https://github.com/SirGargamel/GPU-DIC`.

[JogAmp, 2017] JogAmp (2017). Java binding for opencl (jocl). `http://jogamp.org/jocl/www/`. [Online; accessed 2017-05-30].

[Knauss et al., 2003] Knauss, W. G., Chasiotis, I., and Huang, Y. (2003). Mechanical measurements at the micron and nanometer scales. *Mechanics of materials*, 35(3):217–231.

[Lu and Cary, 2000] Lu, H. and Cary, P. D. (2000). Deformation measurements by digital image correlation: Implementation of a second-order displacement gradient. *Experimental Mechanics*, 40(4):393–400.

[Oracle, 2017] Oracle (2017). Java description. `https://docs.oracle.com/javase/tutorial/getStarted/intro/definition.html`. [Online; accessed 2017-05-30].

[Pan et al., 2008] Pan, B., Xie, H., Wang, Z., Qian, K., and Wang, Z. (2008). Study on subset size selection in digital image correlation for speckle patterns. *Opt. Express*, 16(10):7037–7048.

[Press et al., 2007] Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition.

[Rastogi, 2000] Rastogi, P. K. (2000). *Photomechanics (Topics in Applied Physics)*. Springer.

[Schreier et al., 2000] Schreier, H. W., Braasch, J. R., and Sutton, M. A. (2000). Systematic errors in digital image correlation caused by intensity interpolation. *Optical Engineering*, 39(11):2915–2921.

[Schreier and Sutton, 2002] Schreier, H. W. and Sutton, M. A. (2002). Systematic errors in digital image correlation due to undermatched subset shape functions. *Experimental Mechanics*, 42(3):303–310.

[SEM, 2017] SEM (2017). Society for experimental mechanics 2d test image sets. `https://sem.org/2d-test-image-sets/`. [Online; accessed 2017-05-30].

[Sirkis and Lim, 1991] Sirkis, J. S. and Lim, T. J. (1991). Displacement and strain measurement with automated grid methods. *Experimental Mechanics*, 31(4):382–388.

[The University of Nottingham, 2017] The University of Nottingham (2017). Composites research group texgen application. `http://texgen.sourceforge.net/index.php/Main_Page`. [Online; accessed 2017-05-30].

[Tong, 2005] Tong, W. (2005). An evaluation of digital image correlation criteria for strain mapping applications. *Strain*, 41(4):167–175.

[Vendroux and Knauss, 1998] Vendroux, G. and Knauss, W. (1998). Submicron deformation field measurements: Part 1. developing a digital scanning tunneling microscope. *Experimental Mechanics*, 38(1):18–23.

[Yaofeng and Pang, 2007] Yaofeng, S. and Pang, J. H. (2007). Study of optimal subset size in digital image correlation of speckle pattern images. *Optics and Lasers in Engineering*, 45(9):967–974+.

[Yuan et al., 2014] Yuan, Y., Huang, J., Peng, X., Xiong, C., Fang, J., and Yuan, F. (2014). Accurate displacement measurement via a self-adaptive digital image correlation method based on a weighted {ZNSSD} criterion. *Optics and Lasers in Engineering*, 52:75 – 85.

[Zhao, 2016] Zhao, J. (2016). Deformation measurement using digital image correlation by adaptively adjusting the parameters. *Optical Engineering*, 55(12):124104–124104.

[Zhou et al., 2015] Zhou, Y., Sun, C., Song, Y., and Chen, J. (2015). Image pre-filtering for measurement error reduction in digital image correlation. *Optics and Lasers in Engineering*, 65:46–56.