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

Improving digital correlation algorithm for real time use

Dissertation

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

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky a mezioborových studií

Vylepšení algoritmu digitální obrazové korelace pro použití v reálném čase

Disertační práce

Studijní program:	P3901 – Aplikované vědy v inženýrství
Studijní obor:	3901V055 – Aplikované vědy v inženýrství
Autor práce:	**Ing. Petr Ječmen**
Vedoucí práce:	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 $30\times$ 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.

Abstrakt

V této práci je prezentována sada vylepšení algoritmu DIC. Výsledkem těchto vylepšení by měla větší uživatelská přívětivost algoritmu DIC a použitelnost pro běžného uživatele.

První sada vylepšení se zaměřuje na implementaci algoritmu DIC pomocí programovacího jazyka OpenCL. To umožňuje spustit algoritmus na širokém spektru dostupného hardwaru, zejména na GPU. Jak ukazují testy, výpočet DIC na GPU vede k významnému zrychlení (až 30× oproti základní variantě a 10x v porovnání s paralelní variantou). Další vylepšení se zaměřují na optimalizaci velikosti dat s cílem snížit režii přenosu dat z RAM na GPU a studii o tom, jak implementace OpenCL funguje na integrovaných GPU a procesorech.

Další vylepšení se snaží předzpracovat vstupní data tak, aby zlepšila strukturu vzorků, čímž aby se zlepšila kvalita výsledné korelace. Výsledky ukazují zlepšení kvality výsledků, jsou ale vykoupeny zvýšenou dobou výpočtu.

Poslední vylepšení je návrh plně automatického algoritmu, který vybírá nejlepší velikost okna pro dosažení co nejlepšího výsledku. Algoritmus se pokusí nalézt optimální velikost okna pro vyvážení systematických a náhodných chyb sledováním funkční závislosti kvality korelace a velikosti okna.

Acknowledgements

I would like to thank my whole family and especially my girlfriend for the support they provided during my studies, it helped me greatly to reach the goal of finished dissertation.

I also thank my supervisor, doc. RNDr. Pavel Satrapa, Ph.D. for the persistent help and insightful comments.

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

List of Figures

List of Tables

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. While latest improvements in solvers has decreased the time considerably, it is still not good enough. 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. Several algorithms exist that can correct the camera distortion, but not that many works focus on improvement of the texture of the tested specimen, which also has big impact on the result quality.

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 first builds theoretical foundation about DIC algorithm, which is then used to design and implement improvements that will improve the user experience of using DIC algorithm for detailed analysis of deformation and stress. This work is split in four main chapters.

- Chapter 1 sums the motivation of this works along with the goals.
- Chapter 2 contains a detailed introduction to DIC algorithm with theoretical background and references to valid literature
- 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. It was first developed by a group of researchers at the University of South Carolina in the 1980s [Peters and Ranson, 1982, Chu et al., 1985, Sutton et al., 1986] when digital image processing and numerical computing were still in their infancy [Schreier, 2003]. In numerous literature actually the same technique has been given different names, such as digital speckle correlation method (DSCM) [Zhang et al., 1999, Zhou and Goodson, 2001], texture correlation [Bay, 1995], computer-aided speckle interferometry [Chen et al., 1993, Gaudette et al., 2001] and electronic speckle photography [Sjödahl and Benckert, 1993, Sjödahl, 1997]. Besides, it is worth noting that particle image velocity (PIV) [Willert and Gharib, 1991, White et al., 2003], which has been widely used in experimental fluid mechanics for tracking the velocities of particles seeded in fluids, is very similar to DIC in principle and implementation algorithm. Direct applications of PIV for deformation measurement of soil [White et al., 2003] and metal have also been reported.

During the past few years, the DIC method has been extensively investigated and significantly improved for reducing computation complexity, achieving high accuracy deformation measurement and expanding application range. For example, the two-dimensional (2D) DIC method using a single fixed camera is limited to in-plane deformation measurement of the planar object surface. To obtain reliable measurements, some requirements on the measuring system must be met [Sutton et al., 2000, Schreier, 2003]. If the test object is of a curved surface, or three-dimensional (3D) deformation occurs after loading, the 2D DIC method is no longer applicable. To overcome this disadvantage of 2D DIC, 3D DIC based on the principle of binocular stereo-vision [Helm et al., 1996, Garcia et al., 2002] was developed. Besides, the digital volume correlation method, as a direct 3D extension of a 2D DIC method, has also been proposed by Bay and Smith et al [Bay et al., 1999, Smith et al., 2002], which provides the internal deformation of solid objects by tracking the movement of volume unit within digital image volumes of the object. The DVC method emerges as a novel and effective tool for measuring the internal deformation of solid objects.

It should be noted first that both laser speckle patterns [Chen et al., 1993, Sjödahl and Benckert, 1993, Yamaguchi, 1981] and artificial white-light speckle patterns (or more accurately, the random gray intensity pattern of the object surface) have been used as the carrier of surface deformation information in 2D DIC. The laser speckle pattern can be produced by illuminating the optically rough surface with a coherent light source (laser beam). However, a serious decorrelation effect occurs in laser speckle patterns when the test object is subjected to rigid body motion, excessive straining as well as out-of-plane displacement [Brillaud and Lagattu, 2002], which prevents its practicality. In contrast, the white-light speckle 2D DIC is more robust and appealing. Indeed, it can easily be found that most of the current publications with regard to DIC employ white-light speckle patterns, which used a white light source or natural light illumination.

Compared with the interferometric optical techniques used for in-plane deformation measurement, the 2D DIC method has both advantages and disadvantages. For instance, it offers the following special and attractive advantages [Sutton et al., 2000, Shi et al., 2004].

1. *Simple experimental setup and specimen preparation:* only one fixed CCD camera is needed to record the digital images of the test specimen surface before and after deformation. Specimen preparation is unnecessary (if the natural texture of a specimen surface has a random gray intensity distribution) or can simply be made by spraying paints onto the specimen surface.

2. *Low requirements in measurement environment:* 2DDIC does not require a laser source. A white light source or natural light can be used for illumination during loading. Thus, it is suitable for both laboratory and field applications.

3. *Wide range of measurement sensitivity and resolution:* Since the 2D DIC method deals with digital images, thus the digital images recorded by various highspatial resolution digital image acquisition devices can be directly processed

by the 2D DIC method. For example, 2D DIC can be coupled with optical microscopy [Sun et al., 1997, Pitter et al., 2002, Zhang et al., 2006a], laser scanning confocal microscope (LSCM) [Berfield et al., 2006], scanning electron microscopy (SEM) [Kang et al., 2005], atomic force microscopy (AFM) [Chasiotis and Knauss, 2002] and scanning tunneling microscope (STM) [Vendroux and Knauss, 1998] to realize microscale to nanoscale deformation measurement. Similarly, the instantaneous deformation measurement can be realized by analyzing the dynamic sequence of digital images recorded with high-speed digital image recording equipment using the 2D DIC method [Barthelat et al., 2003].

More importantly, with the constant emergence of high-spatial-resolution and high-time-resolution image acquisition equipment, the 2D DIC method can easily be applied to new areas. So, it can be said that the 2D DIC method is one of the current most active optical measurement technologies, and demonstrates increasingly broad application prospects.

Nevertheless, the 2D DIC method also suffers some disadvantages:

1. the tested planar object surface must have a random gray intensity distribution.
2. the measurements depend heavily on the quality of the imaging system.
3. at present, the strain measurement accuracy of the 2D DIC method is lower than that of interferometric techniques, and is not recommended as an effective tool for non-homogeneous small deformation measurement

.

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.

2.1.1 Specimen preparation and image capture

The specimen surface must have a random gray intensity distribution (i.e. the random speckle pattern), which deforms together with the specimen surface as a carrier of deformation information. The speckle pattern can be the natural texture of the specimen surface or artificially made by spraying black and/or white paints, or other techniques. The camera is placed with its optical axis normal to the specimen surface, imaging the planar specimen surface in different loading states onto its sensor plane.

Since the recorded images are the 2D projection of the specimen surface, the estimated motion of each image point multiplying the magnification of the imaging system (in units of mm/pixel) will not accurately equal that of the actual physical point on the specimen surface, unless the following requirements are met.

1. The specimen surface must be flat and remain in the same plane parallel to the CCD sensor target during loading [8, 9]. This implies that the CCD sensor and the object surface should be parallel, and out-of-plane motion of the specimen during loading should be small enough to be neglected. The out-of-plane motion of the specimen leads to a change in magnification of the recorded images, which further yields additional in-plane displacements. Thus, it should be avoided for accurate displacement estimation. Normally, the out-of-plane motion can be somewhat alleviated by using a telecentric imaging system or placing the camera far from the specimen to approximate a telecentric imaging system [Sutton et al., 2000, Sutton et al., 2008].

2. The imaging system should not suffer from geometric distortion. In an actual optical imaging system or other high-resolution imaging system (e.g. SEM, LSCM or AFM), geometric distortion is more or less presented, which impairs the ideal linear correspondence between the physical point and imaged point and produces additional displacements. If the influence of geometric distortion cannot be neglected, corresponding distortion correction techniques should be used to remove the influence of distortion to provide accurate measurements.

2.1.2 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)

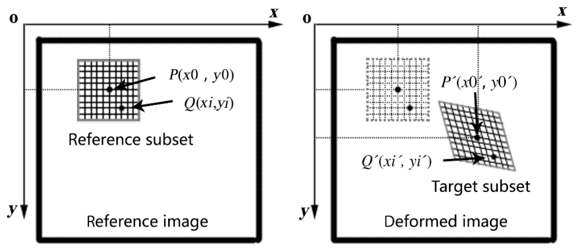


Figure 2.1: Illustration of deformation

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]:

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

If only rigid body translation exists in the reference subset and deformed subset, in other words, the displacements of each point in the subset are the same, then a zero-order shape function can be used:

$$\begin{aligned} \xi_0(x_i, y_j) &= u, \\ \eta_0(x_i, y_j) &= v \end{aligned} \tag{2.2}$$

Obviously, the zero-order shape function is not sufficient to depict the shape change of the deformed subset. Thus, the first-order shape function that allows translation, rotation, shear, normal strains and their combinations of the subset is most commonly used:

$$\begin{aligned} \xi(x_i, y_j) &= u + u_x \Delta x + u_y \Delta y \\ \eta(x_i, y_j) &= v + v_x \Delta x + v_y \Delta y \end{aligned} \tag{2.3}$$

Besides, the second-order shape functions proposed by Lu et al [Lu and Cary, 2000] can be used to depict more complicated deformation states of the deformed subset:

$$\begin{aligned}
\xi(x_i, y_j) &= u + u_x\Delta x + u_y\Delta y + \frac{1}{2}u_{xx}\Delta x^2 + \frac{1}{2}u_{yy}\Delta y^2 + u_{xy}\Delta x\Delta y \\
\eta(x_i, y_j) &= v + v_x\Delta x + v_y\Delta y + \frac{1}{2}v_{xx}\Delta x^2 + \frac{1}{2}v_{yy}\Delta y^2 + v_{xy}\Delta x\Delta y
\end{aligned} \tag{2.4}$$

In equations 2.2 - 2.4, $x = x_i - x_0$, $y = y_j - y_0$, u, v are the x- and y-directional displacement components of the reference subset center, $P(x_0, y_0)$, u_x, u_y, v_x, v_y are the first-order displacement gradients of the reference subset and $u_{xx}, u_{xy}, u_{yy}, v_{xx}, v_{xy}, v_{yy}$ are the second-order displacement gradients of the reference subset.

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] as listed in tables 2.1 and 2.2, respectively.

Cross-correlation (CC)
$C_{CC} = \sum_{i=-M}^{M}\sum_{j=-M}^{M} f(x_i, y_j)g(x'_i, y'_j)$
Normalized cross-correlation (NCC)
$C_{NCC} = \sum_{i=-M}^{M}\sum_{j=-M}^{M} \frac{f(x_i,y_j)g(x'_i,y'_j)}{\bar{f}\bar{g}}$
Zero-normalized cross-correlation (ZNCC)
$C_{ZNCC} = \sum_{i=-M}^{M}\sum_{j=-M}^{M} \frac{[f(x_i,y_j)-f_m][g(x'_i,y'_j)-g_m]}{\Delta f\Delta g}$

Table 2.1: Commonly used cross-correlation criteria

Sum of squared differences (SSD)

$$C_{SSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [f(x_i, y_j) - g(x'_i, y'_j)]^2$$

Normalized sum of squared differences (NSSD)

$$C_{NSSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [\tfrac{f(x_i,y_j)}{\bar{f}} - \tfrac{g(x'_i,y'_j)}{\bar{g}}]^2$$

Zero-normalized sum of squared differences (ZNSSD)

$$C_{ZNSSD} = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [\tfrac{[f(x_i,y_j)-f_m]}{\Delta f} - \tfrac{g(x'_i,y'_j)-g_m}{\Delta g}]^2$$

Table 2.2: Commonly used SSD correlation criteria

In tables 2.1 and 2.2,

$$\begin{aligned}
f_m &= \frac{1}{(2M+1)^2} \times \sum_{i=-M}^{M} \sum_{j=-M}^{M} f(x_i, y_j), \\
g_m &= \frac{1}{(2M+1)^2} \times \sum_{i=-M}^{M} \sum_{j=-M}^{M} g(x'_i, y'_j), \\
\bar{f} &= \sqrt{\sum_{i=-M}^{M} \sum_{j=-M}^{M} [f(x_i, y_j)]^2}, \\
\bar{g} &= \sqrt{\sum_{i=-M}^{M} \sum_{j=-M}^{M} [g(x'_i, y'_j)]^2}, \\
\Delta f &= \sqrt{\sum_{i=-M}^{M} \sum_{j=-M}^{M} [f(x_i, y_j) - f_m]^2}, \\
\Delta g &= \sqrt{\sum_{i=-M}^{M} \sum_{j=-M}^{M} [g(x'_i, y'_j) - g_m]^2}
\end{aligned} \tag{2.5}$$

It is worth noting that the CC criteria are actually related to the SSD criteria. For example, the ZNCC criterion can be deduced from the ZNSSD correlation criterion, the detailed derivation can be found in [Pan et al., 2007] and the following relationship can easily be derived as $C_{ZNSSD}(p) = 2[1 - C_{ZNCC}(p)]$. Similarly, the NSSD criterion can also be deduced from the NCC correlation criterion as $C_{NSSD}(p) = 2[1 - C_{NCC}(p)]$. Also, we should note that if a linear transformation of the target subset gray intensity is made according to the function $g'(x', y') = a \times g(x', y') + b$ [Pan et al., 2007], the correlation values computed using the ZNCC or ZNSSD correlation criterion remain unchanged.

So, it is concluded that the ZNCC or ZNSSD correlation criterion offers the most robust noise-proof performance and is insensitive to the offset and linear scale in illumination lighting. Similarly, the NCC or NSSD correlation criterion is insensitive to the linear scale in illumination lighting but sensitive to offset of the lighting. The CC or SSD correlation criterion is sensitive to all lighting fluctuations.

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 defined in tables 2.1 and 2.2, 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. For this reason, the techniques for an initial guess of deformation are described first in this section.

2.2.1 Initial guess of deformation

Usually, the relative deformation or rotation between the reference subset and the deformed subset is quite small, so it is easy to get accurate estimation of the initial displacements by a simple searching scheme implemented either in spatial domain or in frequency domain. In spatial domain, the accurate locations of the target subset can be determined by a fine search routine, pixel by pixel, performed within the specified range in the deformed image. Some schemes, such as the coarse-to-fine and the nested searching schemes [Zhang et al., 2006b], can be used to speed up the calculation. These search schemes yield 1 pixel resolution. Alternatively, the correlation between the reference subset and the deformed subset can also be implemented in Fourier domain as conducted and advocated by Chen et al [Chen et al., 1993, Sjödahl and Benckert, 1993, Hild et al., 2002]. In Fourier domain, the correlation between two subsets is calculated as the complex multiplication of the first subset's Fourier spectrum by the complex conjugate of the second subset's spectrum. Since the FFT can be implemented with very high speed, the Fourier domain correlation method is also extremely fast. However, the in-plane translation is implicitly assumed in the Fourier domain method, small strains and/or rotations occurring between the two subsets will lead to significant errors.

The above-mentioned technique performs well for most cases. Difficulties occur in certain cases when large rotation and/or large deformation presents between the reference and target subsets, in which some pixels of the reference subset run out of the area of the assumed subset within the deformed image; consequently, the similarity between the reference subset and the assumed deformed subset will decrease substantially. If only rigid body translation exists between the reference and deformed subsets, a single peak can be found in the correlation coefficient distribution. In contrast, while a 20 relative rotation occurs between the reference image and the deformed image, there will not be even a single sharp peak in the correlation distribution map, and thus result in a failure in integer pixel displacement searching.

Other techniques are therefore required to achieve a reliable initial guess of deformation in these cases. Inspired by the nested coarse fine algorithm presented by Zhang et al [Zhang et al., 2006b] that can provide an initial guess for each calculation point, a technique was presented by Pan et al to achieve a reliable initial guess for the NR method for these cases. Slightly different from Zhang's work, this technique only provides the initial guess of the first calculation point. Then, based on the assumption of continuous deformation of solid object, the obtained result of the first point serves as the initial guess for the next point to be calculated. The initial guess of deformation is determined by manually selecting three or more points $(x_i, y_i)(i = 1, 2, ..., n, \text{and } n \geq 3)$ with distinct features around the reference subset center, and their corresponding locations $(x'_i, y'_j)(i = 1, 2, ..., n, \text{and } n \geq 3)$ in the deformed image. Thus, the desired initial guess can be resolved from the coordinate correspondence (depicted by the first-order shape function) using the least-squares method. Alternatively, benefiting from the extraordinary ability of its global optimum, the genetic algorithm [85] can also be used as an automatic technique for determining the initial guess of the first calculation point. However, the genetic algorithm normally costs a lot of computation time to converge to the global extremum.

Calculation path

As described in subsection 2.1.2, before the implementation of DIC analysis, ROI should be specified or defined in the reference image. The regularly spaced points within ROI are considered as points to be computed. Conventional correlation calculation generally starts with the upper-left point of the ROI. Then, the calculation analysis is carried out point by point along each row or column. Normally, to speed calculation and save computation time, the computed displacements and strains of the current point are used as the initial guess of the next point according to the continuous deformation assumption. In this sense, the conventional DIC computation is a path-dependent approach. Although we can estimate the initial guess separately for each point, however, this approach is either impractical because it is extremely time-consuming or impossible if large deformation and/or rotation presents in the deformed image. So, even though the well-established conventional DIC method is effective in most cases, this path-dependent approach may give rise to wrong results in the following cases. First, if the digital images of a practical test object contain discontinuous areas such as cracks, holes or other discontinuous area or if an irregular ROI is defined in the reference image, the transfer of the initial guess will fail to provide a reliable initial guess for the next point at some locations. Second, if apparent discontinuous deformation occurs in the deformed image, the transfer of the initial guess also fails. Third, occasionally occurring wrong data points will also provide a wrong initial guess for the next point. In all these cases, if one point is wrongly computed, the results of wrong points will be passed to the next point, which leads to the propagation of error.

More recently, a universally applicable reliability-guided DIC (RG-DIC) method has been proposed by Pan [Pan, 2009] for reliable image deformation measurement. In the method, the ZNCC coefficient is used to identify the reliability of the point computed. The correlation calculation begins with a seed point and is then guided by the ZNCC coefficient. That means the neighbors of the point with the highest ZNCC coefficient in a queue for the computed points will be processed first. Thus, the calculation path is always along the most reliable direction and possible error propagation of the conventional DIC method can be entirely avoided. The RG-DIC method is very robust and effective. It is universally applicable to the deformation measurement of images with area and/or deformation discontinuities.

2.2.2 Sub-pixel displacement registration algorithms

The techniques described above only provide displacements with pixel level accuracy or approximated initial guess of the deformation vector. To further improve the accuracy of DIC, a certain kind of sub-pixel registration algorithm should be used. In the following, various sub-pixel registration algorithms proposed in the literature are introduced and discussed.

Coarse fine search algorithm

In the determination of integer-pixel displacement, the search method searches the specified searching area of interest with 1 pixel step to find the point with maximum cross-correlation coefficient. It is straightforward to change the search step into 0.1 pixels or 0.01 pixels to achieve a sub-pixel accuracy of 0.1 pixels or 0.01 pixels, respectively [Peters and Ranson, 1982]. Usually the zero-order shape function is implicitly used in the coarse fine searching algorithm while implementing the direct searching scheme. The gray level at sub-pixel locations must be reconstructed in advance using certain interpolation scheme, which generally costs long computation time. Some work [Zhang et al., 2006b] has been done to reduce the computation cost and the probability of misidentification of the traditional coarse fine search algorithm. Nevertheless, compared with the next two algorithms, the coarse fine method is still time-consuming.

Peak-finding algorithm

The peak-finding algorithm refers to a class of algorithms for detecting the peak position of the local discrete correlation coefficient matrix (typically, 3 x 3 or 5 x 5 pixels) around the pixel with maximum CC coefficient or minimum SSD coefficient after the integer-pixel displacement searching scheme. Both least-squares fitting and interpolation algorithms can be used to approximate the local correlation coefficient matrix, and the peak position of the approximated curve surface is taken as the sub-pixel displacements.

For example, Chen et al [Chen et al., 1993] proposed a biparabolic least-squares fitting of the local peak and define the peak position to be the extremum of the obtained polynomial; Sjodahl et al [Sjödahl and Benckert, 1993] used the algorithm by expanding the discrete correlation function in terms of a Fourier series first, followed by a numerical searching scheme to find the exact peak; Hung et al [Hung and Voloshin, 2003] employed a two-dimensional quadratic surface fitting of the peak and the sub-pixel displacements are defined based on the location of the maximum value of the fitting surface; and more recently, Wim et al [Van Paepegem et al., 2009] computed the subpixel displacements by simply calculating the center of mass localization of the discrete correlation matrix. The principle and implementation of peak-finding techniques for sub-pixel displacement measurement is simple, it can be done very fast. However, the peak-finding algorithms do not consider the shape change of the deformed subset, and just compute the approximated peak rather than the actual peak of correlation coefficient; as a consequence, the accuracy is lessened [Bing et al., 2006].

Iterative spatial domain cross-correlation algorithm

If we take the relative deformation (i.e. shape change) between the reference and target subsets into account, the correlation function thus becomes a nonlinear function with respect to the desired mapping parameters vector. For example, if the first-order shape function is used, the desired mapping parameters vector is

$\mathbf{p} = (u, u_x, u_y, v, v_x, v_y)T$. To optimize the nonlinear correlation function, a two-parameter iterative algorithm was first developed by Sutton et al [Bruck et al., 1989]. However, as pointed out in the same paper, the two-parameter iterative algorithm normally does not converge to the exact solutions unless the correlation function is linear with the desired parameters. Naturally, it has been replaced by the more efficient and accurate Newton Raphson (NR) algorithm. As a classic algorithm in 2D DIC, the NR method was also originally presented by Bruck et al [Bruck et al., 1989] and was significantly improved by Vendroux and Knauss [Vendroux and Knauss, 1998] with an adoption of the approximated Hessian matrix. To get the desired deformation vector, using the NR iteration method, the solution can be written as

$$p = p_0 - \frac{\Delta C(p_0)}{\Delta\Delta C(p_0)} \tag{2.6}$$

where p_0 is the initial guess of the solution, which can be provided by the scheme described in the previous section; p is the next iterative approximation solution; $\Delta C(p_0)$ is the gradients of correlation criteria and $\Delta\Delta C(p_0)$ is the second order derivation of correlation criteria, commonly called the Hessian matrix. According to the approach proposed by Vendroux and Knauss [Vendroux and Knauss, 1998], an approximation can be made to the Hessian matrix that can significantly simplify the calculation process of equation 2.6 without affecting the calculation accuracy. Except for the improved NR algorithm, the Levenberg-Marquart algorithm (LM) [Lu and Cary, 2000] and the quasi-NR [Wang and Kang, 2002] were also proposed by some researchers to overcome the shortcoming of the NR algorithm to speed the calculation. Apparently, the iterative spatial domain cross-correlation algorithm is unaffected by large strains and/or rotations of the deformed image, because it is able to take the deformation of a subset into consideration.

Spatial-gradient-based algorithm

Based on the optical flow developed by Davis et al [Davis and Freeman, 1998], an iterative, spatial gradient-based algorithm was proposed by Zhou et al [Zhou and Goodson, 2001] for sub-pixel displacement estimation. The algorithm was further investigated and improved by Zhang et al [Zhang et al., 2003], Pan et al [Bing et al., 2006] and Meng et al [Meng et al., 2007]. More recently, a generalized spatial-gradient-based DIC method has been developed by Pan et al [Pan et al., 2009] to overcome the limitations of the previous algorithms by taking the scale and offset of the intensity into consideration. In the generalized spatial-gradient-based algorithm, the basic optical flow equation for point (x_i, y_i) within the subset is written as

$$\begin{aligned} a \times f(x_i, y_j) + b &= g(x_i + u + u_x\Delta x_i + u_y\Delta y_i, y + v + v_x\Delta xi + v_y\Delta y_i) \\ (i, j &= -M : M) \end{aligned} \tag{2.7}$$

where a is the scale factor of the intensity change and b denotes the offset of the intensity change. Both conventional least-squares and iterative least squares [Pan et al., 2009] can then be used to solve the unknown parameter vector, which contains deformation parameters of the subset, the scale factor and the offset of the intensity change. Calculation using least-squares is exempted from subpixel interpolation and iteration; thus, it can be implemented with very fast computation speed. However, an iterative least-squares algorithm providing higher accuracy in the computed displacements and displacement gradients is highly recommended. Because only the first-order spatial derivatives (i.e. g_x, g_y) of the deformed image are required during calculation, thus, it is little simpler than the classic NR method. However, the spatial-gradient-based method is actually equivalent to the optimization of the following SSD correlation criterion using an improved NR algorithm as proved by Pan et al [Pan et al., 2009]:

$$C_{SSD}(\mathbf{p}) = \sum_{i=-M}^{M} \sum_{j=-M}^{M} [g(x_i + u(x_i, y_j), y_j + v(x_{i,y} j)) - a \times f(x_i, y_j) - b]^2, \quad (2.8)$$

where $\mathbf{p}$ denotes the desired parameter vector.

Genetic algorithms

As a random search algorithm with robust global converge capability, the genetic algorithm is widely used to optimize the multi-dimensional nonlinear function, and it was also introduced into the 2D DIC method to optimize the correlation criterion for determination of deformation parameters [Mahajan, 2004]. To detect a global optimum deformation vector for the objective function (i.e. correlation criterion), an initial population containing N candidate individuals (i.e. deformation parameter vector) is randomly generated within its possible range, and then the objective function cost function is evaluated for all the candidate individuals. The best individuals are kept and the worst discarded, and some new individuals are also randomly generated to replenish the population. Subsequently, some schemes, such as crossover, mutation and recombination, are used to generate a new population from the last population and the objective function is evaluated again for all the candidate individuals. These processes are repeated until the converging conditions are satisfied. Although with extraordinary ability of global optimum, the genetic DIC method generally requires a long computation time to find the global extremum of the correlation coefficient.

Finite element method and B-spline algorithm

The above-mentioned algorithms determine the displacements for each subset center independently and can be considered as a point-wise algorithm. Quite different from these point-wise algorithms, a continuum method (or a global method), in which the surface deformation of the test object throughout the entire image area was represented by the B-spline function, was proposed by Cheng et al [Cheng et al., 2002]. In addition, the so called Q4-DIC developed by Besnard et al [Besnard et al., 2006] and the finite element formulation proposed by Sun et al [Sun et al., 2005] can also be categorized into a global method. These techniques ensure the displacement continuity and displacement gradients continuity among calculation points. Thus, they declared that the mismatching of points can be avoided. However, from the reported results, these two methods seem not to provide higher displacement measurement accuracy than the NR algorithm. Although various algorithms can be used to achieve subpixel displacement, it is worth noting that the most widely used two algorithms in various literature are the iterative spatial domain cross-correlation algorithm and the peak-finding algorithms, due to their simplicity. However, after evaluating their displacement measurement accuracy and precision using computer simulated speckle images with controlled deformation, the iterative spatial domain cross-correlation algorithm with higher accuracy, best stability and broader applicability is highly recommended for practical use [Bing et al., 2006].

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.

It should be noted first that the error of estimated displacement gradients using the NR or genetic method normally limits its use only to local strains greater than approximately 0.010 [Bruck et al., 1989]. Besides, although the relationship between the strain and displacement can be described as a numerical differentiation process in mathematical theory, unfortunately, the numerical differentiation is considered as an unstable and risky operation [Luo et al., 2005], because it can amplify the noise contained in the computed displacement. Therefore, the resultant strains are untrustworthy if they are calculated by directly differentiating the estimated noisy displacements. For example, if the error of displacement estimation is estimated as ± 0.02 pixels and the grid step is 5 pixels, then the error of resultant strain calculated by the forward difference is $\Delta\epsilon = (|\pm 0.02| + |\pm 0.02|)/5 = 8000$, while that calculated by the central difference is $\Delta\epsilon = (|\pm 0.02| + |\pm 0.02|)/10 = 4000$. An error to that extent will probably hide the underlying strain information of the tested specimen and is unbearable in most cases.

So, it is believed that the accuracy of strain estimation will be improved by smoothing the computed displacement fields first and subsequently differentiating them to calculate strains. Based on these considerations, Sutton et al [Sutton et al., 1991] proposed a technique that involves smoothing the computed displacement fields with the penalty finite element method first and subsequently differentiating them to calculate strains. This technique was utilized by Shi et al [Shi et al., 2004] to compute the thermal deformation of electronic packaging. Recently, Meng et al [Meng et al., 2007] further improved the FEM smoothing technique. In addition, the thin plate spline smoothing technique and three other smoothing algorithms were also introduced by Wang et al [Wang et al., 2002] to remove the noises contained in displacement fields. Because the noise level contained in the displacement field is significantly decreased after smoothing operation, these techniques substantially increase the precision in resulting strain estimation. However, smoothing noisy discrete data using the penalty finite element method or thin plate spine is quite cumbersome. More recently, to extract the local deformation technique with the PLC band, Xiang et al [Xiang et al., 2007] used the moving least-squares (MLS) [Lancaster and Salkauskas, 1981] method to smooth the displacement field followed by a numerical differentiation of the smoothed displacement field to get the strain fields. It is necessary to note that the MLS is a weighted least-squares method for reconstructing continuous functions from a set of unorganized points, and is little different from the following pointwise local least-squares fitting method.

The more practical technique for strain estimation is the pointwise local least-squares fitting technique used and advocated by Wattrisse et al [Wattrisse et al., 2001]. In order to analyze the strain localization phenomena that occur during the tension of thin flat steel samples, Wattrisse et al [Wattrisse et al., 2001] implemented a local least-squares method to estimate the strains from the discrete and noisy displacement fields computed by the DIC method. To obtain the strains of the points located at calculation boundaries, a continuity extension of the displacement field is performed at the image boundary. A similar technique was also used by Pan et al [Pan et al., 2009] with a simpler and more effective data processing technique for the calculation of strains for the points located at the image boundary, hole, cracks and the other discontinuity area.

The implementation of the local least-squares fitting technique for strain estimation can be explained as follows. If we want to compute the strains of the current point, we first select a square window containing (2m + 1) x (2m + 1) discrete points (i.e. strain calculation window) around it. If the strain calculation window is small enough, the displacement distributions in it can be approximated as a linear plane; thus, we have

$$\begin{aligned} u(i,j) &= a_0 + a_1 x + a_2 y \\ v(i,j) &= b_0 + b_1 x + b_2 y \end{aligned} \tag{2.9}$$

, where $i, j = -m : m$ are the local coordinates within the strain calculation window, $u(i,j)$ and $v(i,j)$ are the original displacements at location (i, j) obtained by DIC, and $a_{i=0,1,2}, b_{i=0,1,2}$ are the unknown polynomial coefficients to be determined. Equation 2.9 can be rewritten in matrix form. For the first formulation of equation 2.9, it can be rewritten as

$$\begin{bmatrix} 1 & -m & -m \\ 1 & -m+1 & -m \\ \vdots & \vdots & \vdots \\ 1 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 1 & m-1 & m \\ 1 & m & m \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} = \begin{bmatrix} u(-m,-m) \\ u(-m+1,-m) \\ \vdots \\ u(0,0) \\ \vdots \\ u(m-1,m) \\ u(m,m) \end{bmatrix} \tag{2.10}$$

Thus, the least-squares method can be used to solve the unknown polynomial coefficients. It is important to note that, for the points located at the image boundary or in the vicinity of discontinuity area, the strain calculation window around it may contain less than $(2m+1) \times (2m+1)$ points. However, we can still compute strain components using least-squares fitting by simply neglecting these invalid points within the local strain calculation window from equation 2.10 [Meng et al., 2007].

Therefore, the desired Cauchy strains or Green strains at the center point of a local sub-region can be computed based on the obtained coefficients $a_{i=0,1,2}, b_{i=0,1,2}$. Since the noises can be largely removed in the process of local fitting, thus the accuracy of the calculated strains is greatly improved. However, it is quite important to note that, to obtain reasonable and accurate full-field strain estimation, the following two aspects are important and should be considered. One is the accuracy of displacement fields obtained by DIC, and the other is the size of the local strain calculation window used for fitting. For homogeneous deformation, a large strain calculation window is preferred. However, for inhomogeneous deformation, a proper size of strain calculation window should be selected with care to get a balance between strain, accuracy and smoothness. A small strain calculation window cannot suppress the noise of the displacements, while a large strain calculation window may lead to the unreasonable linear approximation of deformation within the strain calculation window.

2.4 GPU computing

Historical beginning of modern graphics processing units date back to mid eighties, when Amiga Corporation released their first computer featuring a device that would be nowadays recognized as a full graphics accelerator. Prior to this turning point, all the computers generated the graphics content on central processing unit (CPU). Offloading the computation of graphics to a dedicated device allowed higher specialization of the hardware and relieved the computational requirements on the CPU. By 1995, replaceable graphics cards with fixed-function accelerators surpassed expensive general-purpose co-processors, which have completely faded away from the market in next few years.

Large number of manufacturers and increasing demand for hardware-accelerated 3D graphics gave led to an establishment of two application programming interface (API) standards named OpenGL and DirectX. Whereas the first one did not restrict its usage to a particular hardware and benefited from cutting-edge technologies of individual card series, the latter was usually one step behind due to its strict marketing policy targeting only a subset of the vendors. Nevertheless, the difference was wiped as Microsoft started to work closely with the GPU developers reaching a widespread adoption of its DirectX 5.0 in gaming market.

After 2001, GPU manufacturers enhanced the accelerators by adding support for programmable shading, greatly allowing game developers and designers to adjust rendering algorithms to produce customized results. Such architecture enabled first

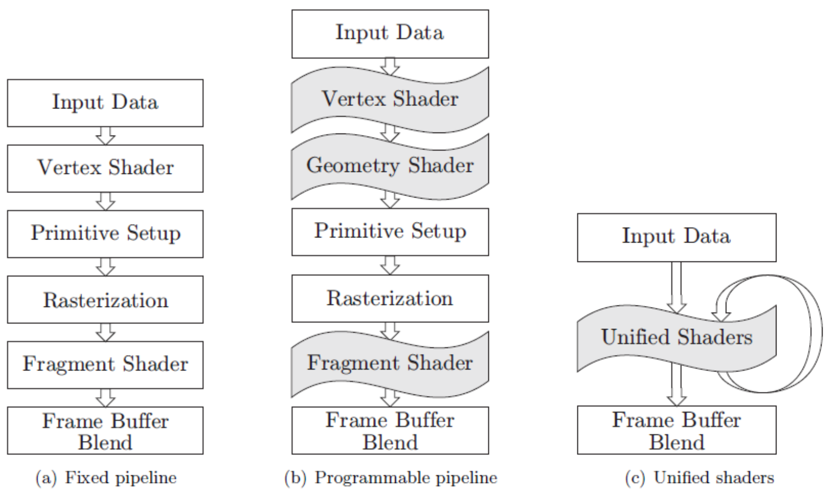


Figure 2.2: Evolution of GPU pipeline

attempts to exploit the graphics dedicated hardware for computing non-graphical tasks. To classify the degree of programmability, the cards were categorized according to supported shading model. The initial Shader Model 1 considerably limited general purpose computation, since the GPUs were equipped with a low number of registers and the rendered image was always stored directly to the frame-buffer, Shader Model 2 represented a notable progress due to its support of floating point numbers, rendering to textures and increased number of registers and code instructions. Subsequent shading model enabled cycles and conditional code execution. Contemporary Shader Model 4 replaced the fixed functionality for assembling geometric primitives by a programmable geometry shader, which allows to create vertices and geometrical primitives directly on the GPU. Graphics pipeline of Shader Model 4 is shown in Figure 2.2 (b). Over the last ten years, the graphics hardware evolved back from specialized multi-core processing units into many-core unified multiprocessors with dynamic workload distribution.

Traditional Graphics Pipeline

Fundamental part of any graphics accelerating hardware is the graphics pipeline. Its concept was imported into the graphics cards from gaming consoles. Due to workload optimization the pipeline is divided into a number processing stages. After the geometric primitives, typically points, triangles, or polygons are passed to the pipeline, the vertex shader performs necessary transformations, applies per-vertex lighting, and modifies parameters of vertices. Modern graphics cards are equipped with a geometry shader, which is able to change the topology of the scene, e.g. generate a new triangle from each vertex. Graphics cards released prior 2007 were not equipped with this shader and the primitives from vertex shader were directly passed to the rasterizer that turns the 3D representation into a set of fragments that are subsequently processed in the fragment shader. The large number of fragments per rendered frame implied high requirements on the throughput of the fragment shader, hence the number of processing units was usually several times greater than in the vertex shader.

Programmable Graphics Accelerators

Endless fantasy of game designers and artists led developers of graphics accelerators to abandon the idea of fixed functionality of the shaders and allow a certain degree of programmability. Starting with the fragment shader, the individual stages of the pipeline were open to the programmers that could modify the rendering pipeline by uploading their own shading programs on the GPU. The architecture of the accelerators was evolving with the previously mentioned shading standards and gradually enabled higher degree of programmability leading to development of advanced rendering techniques such as rendering to vertex buffer and multi-pass method.

The greatest revolution in the design of graphics cards came in 2007, when both market leading vendors NVIDIA and ATI, dismissed the idea of separate specialized shaders and replaced them with a single set of unified processing units. Previously, all the shaders in the pipeline were composed of a number of vector processors that were designed for efficient execution of four-element vector instructions. With cycles and conditional code execution, the GPUs were suddenly required to handle more scalar instructions. Therefore, the accelerators were enhanced by the ability to switch to "3+1" or "2+2" mode of execution, which allowed to process one three-element vector and one scalar instruction, or dual execution of two-element instructions at the time. Nevertheless, the complicated process of scheduling and coupling of instructions led hardware engineers to accept the simplified design of unified shaders, which however does not yield lower performance than the sophisticated vector design. Furthermore, the unified architecture allows less complicated hardware design, which can be manufactured with shorter and faster silicon technology. Rendering of graphics is carried with respect to the traditional graphics pipeline, where the GPU consecutively utilizes the set of processing units for vertex operations, geometry processing, rasterization, and fragment shading, as shown in Figure 2.2.

General Purpose Computing on GPU

In addition to advanced rendering techniques, the programmability of graphics accelerators allowed to exploit the massive parallel power of GPUs for non-graphical tasks such as medical and scientic simulations. One of the best known projects from this category is the distributed simulation of protein synthesis from Stanford University called Folding at Home [FEV+09]. The authors implemented several clients targeting the major graphics accelerators and the project has spread all around the world occupying hundreds of thousands personal computers nowadays. The combination of distributed computing and massive power of GPUs provides results that would never be achievable with the traditional sequential style of programming.

Porting of general computation tasks on a GPU requires careful analysis and compliance with a number of rules. First, the algorithm must be easily parallelizable to fully exploit the benefits of streaming computation. If this condition is met, the algorithm is ported on the GPU using a shading language such as GLSL [Rost, 2005], HLSL [McCool et al., 2002]. Since the primary target is processing of graphics and shading, programs written in these languages tightly follow the graphics pipeline and require the programmer to handle the data as vertices and fragments while storing them in textures. To hide the architecture of the underlying hardware, various research groups created languages for general computation on GPUs and multi-core CPUs. Among the most popular belongs [Buck et al., 2004] that yielded success mostly in other areas than computer graphics.

2.4.1 CUDA

Compute Unified Device Architecture (CUDA) [NVIDIA Corporation, 2007] is a new hardware and software solution for data-intensive computing on NVIDIA GPUs. Completely built from ground up, CUDA represents a new generation of graphics cards. The biggest difference from the previous series resides in simplified architecture, which however allows the processors to run on higher clock speed and breaks down the traditional concepts of general purpose programming on GPUs. Programs running in CUDA suer less from the overhead of graphics oriented pipeline. The language enables techniques such as data scattering, inter-thread communication and synchronization, atomic instructions, and a few features typical for high-level programming languages. The nvcc compiler allows to build the application for emulated execution on CPU, which enables easy debugging that was impossible up to now.

Programming Model

CUDA programming model is designed to fully expose parallel capabilities of NVIDIA GPUs. Even though the language is devoted to general purpose computing, it still requires the programmer to follow a set of paradigms arising from the GPU architecture.

Host and Device

In spite of the fact that all the demanding computation is carried by the GPU, the CPU is required to assist and provide the management and control over the execution. The programmer specifies the order of calling GPU routines, which is then executed by the CPU, called host. GPU serves as a separate device obeying the predefined workflow, in which it processes streaming data. CUDA assumes that both the host and the device are equipped with their own memory, referring to them as to the host memory and device memory. The responsibility for organizing data transfers is left on host, which also calls appropriate functions for allocating and deallocating device memory.

Kernel Functions

The heart of each CUDA program is represented by a set of kernel functions (kernels) that are responsible for simultaneous processing of data streams. Kernels are executed on the device, but initiated by the host, which also specifies starting configuration of the kernel.

2.4.2 OpenCL

OpenCL is a programming framework and standard set from Khronos group ([Khronos group, 2017a]) for heterogeneous parallel computing on cross-vendor and cross-platform hardware. It provides a top level abstraction for low level hardware routines as well as consistent memory and execution models for dealing with massively-parallel code execution. The advantage of this abstraction layer is the ability to scale code from simple embedded microcontrolers to general purpose CPUs from Intel and AMD, up to massively-parallel GPGPU hardware pipelines, all without reworking code. While the OpenCL standard allows OpenCL code to execute on CPU devices, this paper will focus specifically on using OpenCL with Nvidia and ATI graphics cards as this represents (in the authors opinion) the pinnacle of consumer-level high-performance computing in terms of raw FLOPS throughput, and has significant potential for accelerating "suitable" parallel algorithms.

The specific definition of compute units is different depending on the hardware vendor. In AMD hardware, each compute unit contains numerous "stream cores" (or sometimes called SIMD Engines) which then contain individual processing elements. The stream cores are each executing VLIW (Very Long Instruction Word) 4 or 5 wide SIMD instructions. In NVIDIA hardware they call compute units "stream multiprocessors" (SM‘s) (and in some of their documentation they are refereed to as "CUDA cores"). In either case, the take away is that there is a fairly complex hardware hierarchy capable of executing at the lowest level SIMD VLIW instructions.

An important caveat to keep in mind is that the marketing numbers for core count for NVIDIA and ATI aren‘t always a good representation of the capabilities of the hardware. For instance, on NVIDIA‘s website a Quadro 2000 graphics card has 192 "Cuda Cores". However, we can query the lower-level hardware capabilities using the OpenCL API and what we find is that in reality there are actually 4 compute units, all consisting of 12 stream multiprocessors, and each stream multiprocessor is capable of 4-wide SIMD ($192 = 4 \times 12 \times 4$). Similarly, the marketing documentation for a HD6970 shows 1536 processing elements, while in reality the hardware has 24 compute units (SIMD engines), and 16 groups of 4-wide processing elements per compute unit ($1536 = 24 \times 16 \times 4$).

OpenCL Execution Models

At the top level the OpenCL host uses the OpenCL API platform layer to query and select compute devices, submit work to these devices and manage the workload across compute contexts and workqueues. In contrast, at the lower end of the execution hierarchy (and at the heart of all OpenCL code) are OpenCL "Kernels"

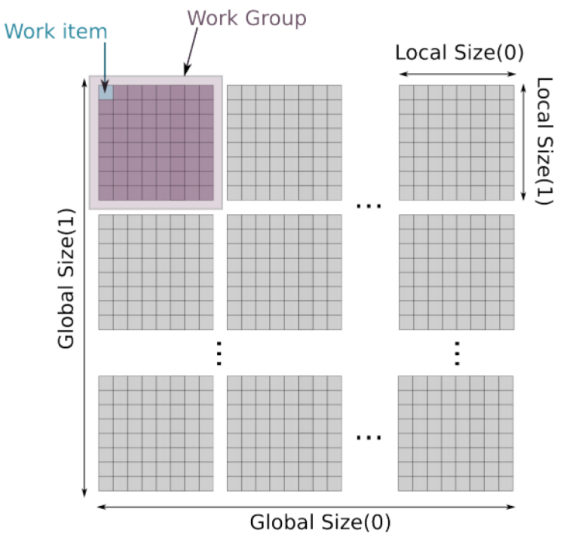


Figure 2.3: 2D Data-Parallel execution in OpenCL

running on the each processing element. These Kernels are written in OpenCL C (a subset of C99 with appropriate language additions) that execute in parallel over a predefined N-dimensional computation domain. In OpenCL vernacular, each independent element of execution in this domain is called a "work-item" (which NVIDIA refers to as "CUDA threads"). These work-items are grouped together into independent "work-groups" (which NVIDIA refers to as a "thread block"). See Figure 2.3 for a top level overview of this structure. According to the documentation, the execution model is "fine-grained" data parallelism and thread parallelism, nested within coarse-grained data parallelism and task parallelism" [NVIDIA, 2017]. Data-parallel programming is where the domain of execution for each thread is defined by some region over a data structure or memory object (typically a range of indices into an N-by-N array as depicted by Figure 3), where execution over these sub-regions are deemed independent.

The alternative model is task-parallel programming, whereby concurrency is exploited across domains of task level parallelism. OpenCL API exploits both of these, however since access to global memory is slow one must be careful in writing Kernel code that reflects the memory access performances of certain memory locations in the hierarchy (more on memory hierarchy later). In this way work-groups can be separated by task-parallel programming (since threads within a work-group can share local memory), but are more likely sub-domains in some larger data structure as this benefits hardware memory access (since getting data from DRAM to global GPU memory is slow, as is getting data from global GPU memory to local work-group memory).

Since hundreds of threads are executed concurrently which results in a linear scaling in instruction IO bandwidth, NVIDIA uses a SIMT (Single-Instruction, Multiple-Thread) architecture. One instruction call in this architecture executes identical code in parallel by different threads and each thread executes the code with different data. Such a scheme reduces IO bandwidth and allows for more compact thread execution logic. ATI‘s architecture follows a very similar model (although the nomenclature is different).

With the framework described above, we can now outline the basic pipeline for a GPGPU OpenCL application.

1. Firstly, a CPU host defines an N-dimensional computation domain over some region of DRAM memory. Every index of this N-dimensional computation domain will be a work-item and each work-item executes the same Kernel.
2. The host then defines a grouping of these work-items into work-groups. Each work-item in the work-groups will execute concurrently within a compute unit (NVIDIA streaming multiprocessor or ATI SIMD engines) and will share some local memory (more later). These work-groups are placed onto a work-queue.
3. The hardware will then load DRAM memory into the global GPU RAM and execute each work-group on the work-queue.
4. On NVIDIA hardware the multiprocessor will execute 32 threads at once (which they call a "warp group"), if the workgroup contains more threads than this they will be serialized, which has obvious implications on the consistency of local memory.

Each processing element executes purely sequential code. There is no branch prediction and no speculative execution, so that all instructions in a thread are executed in order. Furthermore, some conditional branch code will actually require execution of both branch paths, which are then data-multiplexed to produce a final result. I will refer the reader to the Khronos OpenCL, ATI and NVIDIA documentations for further details since the details are often complicated. For instance, a "warp" in NVIDIA hardware executes only one common instruction at a time on all threads in the work-group (since access to individual threads is through global SIMT instructions), so full efficiency is only realized when all 32 threads in the warp agree on their execution path.

There are some important limitations on work-groups to always keep in mind. Firstly, the global work size must be a multiple of the work-group size, or another way of saying that is that the work-groups must fit evenly into the entire data structure. Secondly, the work-group size (which of a 2D array would be the $size^2$) must be less than or equal to the CL KERNEL WORK GROUP SIZE constant. This is a hardware constant stating the limitation on the maximum concurrent threads within a work-group. OpenCL will return an error code if either of these conditions is violated (in general, not checking the return conditions for all the API functions will either cause the host program to crash or crash the OS.).

OpenCL Memory Model

The OpenCL memory hierarchy (shown in Figure 2.4) is structured in order to "loosely" resemble the physical memory configurations in ATI and NVIDIA hardware.

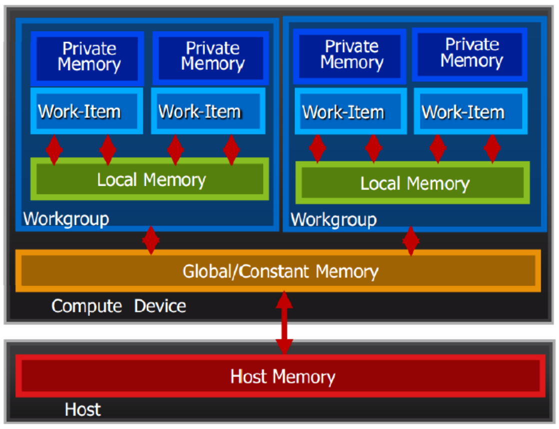


Figure 2.4: OpenCL Memory Model

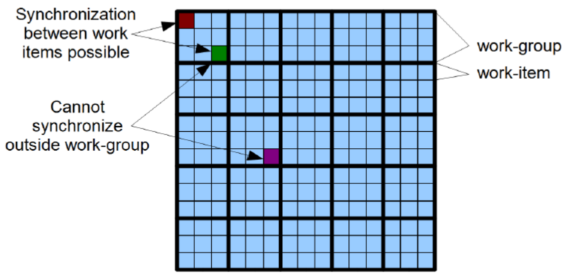

Synchronization between work items possible
Cannot synchronize outside work-group
work-group
work-item

Figure 2.5: OpenCL Work-group / Work-unit structure

The mapping is not 1 to 1 since NVIDIA and ATI define their memory hierarchies differently. However the basic structure of top global memory vs local memory per work-group is consistent across both platforms. Furthermore, the lowest level execution unit has a small private memory space for program registers. These work-groups can communicate through shared memory and synchronization primitives, however their memory access is independent of other work-groups (as depicted in Figure 2.5). This is essentially a data-parallel execution model, where the domain of independent execution units is closely tied and defined by the underlining memory access patterns. For these groups, OpenCL implements a relaxed consistency, shared memory model. There are exceptions, and some compute devices (notably CPUs) can execute task-parallel compute Kernels, however the bulk of OpenCL applications on GPGPU hardware will execute strictly data-parallel workers. An important issue to keep in mind when programming OpenCL Kernels is that memory access on the DRAM global and local memory blocks is not protected in any way. This means that segfaults are not reported when work-items dereference memory outside their own global storage. As a result, GPU memory set aside for the OS can be clobbered unintentionally, which can result in behaviors ranging from benign screen flickering up to frustrating blue screens of death and OS level crashes.

Another important issue is that mode-switches may result in GPU memory allocated to OpenCL to be cannibalized by the operating system. Typically the OS allocates some portion of the GPU memory to the "primary-surface", which is a frame buffer store for the rendering of the OS. If the resolution is changed during OpenCL execution, and the size of this primary-surface needs to grow, it will use OpenCL memory space to do so. Luckily these events are caught at the driver level and will cause any call to the OpenCL runtime to fail and return an invalid context error.

Memory fences are possible within threads in a work-group as well as synchronization barriers for threads at the work-item level (between individual threads in a processing element) as well as at the work-group level (for coarse synchronization between workgroups). On the host side, blocking API functions can perform waits for certain events to complete, such as all events in the queue to finish, specific events to finish, etc. Using this coarse event control the host can decide to run work in parallel across different devices or sequentially, depending on how markers are placed in the work-queue.

2.5 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 2.1 and 2.2, 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.

2.5.1 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.

2.5.2 Parallelism on CPU

Most basic mean of running the computation in parallel on CPU is to use threads. Modern processors have multiple cores (ranging from 2 to 16) and in ideal case each core computes one thread independently from other cores and threads. In real cases threads are sharing some data and need to wait for other threads to finish the work before they can utilize shared data. In most cases, the computation work-flow is to split the task into multiple sub-tasks, run sub-tasks on different cores using threads, wait until all threads are done and condense the results from threads in one final result.

2.5.3 Parallelism on GPU

There has been some success in DIC computation using GPU for computation. Initial test concerning the speedup of correlation computation using CUDA can be found in [Gembris et al., 2011]. The results are promising, the gain is usually around factor 15, which might be good, but the paper uses single thread CPU computation as reference. The paper [Beata et al., 2012] focuses on digital image correlation (as opposed to general correlation in [Gembris et al., 2011]). The authors used OpenCV library to achieve computation on GPU and their results showed average improvement by factor of 10. Further speedup has been achieved by better utilizing GPU using simultaneous computation of products sums and also utilizing texture cache. Another way to achieve GPU computation using CUDA is to use MATLAB coupled with Jacket software, which is able to run MATLAB code on GPU. A test case using FFT variant of DIC algorithm can be found in [Singh and Omkar, 2013]. The results clearly show that GPU computing provides a noticeable speedup, but performance of general solution like Jacket might vary a lot according to tested algorithm and input data size and organization.

One of the first papers mentioning GPU implementation of DIC algorithm was [Leclerc et al., 2009]. The description of the conversion is very vague, but they develop a finite element based correlation technique and use an in-house code generator to generate C++, x86, x64 and CUDA code. They reach an approximate performance of one processed image per second. [Beata et al., 2012] presents a more detailed study on CUDA implementation of DIC algorithm. They use cross correlation for matching computed directly (without Fourier transformation) and reach performance of approximately 5 images per second (with images slightly larger than the ones used in [Leclerc et al., 2009]).

[Singh and Omkar, 2013] test the FFT implementation of DIC algorithm on GPU using CUDA. They use MATLAB to implement the algorithm itself and then the AccerlerEyes (current name of the company is ArrayFire) Jacket (details can be found at [ArrayFire, 2017], nowadays the Jacket is implemented inside MATLAB in Parallel Computing Toolbox). It allows to choose if the code is run on CPU or GPU. They report an average speedup of 3, but they reached a speedup of 6 for repetitive tasks. Further investigation on improving the speed of FFT based DIC is presented in [Zhang et al., 2015]. They combine FFT based cross correlation with inverse compositional Gauss–Newton algorithm for sub-pixel registration. While comparing the performance of pure sequential solution to pure GPU implementation, the speedup of 60 is very good and they also report no loss of precision, which is also very important due to preference of precision over speed in todays use of DIC algorithm.

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.

Each section will present one step in our optimization process. It will contain description what has been changed along with discussion on how to choose the best input parameter (thread count or local work size). Graph with numerical results can be found in Figure 3.2.

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 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

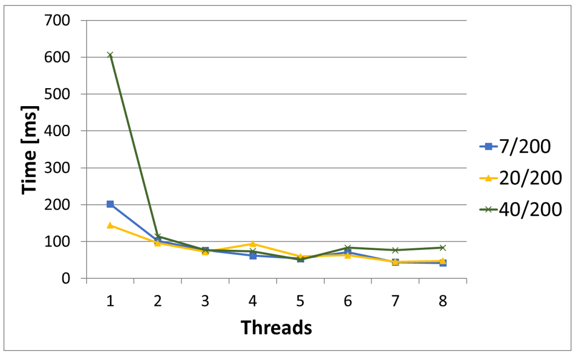


Figure 3.1: Computation time vs. thread count

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.

Result graphs

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). The following chapters will provide a discussion on achieved results.

Direct code conversion to OpenCL

First variant is almost a direct code conversion to OpenCL program flow. The deformation and correlation is computed on GPU, the rest will be done on CPU (looping etc.). Images are arrays of integers, deformations, facets and results are all float arrays. In this variant we utilized 1D kernel, which means we let OpenCL control one of our indexes and we control the second one in Java code. Figure 3.3 shows how the kernel behaves with different LWS0 parameter, which is equivalent to thread count in CPU version. OpenCL also uses threads, but GPU is designed to

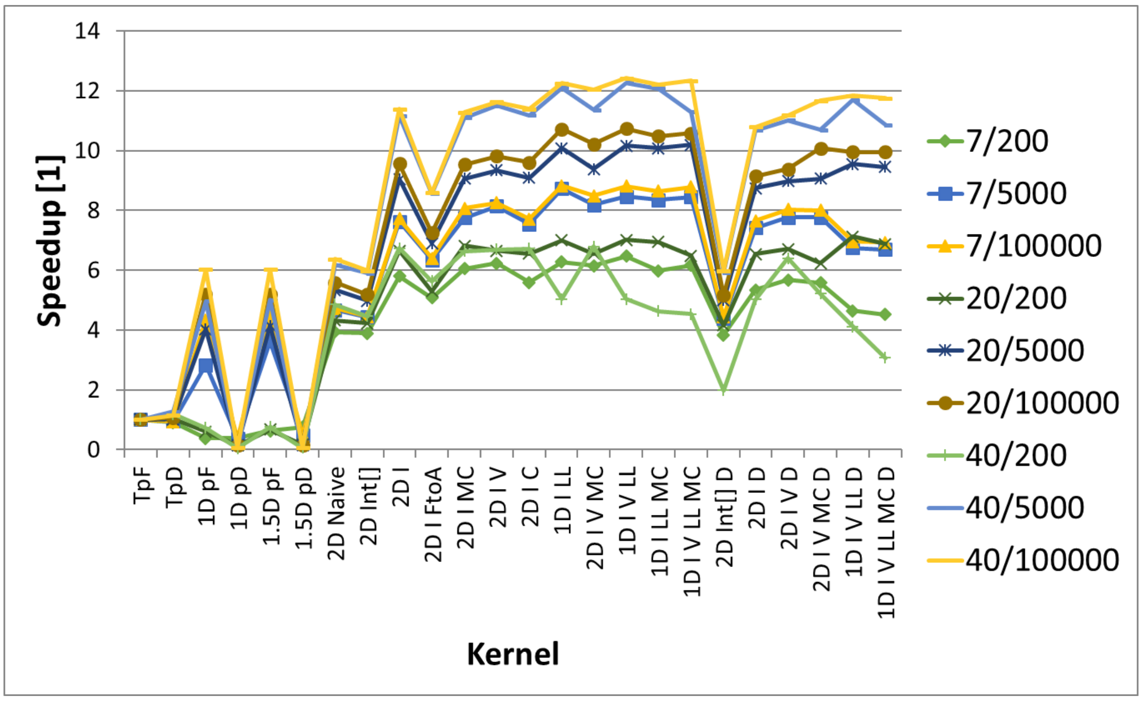


Figure 3.2: Speedup of kernels compared to base variant

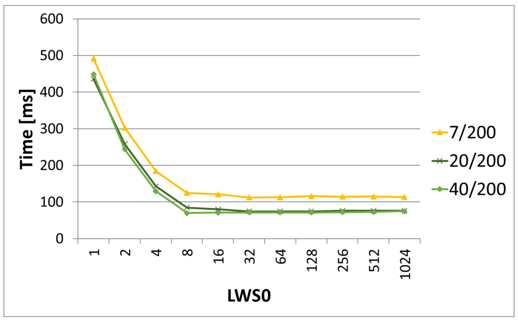


Figure 3.3: Execution time with different LWS0 values

run tens or hundreds at the same time, so in test we tested all available resources (maximal count of workers on GT650M is 1024). The graph shows that it is quite easy to choose the best LWS0, picking any value higher than 8 delivers best performance.

Absolute results show that Java variants are superior to 1D OpenCL per-deformation variant, but per-facet variant performs the same or better in all cases than Java variant.

1.5D kernel

Next kernel variant also uses 1D kernel execution, but we copy all of the data before the computation and only change the index of facet / deformation via kernel parameter. The results show identical behavior to 1D variants, but with far less overhead which provides better computation times.

2D kernel

2D variant removes all looping from Java and lets OpenCL control the indexing. In this case we need to provide two parameters - LWS0 and LWS1, denoting the count of deformations (LWS0) and facets (LWS1) computed in parallel. From the graph we

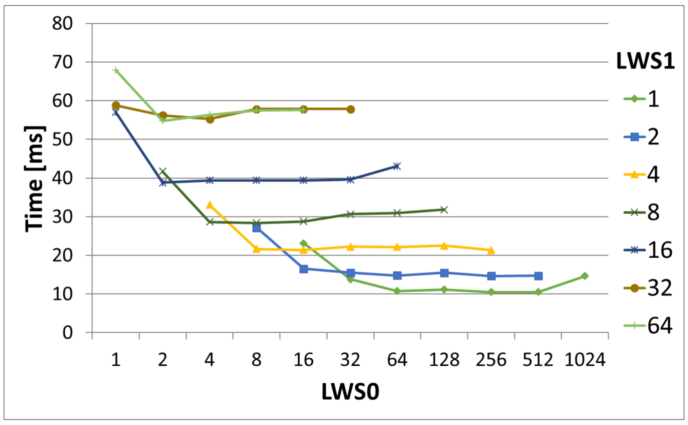


Figure 3.4: Behavior of kernel with different LWS0 and LWS1 values

can tell that computing more than 1 facet in parallel is counterproductive, perhaps due to memory bandwidth limitations. Total count of workers (LWS0 × LWS1) should not reach the highest values (1024 in our case), because the performance drops with higher counts.

Array vs image2D_t

OpenCL contains class called „image2d_t“ which offers optimized memory access using cards texture cache. The downside of image2d_t is that it requires more data preparation, some setup in kernel code and some devices might not support it (this will be discussed later). The results show that image2d_t variants is superior to array variant, cutting the time by as much as 40% in all cases.

Store facet to private memory

This variant aims to use kernels private memory to store facet data, which might lead to some performance gain. The problem is that private memory size is quite small, usually available private memory pool is not that big and the count of work-items is large. The results clearly show that the use of local memory is not productive. The reason is that each work-item has only a handful of registers available, so even when we store the data to local memory in source code, OpenCL discovers that the data are too large to fit so they are read from global memory each time, not from private memory.

Memory coalescing

Main bottleneck in OpenCL applications are usually memory transfers. The reads can be „misaligned“ and a lot of data on the bus might get discarded. The fix for this is called memory coalescing (details can be found in [Khronos group, 2017b]). The results are not conclusive, there is some improvement but also some worsening. This is perhaps due to use of image2d_t, which helps with data transfers by sending image data via another bus.

Vector data types

DIC is a 2D task (can be 3D), so we utilize OpenCL's vector instructions to compute both X and Y coordinates at once. The results are showing that optimized variant performs better in almost all cases than the basic one (up to 7% faster), only in one case it performs same as the basic variant.

Using constant memory

We tried to mark some input data as constant to force OpenCL to store them in constant memory, which might offer quicker access to some data. The results show that there is virtually no improvement, sometimes even the optimized variant performs worse than the basic one. The main problem with this approach is that while definition of constant memory in OpenCL is quite clear, the real implementation and thus the performance of constant memory varies a lot.

Using local memory

Last block of memory we have not used is local memory. This memory is shared between threads in a group. In our case we used local memory to store facet data. The improvement in our test cases is around 10% which is probably the best optimization we have used so far.

Combining approaches

Optimization we used varied in performance, some performed very well while others not so much. Next step in speeding up the computation was to try to combine the approaches and see how they behave. We decided not to use the variant with constant parameters, because it is a not so robust approach with questionable performance gains. This left us with 3 optimization (memory coalescing, using vectors and local memory storage), which we tried in different combinations to find the best one. There is no clear absolute winner, but the best candidate would be the combination of vectorization and local memory.

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 Rest of the algorithm

We did not try to create parallel version of complete DIC computation chain, we focused solely on the correlation computation. However, it is good to perform complete chain on one device to minimize the overhead (data copying, data format changes etc.). So in this chapter we will in short discuss how to achieve full computation on GPU.

Data generation can be done on GPU (both facets and deformations), but it depends on each implementation how it handles input data creation. For example facet data could be generated once and will be applicable for all input images, while in other implementations the target area is dynamic so facets need to be generated again each round so in this case GPU generation will be beneficiary.

After the correlation computation is done we need to pick the best result. This is a general task of searching for the best result in an array. Several very fast implementations exist for this task for both CPU and GPU.

The last step of DIC algorithm is strain field calculation. There are multiple approaches to do this, but for most of these there exists several GPU implementations, because the approaches are usually some application of more general method (eg. filtering, differentiation, least squares) to a special case. And for those general methods, there again exist many libraries computing this kind of problem on GPU, so it is just case of correct data handling to achieve the computation on GPU.

3.1.6 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.

All of the mentioned approaches can be converted to 3D case (larger vectors are available in OpenCL, image2d can be switched to image3d). This means that our approach is universal and can be used in any DIC implementation, because it can be used both in 2D and 3D and is solver independent (some solvers may benefit more, some less but virtually any kind of solver can be sped up using our GPU oriented approach).

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. From this we can see that the size of deformations can get quite big.

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 Implementation

This section will focus on how we can generate data from given limits along with description of scenarios we have used for testing.

Subset data

Given subset center and its size we can easily compute top-left point of subset via simple subtraction. Now we can iterate over all points and generate theirs positions (X and Y) by simple division and modulo of points rank (see equation 3.1), where S

is $(2M+1)$ and i is the points rank (ranging from 0 to $[(2M+1)\times(2M+1)-1]$).

$$x = i \bmod S \tag{3.1}$$

$$y = i \operatorname{div} S$$

We can see that the formula for computing points position is very simple and thus the computation of values should be fast.

Deformation data

For deformation generation we can utilize OpenCL's built-in indexing system, where one of the indexes describes the rank of deformation. We only need to convert this decimal number to set of coefficients. This can be also achieved by simple division and modulo, because we can imagine this problem as conversion from one numbering to another. We only need to define the order of coefficients and then we can find the n-th element.

GPU data generation

We devised two types off GPU data generation - off-line and on-the-fly. Off-line generation mimics CPU data generation - launch multiple workers that will generate a portion of deformations and store them. While testing this approach we have found out that smaller count of workers is desirable, perhaps due to limited memory bandwidth. Computation kernel then gets same data as in CPU generated case.

On-the-fly approach skips the generation part and the data are generated when they are needed and are not stored anywhere. This might seem counterproductive, because we will generate same data multiple times, but we need to keep in mind that GPUs are very good at computing things, but perform worse in terms of memory bandwidth. We have tried all possible combinations of CPU, off-line GPU and on-the-fly GPU data generation for subset and deformation data to find the best variant.

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 3.5 and 3.6, 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 3.6). 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 3.5 and 3.6, all variants behave quite the same, and only the speedup curve looks cleaner. Effect of subset count on performance can be seen in Figure 3.7. 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.

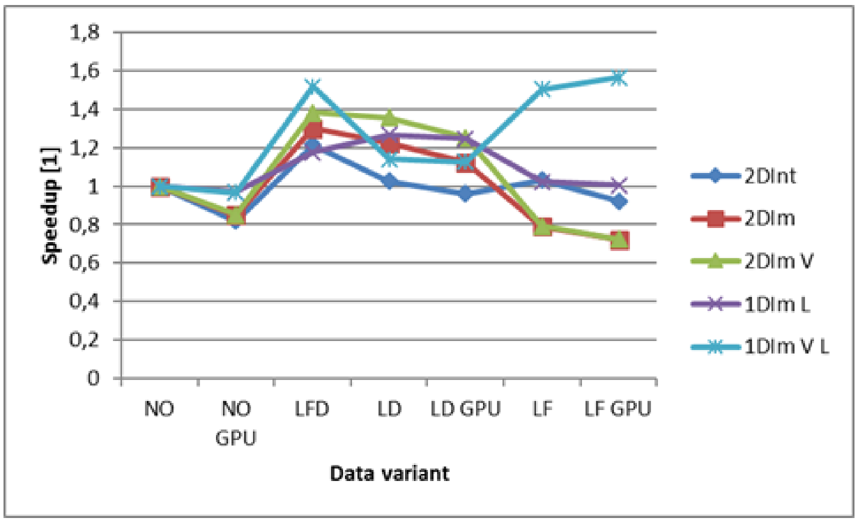


Figure 3.5: Subset size = 20, Subset multiplier = 1, Image size = 88x240, Deformation count = 50

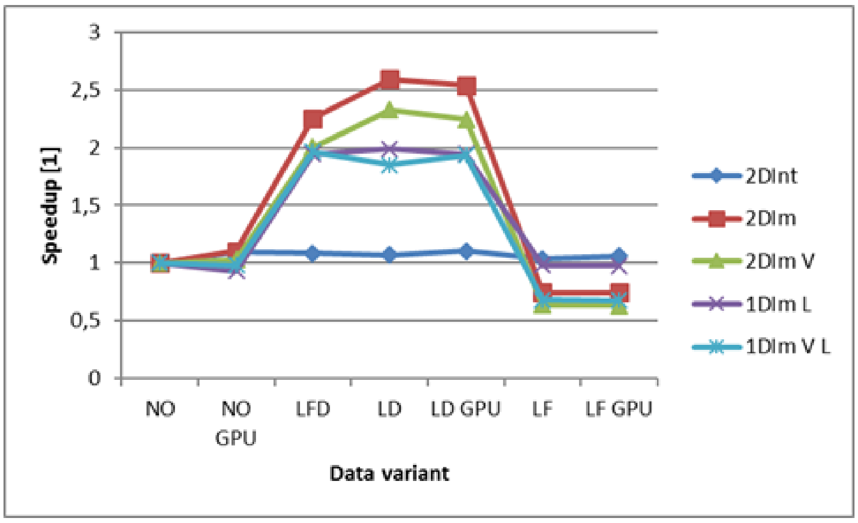


Figure 3.6: Subset size = 20, Subset multiplier = 1, Image size = 88x240, Deformation count = 1000

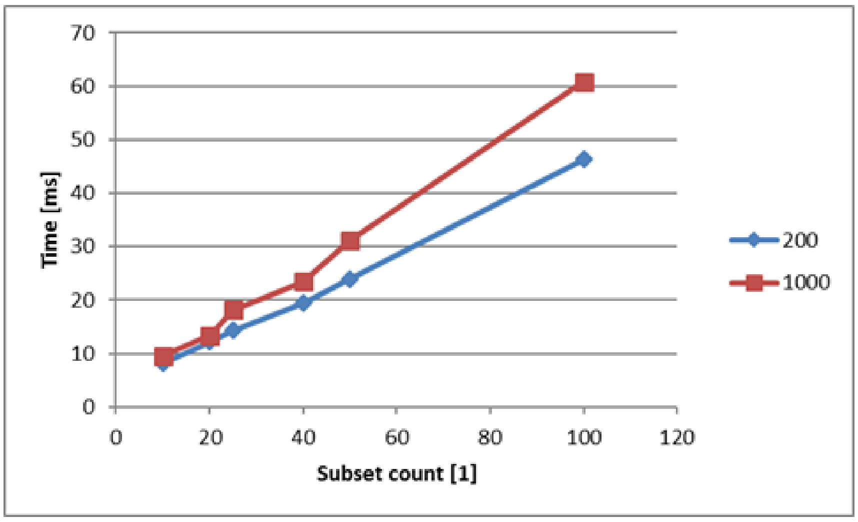


Figure 3.7: Time needed for computation with different subset counts

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 chapter 3.1). 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 Test setup

Input data configuration consists of two parameters - subset count and deformation count. As mentioned earlier, the deformation can be of different order, in our tests we have tested zero and first order. The subset size was set to 31 pixels. The counts of subsets and deformation we used can be seen in Table 3.1.

Subset count	1	2	8	32
Zero order deformations	3	21	169	441
First order deformations	160	486		

Table 3.1: Input data configuration

The kernel generation is as a task of generative programming. We have created a base template, which can be seen in Listing 3.1. Each token (text surrounded with percent signs) is then replaced with valid OpenCL code according to kernel specification.

Listing 3.1: OpenCL kernel base file

```
%INT%    // interpolation function
%CORR-G%  // function to compute gauss distribution

kernel void DIC(%HEAD%) // function header
{
       %INIT%  // id checks, memory init
       %DEF-C%  // prepare deformation coeffs
       %DEF-S%  // deform subset
       %CORR-M% // compute mean
       %CORR-D% // compute delta
       %CORR-C% // compute correlation
       %S%  // store result
}
```

OpenCL code itself is not enough to commence the computation. We also need a „host“ code, which essentially launches and drives the computation. The host code must copy the data to the device and then launch the kernel with proper parameters. We have used Java and JogAmp's JOCL library [JogAmp, 2017] to access OpenCL functionality. The Java code is not universal as OpenCL code, we have created separate classes for 1D, 1.5D and 2D kernel launch. JOCL library offers CLMemory interface for all data, that can be copied to GPU, so the data generation is also universal, because we only pass a list of CLMemory objects to JOCL function. To

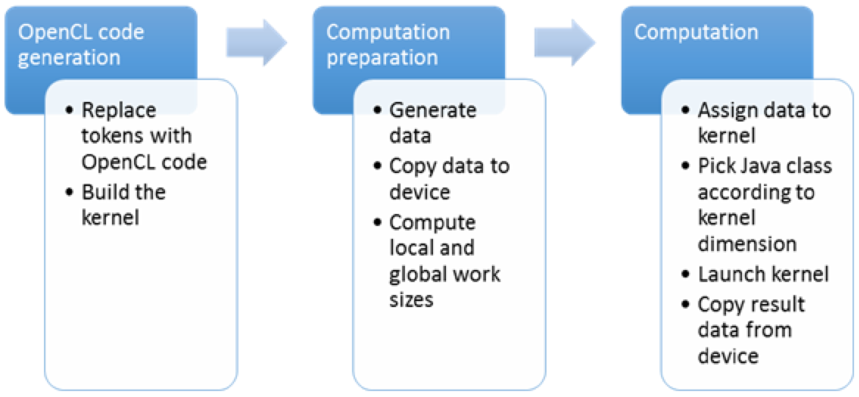

OpenCL code generation
Replace tokens with OpenCL code
Build the kernel
Computation preparation
Generate data
Copy data to device
Compute local and global work sizes
Computation
Assign data to kernel
Pick Java class according to kernel dimension
Launch kernel
Copy result data from device

Figure 3.8: OpenCL dynamic kernel generation

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

3.3.2 Results

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.

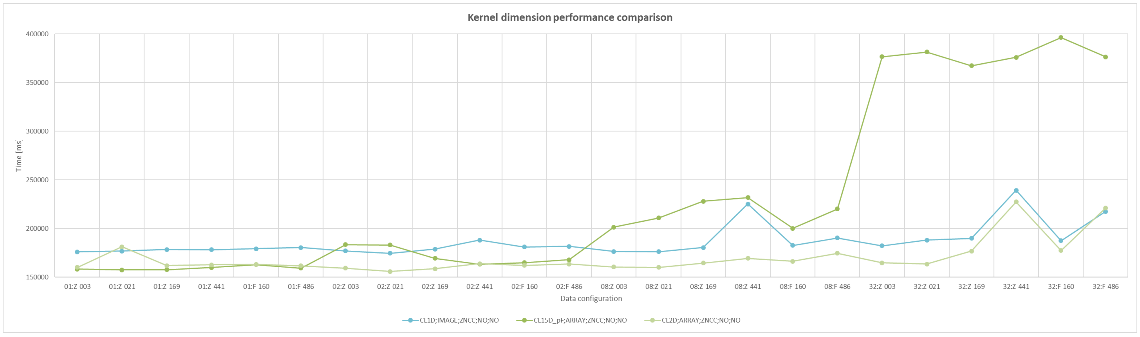


Figure 3.9: Kernel dimension performance comparison

Figure 3.9 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

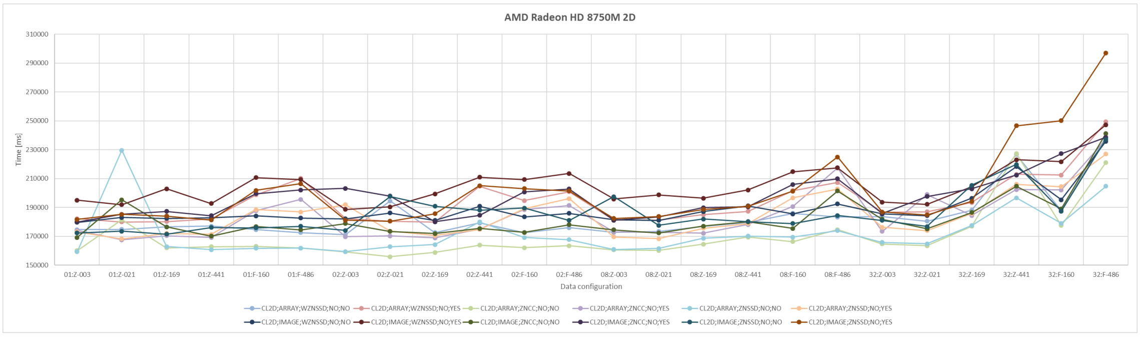


Figure 3.10: 2D kernel comparison for AMD Radeon HD 8750M

Figure 3.10 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 3.11 and 3.12.

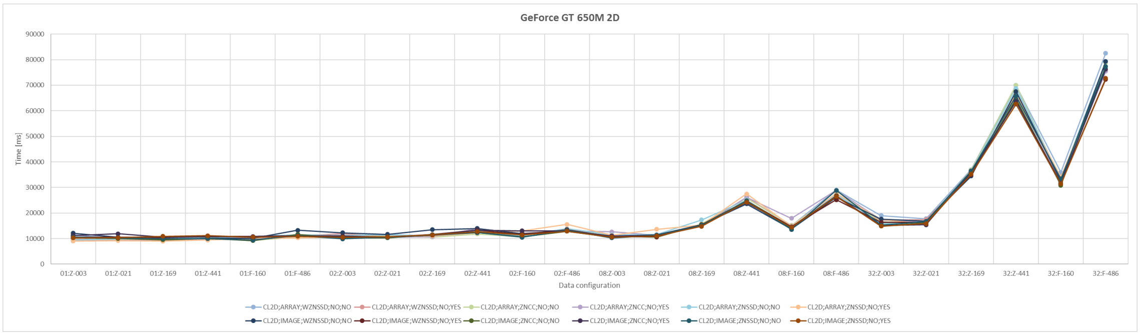


Figure 3.11: 2D kernel comparison for NVIDIA GT 650M

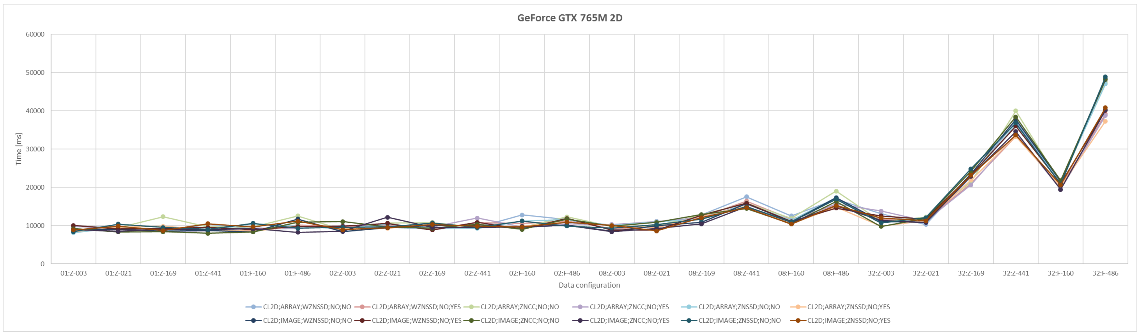


Figure 3.12: 2D kernel comparison for NVIDIA GTX 765M

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 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.

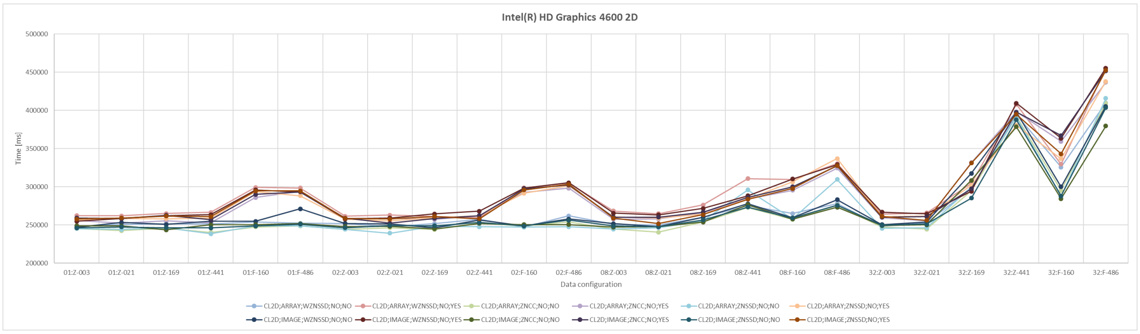


Figure 3.13: 2D kernel comparison for Intel HD 4600

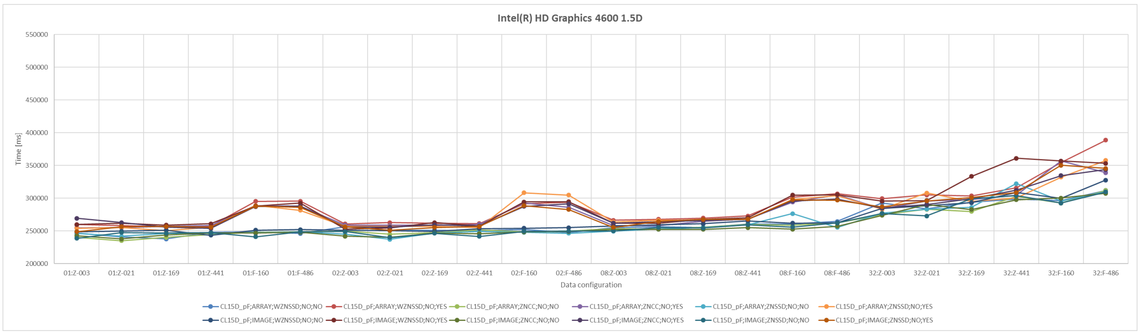


Figure 3.14: 1.5D kernel comparison for Intel HD 4600

First surprise was that 2D kernel's (Figure 3.13) performance was about 5% worse than performance of 1.5D kernel (3.14). 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 3.14 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.

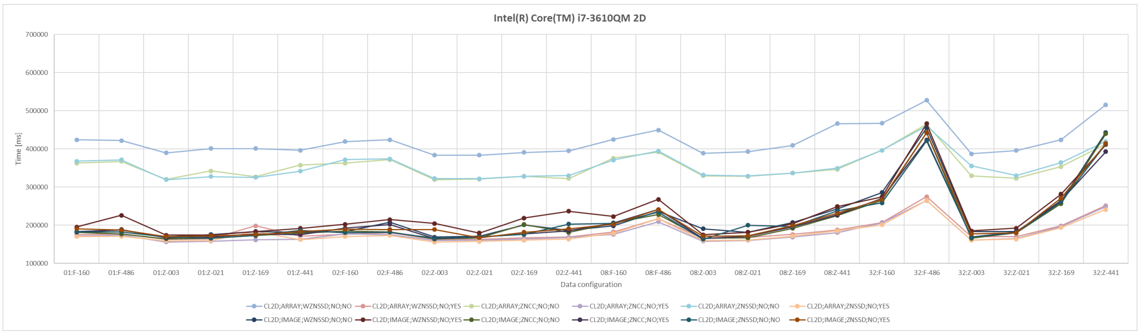


Figure 3.15: 2D kernel comparison for Intel Core i7-3610QM

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.

3.3.3 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 3.19 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]. Following chapters will describe filters we have used along with parameters we have tested.

Histogram filters

The goal of histogram filters is to improve contrast in images using the image's histogram. The contrast is an important property of image for DIC, because it allows better distinction of subsets.

Histogram equalization Classical histogram equalization is done by computing image histogram and its flat version by computing linear cumulative distribution function.

CLAHE CLAHE algorithm expands the equalization process by computing several histograms for distinct sections of image. This approach is more suitable for local contrast improvement.

Spatial filters

Spatial filters are implemented as 2D convolution of input image with a 2D kernel. As mentioned in [Zhou et al., 2015], both Gaussian and binomial filters perform well for DIC and we wanted to test them as baseline for other filters.

Gaussian The coefficients of kernel are computed from Gaussian distribution (equation 3.2). We have tested 3 kernel sizes - 3x3, 5x5 and 7x7.

$$G(x) = \frac{1}{2\pi\sigma^2} \times e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.2}$$

Binomial Kernel values of binomial filter are obtained from binomial coefficients. The 2D kernel is computed as outer product of binomial coefficients, once as row, once as column. We have also tested 3 sizes - 3x3, 5x5 and 7x7.

Other filters

Wiener Wiener filter minimizes the mean square error between input image and estimated error. Thus we need to provide noise estimate in order to be able to use Wiener filter. We have estimated the noise with Gaussian point spread function with multiple sizes and sigma values and also tested several noise-to-signal ratios for wiener filter.

Lucy-Richardson Lucy-Richardson filter tries to filter input data that have been blurred by known point spread function. This filtering should remove blurring caused either by movement or by detector faults. The parameters of this filter are same as for Wiener filter.

Median Median filter is a nonlinear filter for noise removal, which is able to preserve edges. The main idea of the filter is to replace each pixel value with the median of neighboring pixels.

Bilateral Bilateral filter computed new intensity of pixel as weighed average of intensity values from nearby pixels. The weight is based on multiple criteria, for example on Gaussian distribution ad radiometric differences. The radiometric approach helps in edge preservation. The filter has 2 parameters - size and radiometric distance, we have tested sizes from 3 to 9 and distances from 25 to 150.

3.4.2 Results

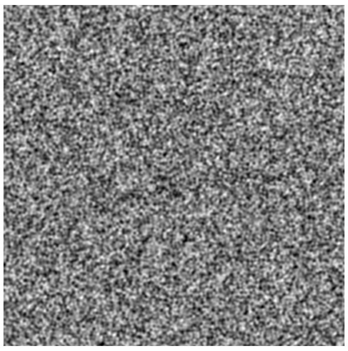
As mentioned in previous chapter, we have multiple combinations of image sets, filters and subset sizes. We will present only a portion of numerical results along with discussion about the behavior of filter in other data combinations and overall filter quality. This section will be divided in three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

SEM data

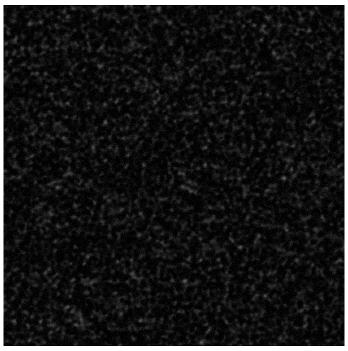
Society for Experimental Mechanics offers test data ([SEM, 2017]) to allow testing and comparison of DIC solvers. We have picked several data sets to thoroughly test the impact of pre-filtering on solver accuracy. SEM offers total of 17 data sets with varying deformation magnitude and order. We have 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 3.16 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 3.17 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.

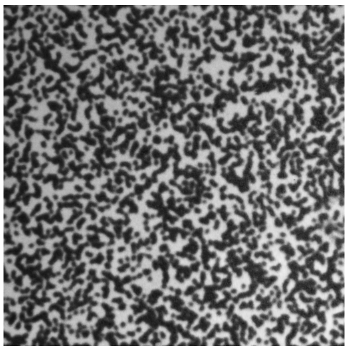
The quality results in figure 3.18 show a little bit different results than displacement error analysis. Some filters work very well in terms on correlation quality, while others not. Bilateral filter performs the best in terms of correlation quality, followed closely by Gaussian and Binomial filters. Median filter provides good result quality, but the precision of the results drops significantly (as stated



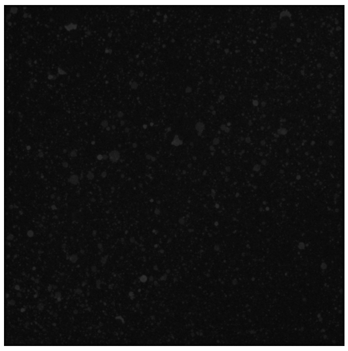
(a) Sample1



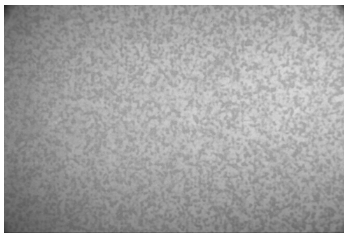
(b) Sample2



(c) Sample3



(d) Sample4



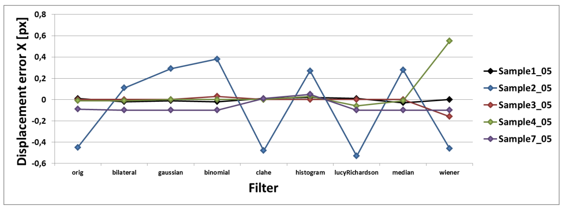
(e) Sample7

Figure 3.16: SEM test data examples

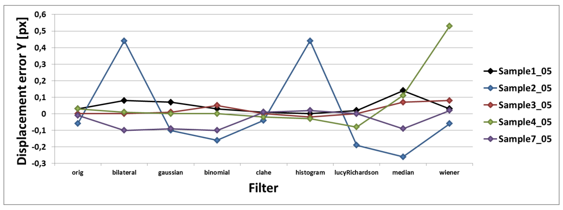
before), so Median filter is a poor choice. Wiener filter is also a good choice, but the selection of good input parameters is not as easy as for other filters.

We have also analyzed the deviance of resulting values, as can be seen in Figure 3.18b. 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.

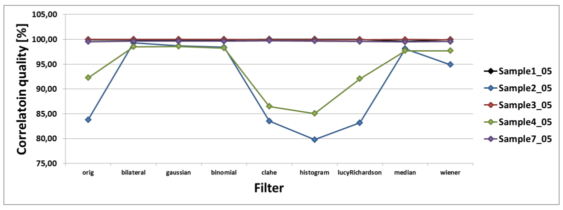


(a) X direction

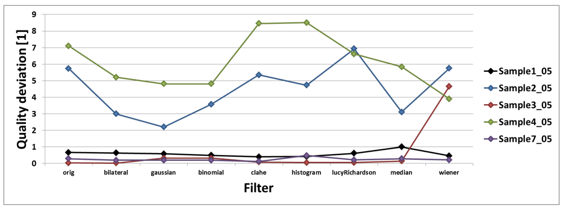


(b) Y direction

Figure 3.17: Displacement error for SEM data, subset size 5px



(a) Median value

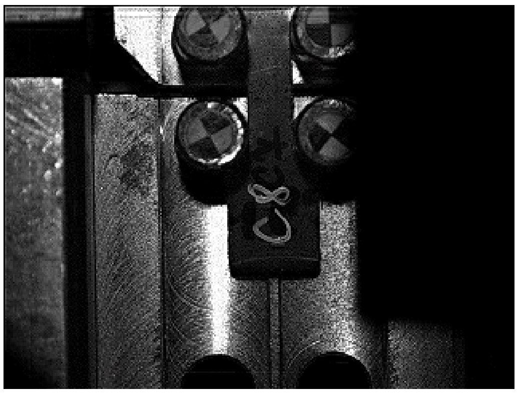


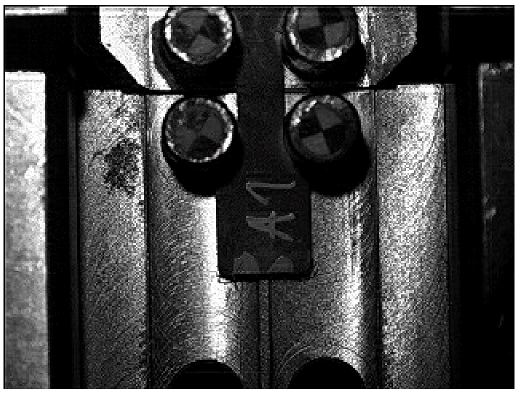
(b) Deviance

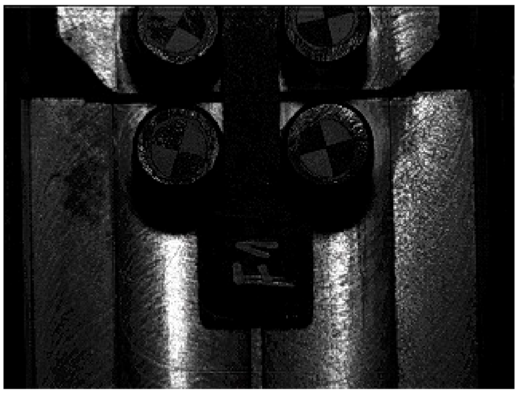
Figure 3.18: Correlation quality results for SEM data, subset size 5px

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 3.19 shows examples from 4 sets of


(a) 6107544m


(b) 6203652m


(c) 7202845m

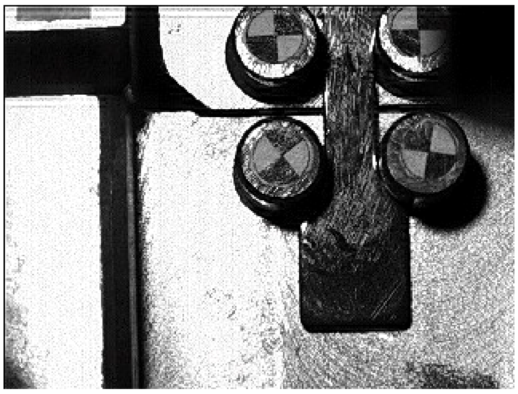
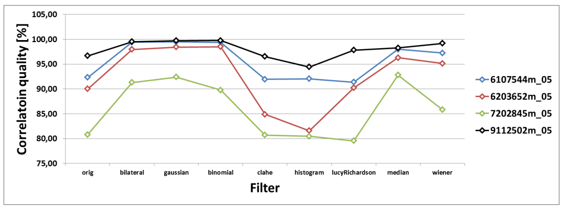
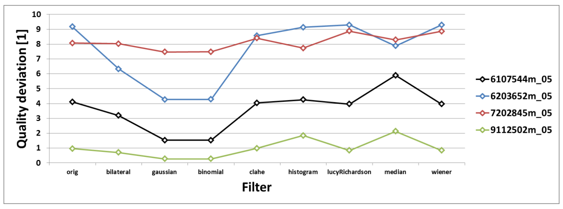

(d) 9112502m

Figure 3.19: Real world test data examples

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. 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.



(a) Median value



(b) Deviance

Figure 3.20: Correlation quality results for real data, subset size 5px

Figure 3.20 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.

The graph of result quality values deviation (Figure 3.20b) confirms the behavior seen in absolute values, best filters lower the value deviation, while the others keeps it same or make it bigger.

General filter performance

In this section we will summarize performance for each filter in all configurations.

Histogram equalization

Histogram filter performed poorly in most cases. The mean quality stayed the same, the deviation however grew a lot in most cases, which is not desirable and thus the histogram filter is not a good pre-filtering choice.

CLAHE

While CLAHE filter's function is similar to histogram filter, it performs the equalization locally, which can produce better contrast for complex scenes. The results show that this filter performs better than basic histogram equalization. For Sample 7 from SEM data set, it was the best filter of all. For other data sets, the quality of results remained almost the same as for un-filtered image, in some cases it performed slightly better.

Gaussian

Gaussian filter is a standard in filtering and our results confirmed, what Zhou et al. have found out. The filter improves results quality and precision in almost all cases. For Sample 3, the quality remains the same, but the results deviance rises.

Binomial

Binomial filter behaved much like Gaussian, in some cases slightly better in terms of data deviance.

Wiener

Wiener filter worked pretty well, in most cases its performance was comparable to Gaussian and Binomial filter. The problem is input parameter estimation. You need to provide the error functions (typically Gaussian, so you need two parameters - size and sigma) and noise-to-signal power ratio and it is quite hard to estimate proper parameter values beforehand. But we found that using higher noise-to-signal ratio and Gaussian error function with larger size and sigma value provides good results in all data configurations.

Lucy-Richardson

Lucy-Richardson filter was the worst filter we have tested. The results quality plummeted and the precision in SEM data was terrible. There was only one exception, for Sample 7, Lucy-Richardson filter allowed correct estimation of deformation, which was not common for other filters.

Median

At first look Median filter performed quite well in terms of results quality. But when we looked at displacement results, they were pretty much rubbish in all cases. This might be because while Median filter does remove noise, its main focus are edges, not the texture overall. This probably harms the specimen texture and thus the DIC algorithm fails to find good results.

Bilateral

In our test, Bilateral filter performed on par with Gaussian filter, in some cases Bilateral filter even outperformed Gaussian and Binomial filter. The main advantage of Bilateral filter in terms of result quality was its ability to lower data deviance, so it is less prone to random errors. The downside of Bilateral filter is its computation complexity, the computation time rises almost 10 times, depending on input image and filter implementation compared to Gaussian filter.

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.

[Lecompte et al., 2006] analyzes the influence of the speckle pattern and subset size on the accuracy of the measured displacement. They show that it is not necessary to obtain the smallest possible speckle pattern to achieve high accuracy, often it is sufficient to use evenly distributed speckle pattern. Another result of their paper is that with larger subset sizes, the results get better, but the subset size must be chosen in accordance with expected deformations. Too large subset size leads to errors due to smoothing of the real material behavior. [Wang et al., 2007] introduces a formula, which can estimate the standard deviation of the displacement measurement error. It can be used to predict the precision of the experiment, but also as a guide for tuning the experimental setup in order to obtain best results (by changing the subset size, filtering out the noise or altering the speckle pattern diversity). [Yaofeng and Pang, 2007] focuses only on subset size and its effect on result's precision. They use subset entropy (average of absolute intensity deviations at any point in the subset from its neighboring points) to define „effective subset size“ to allow the comparison of subset sizes in literature. The entropy characterizes the speckle pattern, which is essential for the precision (as shown in the paper).

The first paper offering an automated way of estimating optimal subset size is [Pan et al., 2008]. It offers an algorithm using a SSSIG criterion (Sum of Square of Subset Intensity Gradients) and the variance of image noise for selecting the optimal subset size based on required standard deviation of displacement error. The SSSIG criterion is loosely related to subset entropy introduced in [Yaofeng and Pang, 2007]. The main difference is that while subset entropy is based on intuitive idea, the SSSIG is based on a mathematical concept. All of the mentioned papers can pre-calculate the optimal subset size, but the computation is based solely on the image qualities and does not take into account the solver and the numerical calculation.

[Huang et al., 2013] introduces a weighting parameter to classical Newton-Raphson sub-pixel registration method to improve the results quality by adaptive weighing the pixel values in the subset. The weighing parameter is a coefficient parameter (D_0) in Gaussian window function

$$W_G(X, Y; D_0) = exp[-D^2(X, Y)/2D_0^2]. \tag{3.3}$$

By lowering the parameter we select only central pixels in the subset and thus we can alter the subset size dynamically. The results show 30% error reduction for small deformations and 60% error reduction in large deformation test case. [Yuan et al., 2014] couples the weighing with ZNSSD correlation criterion to create WZNSSD criterion. The main result of the paper is that the weighed approaches can very effectively diminish the wrong choice of the subset size, both globally (due to poor use choice) and locally (optimal subset size may vary in different parts of the image). While providing good results for displacement error, both algorithms could not reduce the errors in strain field estimation.

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.

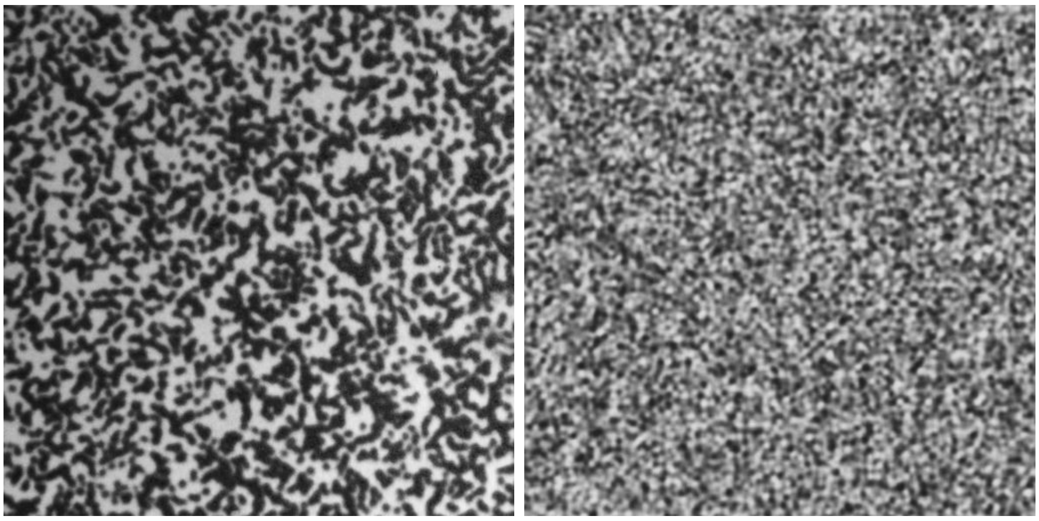

(a) Dataset "Sample 03" (b) Dataset "Sample 10"

Figure 3.21: DIC Challenge datasets

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 3.21a) and „Sample 10“ (Figure 3.21b) 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).

3.5.1 Theoretical framework and algorithm

One type 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 3.22). 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 3.22). 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 3.22), 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

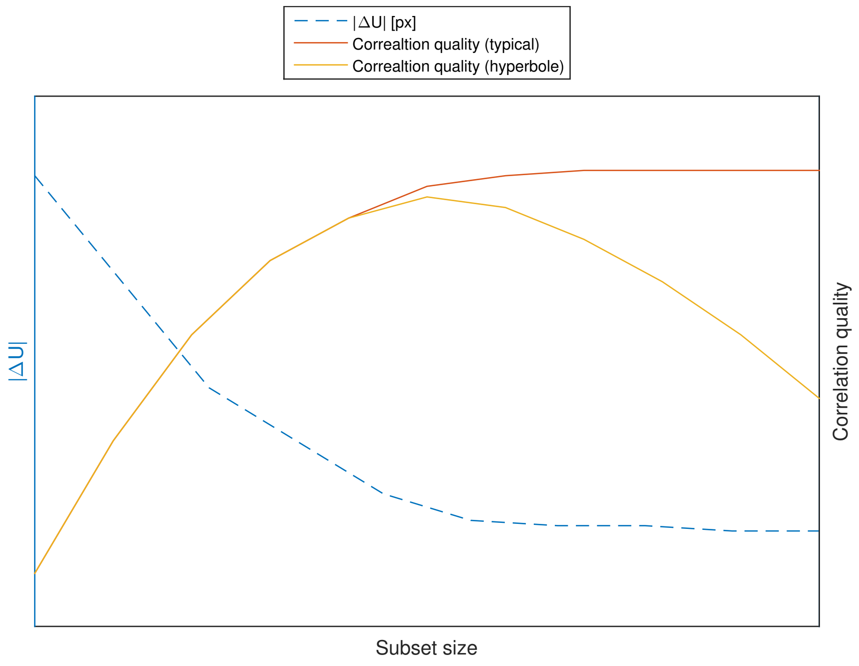


Figure 3.22: Illustration of typical displacement error function of subset size vs correlation quality function

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 3.4). The value of new subset size is then computed using equation 3.5. 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 3.24 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.

$$\begin{aligned} y &= m \times x + b \\ m &= \frac{y_2 - y_1}{x_2 - x_1} \\ b &= y - m \times x \end{aligned} \tag{3.4}$$

$$x_{new} = \frac{y_{new} - b}{m} \tag{3.5}$$

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.

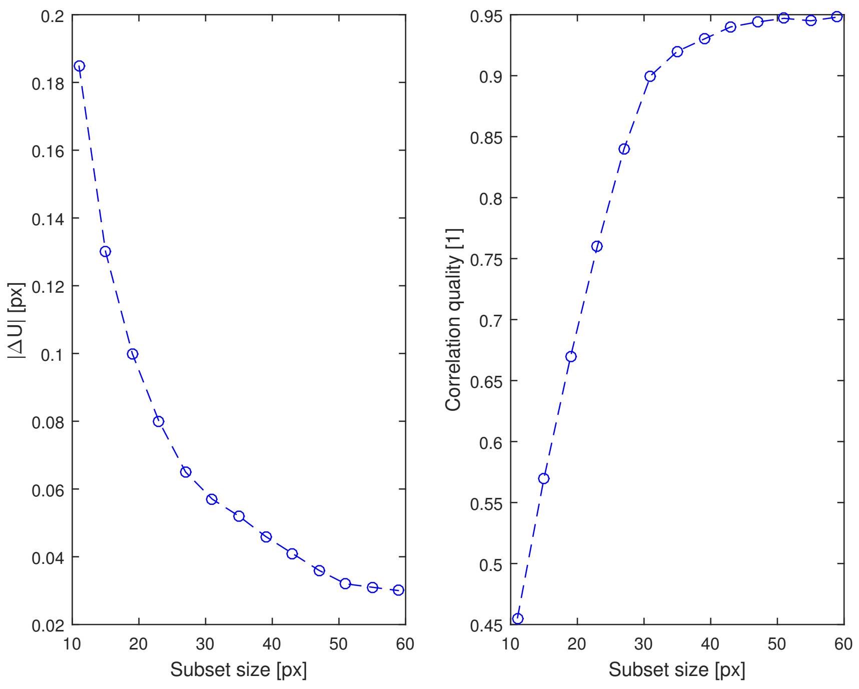
Approximation on one step

Figures 3.23a and 3.23b 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 approximation can be seen in Figure 3.24. 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.

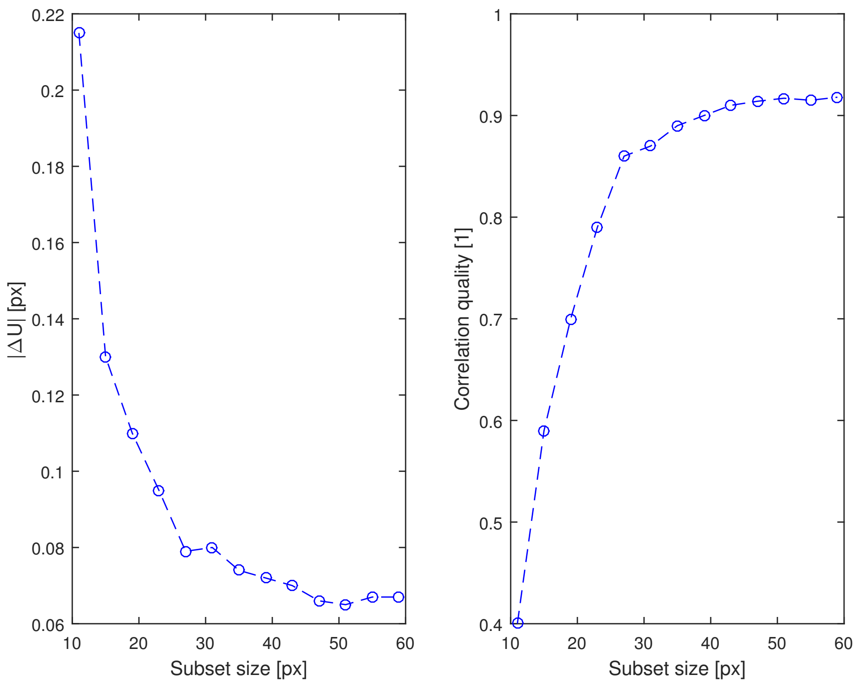
Next we ran the test with spline approximation. First we have tried to approximate the function on three points. The results are presented in Figure 3.25. The first sub-figure (3.25a) uses the same data points as the initial step in iterative approximation (two values for smallest subset sizes and one 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 3.25b 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 pixels). The results of such approximation can be seen in Figure 3.25. The error of the approximation is under 5%, which is enough to allow very precise estimation of optimal subset size.

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



(a) Dataset "Sample 3"



(b) Dataset "Sample 10"

Figure 3.23: One step approximation displacement error and correlation quality

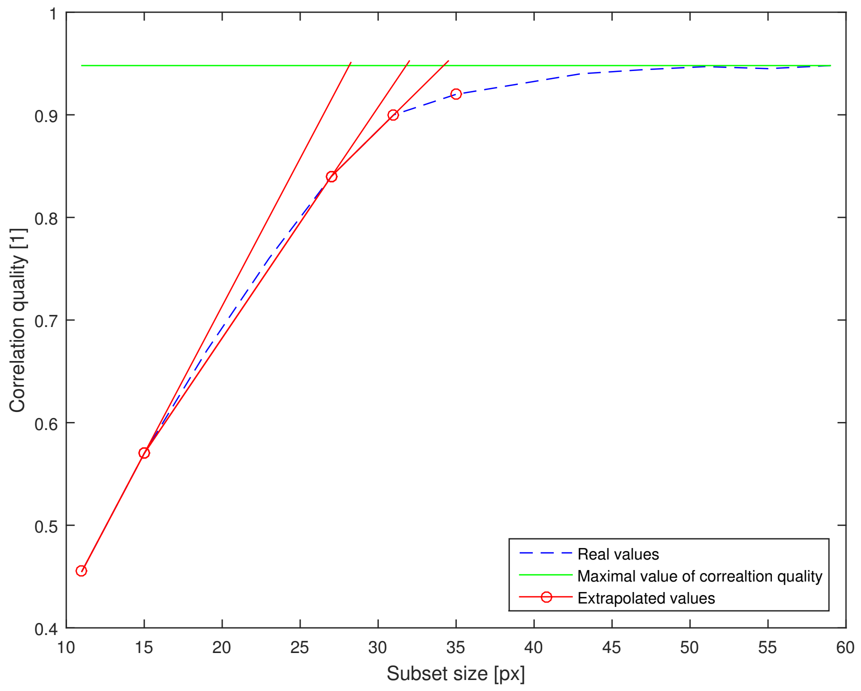
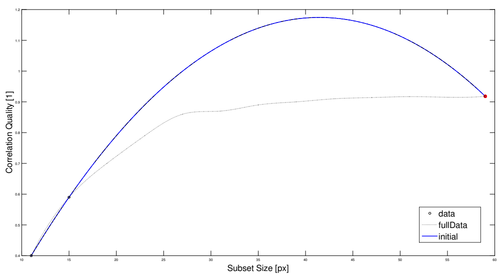


Figure 3.24: Iterative linear approximation

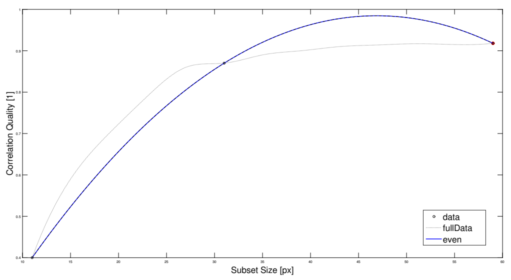
the number of points needed strongly depends on the termination criterion and the exact shape of the function.

3.5.3 Numerical results

Figures 3.27a and 3.27b 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 3.23a and 3.23b we can see that the



(a) Same points as iterative approximation



(b) Three evenly spaced points

Figure 3.25: Approximation with cubic spline on three points

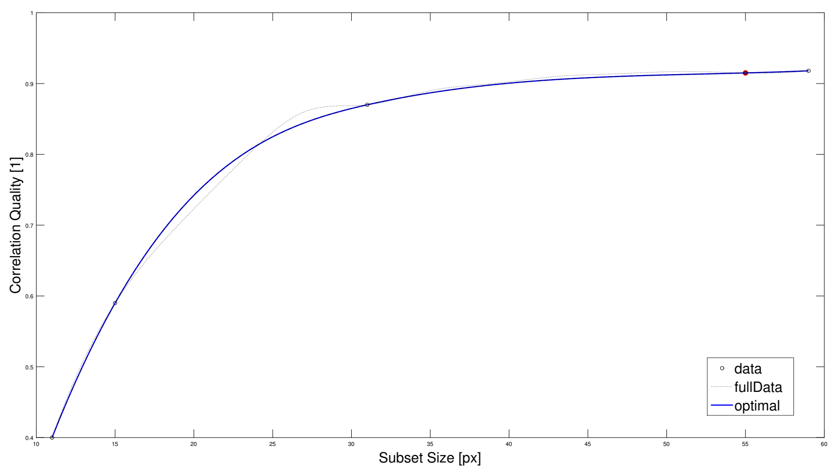
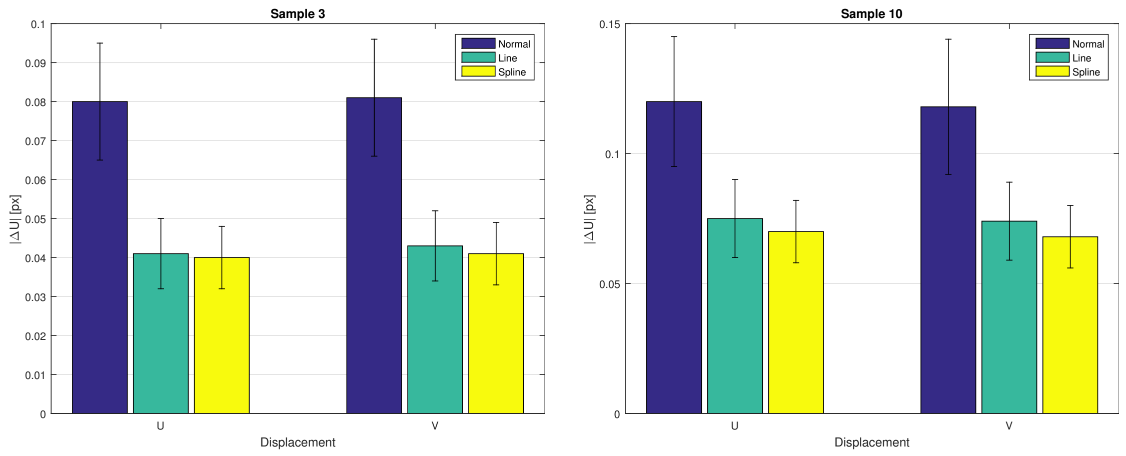


Figure 3.26: Approximation with cubic spline on five points



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

Figure 3.27: Comparison of statistical errors (mean and standard deviation)

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 Application GUI

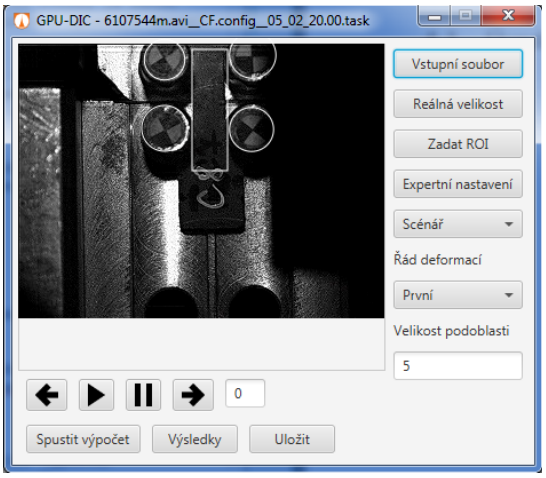
Main window of the application (image 3.28a) offers basic options that are expected - input data specification, input data browsing, application and computation engine settings, results and project persistence. Input data can be in form of video or set of images. The video is split into separate images and input data are converted to gray-scale (if needed).

Usually the supplied input data also contain other data than the specimen itself. The application offers to mark the ROI in two ways (image 3.28b). First is the classical rectangle marking the region. The second way is designed for stress tests where the jaws hold the specimen. The user can mark them with 4 circles and the computation engine will track the movement of the jaws and adjust the rectangular ROI accordingly.

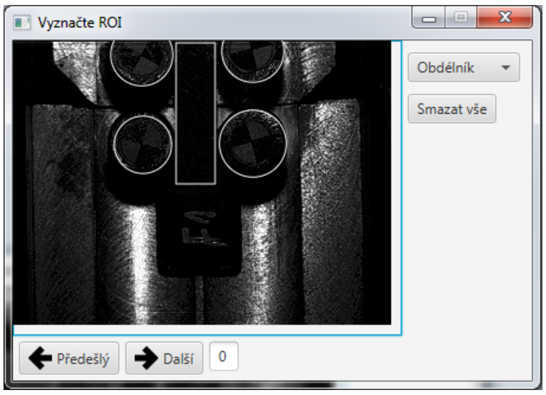
Standard output of the DIC analysis is in the form of pixels (how much has the given pixel moved between images). But for the user the output in form of pixels is unusable because typically he has no knowledge in image processing. For this reason we have created an interface to mark the real size of the specimen (or some known dimension, the marking process can be seen in image 3.28c). In standart 2D DIC test setup the specimen plane is parallel to camera plane, thus we can assume that the ratio of pixel size versus real size is constant across the image (or at least the specimen). Using this information the application can provide the result in form of millimeters, not pixels.

Apart from the map view, the user can also view the result for one point (image 3.29b). This presents him a time series of computed values for a given point with the ability to save this data series.

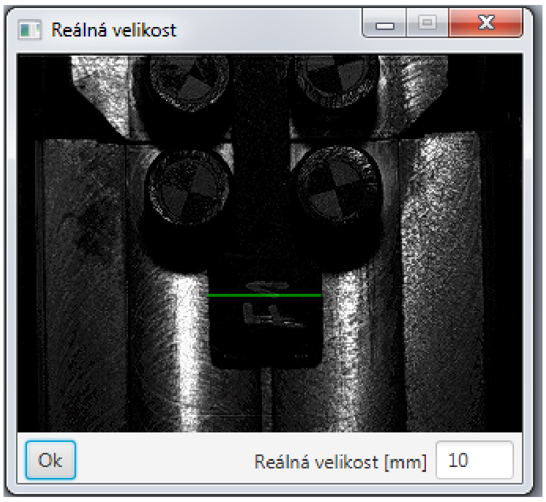
A typical outcome of the traditional deformation analysis with stress gauges is in the form of deformation that occurred between the measured points. Our application mimics this by allowing the user to specify two points and showing him the relative deformation (or strain) between the selected points (image 3.29c).



(a) Main window



(b) ROI marking



(c) Marking real size of specimen

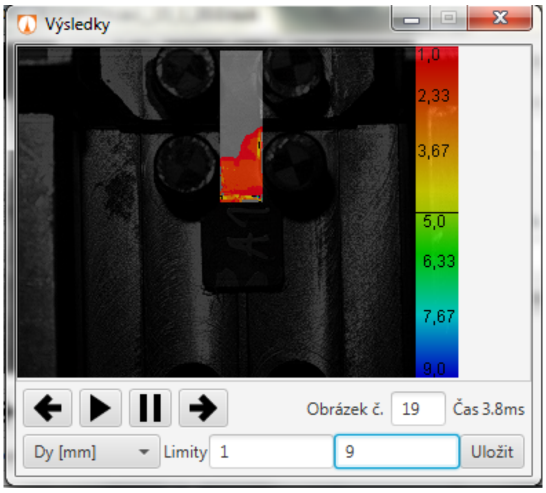
Figure 3.28: GPU-DIC application GUI - task specification

There is also an option to export the results to external format to allow further processing and / or presentation. The results can be exported in the form of images (deformation / strain maps for all round), excel files (numerical values of maps or values of a time series in given point) or in the form of video, which joins the result images in one video.

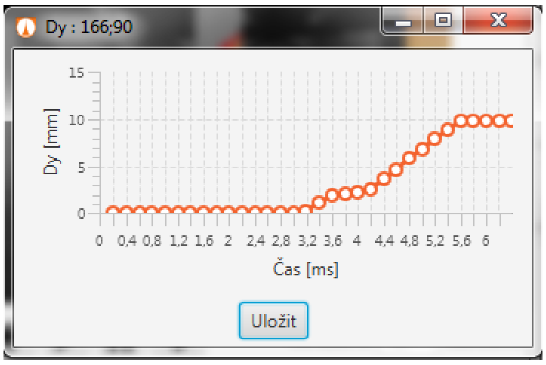
After the computation is done the user can view the results (image 3.29a). The result window offers basic controls for browsing and viewing the result such as image browsing or playback, choice of result type (deformation, strain etc.) and adjustable value limits to be able to quickly filter out extreme values.

3.6.2 GPU-DIC vs other applications

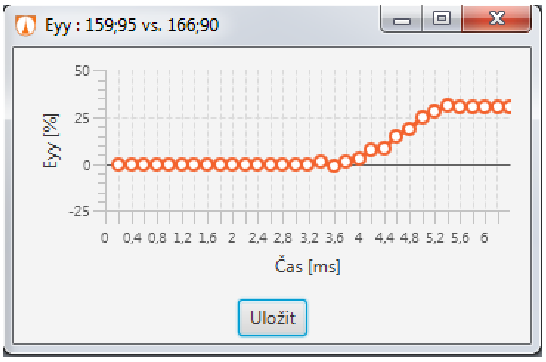
There exists several solutions to perform DIC analysis ranging from complex systems with full test setups (cameras, mounts, lights etc) to software-only solutions, where



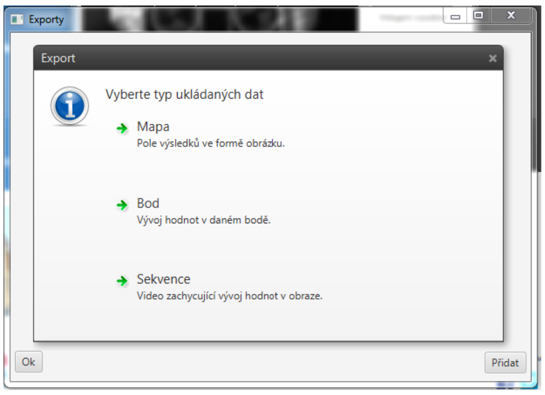
(a) Result window



(b) Results for one point



(c) Results for 2 point strain


Exporty

Export

Vyberte typ ukládaných dat

Mapa
Pole výsledků ve formě obrázku.

Bod
Vývoj hodnot v daném bodě.

Sekvence
Video zachycující vývoj hodnot v obraze.

Ok

Přidat

(d) Result export

Figure 3.29: GPU-DIC application GUI - results

the users needs to obtain the recordings some other way. We have explored and tested several solutions to see how they behave compared to our software.

Dante Dynamics offers integrated solutions for both 2D and 3D measurements ([Dante Dynamics, 2017]). They offer modeling the deformation with first order deformation function. They do not specify the exact algorithm used for the optimization but they report the precision up to 0.01px. A complex software solution is supplied allowing camera calibration and contour detection. The result presentation module contains basic range of modes such as map view, single point analysis etc.

Correlated solutions is a company focusing on developing software for non-contact shape and deformation measurement ([Correlated solutions, 2017]). They offer a complex software solution bundled with a high speed camera (several models are available with different speeds and resolution). One of the key points of their solution is a possibility of real time analysis during the stress test. However, they allow analysis of only several points in this mode, not the whole image.

GOM ([GOM, 2017]) develops and produces precise 3D measurement solutions based on DIC algorithm. Their main focus is on ease of use and intuitive inspection process, where the software is able to offer insights on which measuring principles and inspection criteria are best suited for given problem. With CAD import their target audience are designers and developers from engineering fields.

NCORR ([Blaber and Antoniou, 2017]) is an open source solutions for DIC analysis. It is implemented mostly using MATLAB programming language, the most performance demanding parts are implemented using C++. The NCORR software offers several different algorithms for the optimization based on the Gauss-Newton (GN) iterative optimization scheme such as FA-GN (Forward Additive) and IC-GN (Inverse Compositional). While the GUI is created in MATLAB (the support for GUI creation in MATLAB is limited at best), the application offers complex interface for DIC analysis best suited for advanced users.

The MOIRE software developed by Opticist.org ([Wang and Vo, 2017]) is another free 2D / 3D DIC software solution. The main difference between MOIRE and other solutions is the use of reliability-guided algorithm, which can automatically identify and exclude erroneous (typically points around the edges of ROI) result from computation allowing to achieve uniform precision of results across the whole image.

3.6.3 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 3.21a). 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

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 3.2: Comparison of performance of various DIC solutions

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.

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

[ArrayFire, 2017] ArrayFire (2017). Jacket matlab toolbox. `https://arrayfire.com/tag/jacket/`. [Online; accessed 2017-05-30].

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

[Barthelat et al., 2003] Barthelat, F., Wu, Z., Prorok, B., and Espinosa, H. (2003). Dynamic torsion testing of nanocrystalline coatings using high-speed photography and digital image correlation. *Experimental Mechanics*, 43(3):331–340.

[Bay, 1995] Bay, B. K. (1995). Texture correlation: a method for the measurement of detailed strain distributions within trabecular bone. *Journal of Orthopaedic Research*, 13(2):258–267.

[Bay et al., 1999] Bay, B. K., Smith, T. S., Fyhrie, D. P., and Saad, M. (1999). Digital volume correlation: three-dimensional strain mapping using x-ray tomography. *Experimental mechanics*, 39(3):217–226.

[Beata et al., 2012] Beata, M., Tomasz, M., Zbigniew, L., and Sławomir, B. (2012). Usage of digital image correlation in analysis of cracking processes. *Image Processing & Communications*, 17(3):21–28.

[Berfield et al., 2006] Berfield, T. A., Patel, J. K., Shimmin, R. G., Braun, P. V., Lambros, J., and Sottos, N. R. (2006). Fluorescent image correlation for nanoscale deformation measurements. *Small*, 2(5):631–635.

[Besnard et al., 2006] Besnard, G., Hild, F., and Roux, S. (2006). “finite-element” displacement fields analysis from digital images: application to portevin–le châtelier bands. *Experimental Mechanics*, 46(6):789–803.

[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.

[Blaber and Antoniou, 2017] Blaber, J. and Antoniou, A. (2017). Ncorr. `http://www.ncorr.com/`. [Online; accessed 2017-05-30].

[Brillaud and Lagattu, 2002] Brillaud, J. and Lagattu, F. (2002). Limits and possibilities of laser speckle and white-light image-correlation methods: theory and experiments. *Applied Optics*, 41(31):6603–6613.

[Bruck et al., 1989] Bruck, H. A., McNeill, S. R., Sutton, M. A., and Peters, W. H. (1989). Digital image correlation using newton-raphson method of partial differential correction. *Experimental Mechanics*, 29(3):261–267.

[Buck et al., 2004] Buck, I., Foley, T., Horn, D., Sugerman, J., Fatahalian, K., Houston, M., and Hanrahan, P. (2004). Brook for gpus: Stream computing on graphics hardware. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 777–786, New York, NY, USA. ACM.

[Chasiotis and Knauss, 2002] Chasiotis, I. and Knauss, W. G. (2002). A new microtensile tester for the study of mems materials with the aid of atomic force microscopy. *Experimental Mechanics*, 42(1):51–57.

[Chen et al., 1993] Chen, D., Chiang, F.-P., Tan, Y., and Don, H. (1993). Digital speckle-displacement measurement using a complex spectrum method. *Applied optics*, 32(11):1839–1849.

[Cheng et al., 2002] Cheng, P., Sutton, M. A., Schreier, H. W., and McNeill, S. R. (2002). Full-field speckle pattern image correlation with b-spline deformation function. *Experimental mechanics*, 42(3):344–352.

[Chu et al., 1985] Chu, T., Ranson, W., and Sutton, M. A. (1985). Applications of digital-image-correlation techniques to experimental mechanics. *Experimental mechanics*, 25(3):232–244.

[Correlated solutions, 2017] Correlated solutions (2017). Vic-2d^{TM} system. `http://correlatedsolutions.com/vic-2d/`.

[Dante Dynamics, 2017] Dante Dynamics (2017). Dante Dynamics DIC system. `https://www.dantecdynamics.com/digital-image-correlation`. [Online; accessed 2017-05-30].

[Davis and Freeman, 1998] Davis, C. Q. and Freeman, D. M. (1998). Statistics of subpixel registration algorithms based on spatiotemporal gradients or block matching. *Optical Engineering*, 37(4):1290–1298.

[Garcia et al., 2002] Garcia, D., Orteu, J., and Penazzi, L. (2002). A combined temporal tracking and stereo-correlation technique for accurate measurement of 3d displacements: application to sheet metal forming. *Journal of Materials Processing Technology*, 125:736–742.

[Gaudette et al., 2001] Gaudette, G. R., Todaro, J., Krukenkamp, I. B., and Chiang, F.-P. (2001). Computer aided speckle interferometry: a technique for measuring deformation of the surface of the heart. *Annals of biomedical engineering*, 29(9):775–780.

[Gembris et al., 2011] Gembris, D., Neeb, M., Gipp, M., Kugel, A., and Männer, R. (2011). Correlation analysis on gpu systems using nvidia's cuda. *Journal of Real-Time Image Processing*, 6(4):275–280.

[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.

[GOM, 2017] GOM (2017). Gom correlate software. `http://www.gom-correlate.com/`. [Online; accessed 2017-05-30].

[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.

[Helm et al., 1996] Helm, J. D., McNeill, S. R., and Sutton, M. A. (1996). Improved three-dimensional image correlation for surface displacement measurement. *Optical Engineering*, 35(7):1911–1920.

[Hild et al., 2002] Hild, F., Raka, B., Baudequin, M., Roux, S., and Cantelaube, F. (2002). Multiscale displacement field measurements of compressed mineral-wool samples by digital image correlation. *Applied optics*, 41(32):6815–6828.

[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.

[Hung and Voloshin, 2003] Hung, P.-C. and Voloshin, A. (2003). In-plane strain measurement by digital image correlation. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, 25(3):215–221.

[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].

[Kang et al., 2005] Kang, J., Jain, M., Wilkinson, D., and Embury, J. (2005). Microscopic strain mapping using scanning electron microscopy topography image correlation at large strain. *The Journal of Strain Analysis for Engineering Design*, 40(6):559–570.

[Khronos group, 2017a] Khronos group (2017a). Khronos group. `https://www.khronos.org/`. [Online; accessed 2017-05-30].

[Khronos group, 2017b] Khronos group (2017b). Opencl best practices. `http://www.nvidia.com/content/cudazone/cudabrowser/downloads/papers/nvidia_opencl_bestpracticesguide.pdf`. [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.

[Lancaster and Salkauskas, 1981] Lancaster, P. and Salkauskas, K. (1981). Surfaces generated by moving least squares methods. *Mathematics of computation*, 37(155):141–158.

[Leclerc et al., 2009] Leclerc, H., Périé, J.-N., Roux, S., and Hild, F. (2009). *Integrated Digital Image Correlation for the Identification of Mechanical Properties*, pages 161–171. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Lecompte et al., 2006] Lecompte, D., Smits, A., Bossuyt, S., Sol, H., Vantomme, J., Hemelrijck, D. V., and Habraken, A. (2006). Quality assessment of speckle patterns for digital image correlation. *Optics and Lasers in Engineering*, 44(11):1132 – 1145.

[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.

[Luo et al., 2005] Luo, J., Ying, K., He, P., and Bai, J. (2005). Properties of savitzky–golay digital differentiators. *Digital Signal Processing*, 15(2):122–136.

[Mahajan, 2004] Mahajan, A. (2004). Measurement of whole-field surface displacements and strain using a genetic algorithm based intelligent image correlation method.

[McCool et al., 2002] McCool, M. D., Qin, Z., and Popa, T. S. (2002). Shader metaprogramming. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware*, HWWS '02, pages 57–68, Aire-la-Ville, Switzerland, Switzerland. Eurographics Association.

[Meng et al., 2007] Meng, L., Jin, G., and Yao, X. (2007). Application of iteration and finite element smoothing technique for displacement and strain measurement of digital speckle correlation. *Optics and Lasers in Engineering*, 45(1):57–63.

[NVIDIA, 2017] NVIDIA (2017). Opencl programming guide for the cuda architecture. `http://www.nvidia.com/content/cudazone/download/opencl/nvidia_opencl_programmingguide.pdf`. [Online; accessed 2017-05-30].

[NVIDIA Corporation, 2007] NVIDIA Corporation (2007). *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. NVIDIA Corporation.

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

[Pan, 2009] Pan, B. (2009). Reliability-guided digital image correlation for image deformation measurement. *Applied optics*, 48(8):1535–1542.

[Pan et al., 2009] Pan, B., Asundi, A., Xie, H., and Gao, J. (2009). Digital image correlation using iterative least squares and pointwise least squares for displacement field and strain field measurements. *Optics and Lasers in Engineering*, 47(7):865–874.

[Pan et al., 2007] Pan, B., Xie, H., Guo, Z., and Hua, T. (2007). Full-field strain measurement using a two-dimensional savitzky-golay digital differentiator in digital image correlation. *Optical Engineering*, 46(3):033601–033601.

[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.

[Peters and Ranson, 1982] Peters, W. H. and Ranson, W. F. (1982). Digital imaging techniques in experimental stress analysis. *Optical Engineering*, 21(3):213427–213427–.

[Pitter et al., 2002] Pitter, M. C., See, C. W., Goh, J. Y., and Somekh, M. G. (2002). Focus errors and their correction in microscopic deformation analysis using correlation. *Optics express*, 10(23):1361–1367.

[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.

[Rost, 2005] Rost, R. J. (2005). *OpenGL(R) Shading Language (2Nd Edition)*. Addison-Wesley Professional.

[Schreier, 2003] Schreier, H. (2003). *Investigation of Two and Three-dimensional Image Correlation Techniques with Applications in Experimental Mechanics*. University of South Carolina.

[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].

[Shi et al., 2004] Shi, X., Pang, H., Zhang, X., Liu, Q., and Ying, M. (2004). In-situ micro-digital image speckle correlation technique for characterization of materials' properties and verification of numerical models. *IEEE transactions on components and packaging technologies*, 27(4):659–667.

[Singh and Omkar, 2013] Singh, A. and Omkar, S. (2013). Digital image correlation using gpu computing applied to biomechanics. *Biomedical Science and Engineering*, 1(1):1–10.

[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.

[Sjödahl, 1997] Sjödahl, M. (1997). Accuracy in electronic speckle photography. *Applied Optics*, 36(13):2875–2885.

[Sjödahl and Benckert, 1993] Sjödahl, M. and Benckert, L. (1993). Electronic speckle photography: analysis of an algorithm giving the displacement with subpixel accuracy. *Applied Optics*, 32(13):2278–2284.

[Smith et al., 2002] Smith, T. S., Bay, B. K., and Rashid, M. M. (2002). Digital volume correlation including rotational degrees of freedom during minimization. *Experimental Mechanics*, 42(3):272–278.

[Sun et al., 2005] Sun, Y., Pang, J. H., Wong, C. K., and Su, F. (2005). Finite element formulation for a digital image correlation method. *Applied optics*, 44(34):7357–7363.

[Sun et al., 1997] Sun, Z., Lyons, J. S., and McNeill, S. R. (1997). Measuring microscopic deformations with digital image correlation. *Optics and Lasers in Engineering*, 27(4):409–428.

[Sutton et al., 1986] Sutton, M., Mingqi, C., Peters, W., Chao, Y., and McNeill, S. (1986). Application of an optimized digital correlation method to planar deformation analysis. *Image and Vision Computing*, 4(3):143–150.

[Sutton et al., 1991] Sutton, M., Turner, J., Bruck, H., and Chae, T. (1991). Full-field representation of discretely sampled surface deformation for displacement and strain analysis. *Experimental Mechanics*, 31(2):168–177.

[Sutton et al., 2008] Sutton, M., Yan, J., Tiwari, V., Schreier, H., and Orteu, J. (2008). The effect of out-of-plane motion on 2d and 3d digital image correlation measurements. *Optics and Lasers in Engineering*, 46(10):746–757.

[Sutton et al., 2000] Sutton, M. A., McNeill, S. R., Helm, J. D., and Chao, Y. J. (2000). Advances in two-dimensional and three-dimensional computer vision. In *Photomechanics*, pages 323–372. Springer.

[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.

[Van Paepegem et al., 2009] Van Paepegem, W., Shulev, A. A., Roussev, I. R., De Pauw, S., Degrieck, J., and Sainov, V. C. (2009). Study of the deformation characteristics of window security film by digital image correlation techniques. *Optics and Lasers in Engineering*, 47(3):390–397.

[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.

[Wang et al., 2002] Wang, C. C., Deng, J.-M., Ateshian, G. A., and Hung, C. T. (2002). An automated approach for direct measurement of two-dimensional strain distributions within articular cartilage under unconfined compression. *TRANSACTIONS-AMERICAN SOCIETY OF MECHANICAL ENGINEERS JOURNAL OF BIOMECHANICAL ENGINEERING*, 124(5):557–567.

[Wang and Kang, 2002] Wang, H. and Kang, Y. (2002). Improved digital speckle correlation method and its application in fracture analysis of metallic foil. *Optical Engineering*, 41(11):2793–2798.

[Wang and Vo, 2017] Wang, Z. and Vo, M. P. (2017). Opticist.org 2d & 3d dic moire software. `http://www.opticist.org/node/73`. [Online; accessed 2017-05-30].

[Wang et al., 2007] Wang, Z. Y., Li, H. Q., Tong, J. W., and Ruan, J. T. (2007). Statistical analysis of the effect of intensity pattern noise on the displacement measurement precision of digital image correlation using self-correlated images. *Experimental Mechanics*, 47(5):701–707.

[Wattrisse et al., 2001] Wattrisse, B., Chrysochoos, A., Muracciole, J.-M., and Némoz-Gaillard, M. (2001). Analysis of strain localization during tensile tests by digital image correlation. *Experimental Mechanics*, 41(1):29–39.

[White et al., 2003] White, D., Take, W., and Bolton, M. (2003). Soil deformation measurement using particle image velocimetry (piv) and photogrammetry. *Geotechnique*, 53(7):619–632.

[Willert and Gharib, 1991] Willert, C. E. and Gharib, M. (1991). Digital particle image velocimetry. *Experiments in fluids*, 10(4):181–193.

[Xiang et al., 2007] Xiang, G., Zhang, Q., Liu, H., Wu, X., and Ju, X. (2007). Time-resolved deformation measurements of the portevin–le chatelier bands. *Scripta Materialia*, 56(8):721–724.

[Yamaguchi, 1981] Yamaguchi, I. (1981). Speckle displacement and decorrelation in the diffraction and image fields for small object deformation. *Journal of Modern Optics*, 28(10):1359–1376.

[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.

[Zhang et al., 2006a] Zhang, D., Luo, M., and Arola, D. D. (2006a). Displacement/strain measurements using an optical microscope and digital image correlation. *Optical Engineering*, 45(3):033605–033605.

[Zhang et al., 1999] Zhang, D., Zhang, X., and Cheng, G. (1999). Compression strain measurement by digital speckle correlation. *Experimental mechanics*, 39(1):62–65.

[Zhang et al., 2003] Zhang, J., Jin, G., Ma, S., and Meng, L. (2003). Application of an improved subpixel registration algorithm on digital speckle correlation measurement. *Optics & Laser Technology*, 35(7):533–542.

[Zhang et al., 2015] Zhang, L., Wang, T., Jiang, Z., Kemao, Q., Liu, Y., Liu, Z., Tang, L., and Dong, S. (2015). High accuracy digital image correlation powered by gpu-based parallel computing. *Optics and Lasers in Engineering*, 69:7–12.

[Zhang et al., 2006b] Zhang, Z.-F., Kang, Y.-L., Wang, H.-W., Qin, Q.-H., Qiu, Y., and Li, X.-Q. (2006b). A novel coarse-fine search scheme for digital image correlation method. *Measurement*, 39(8):710–718.

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

[Zhou and Goodson, 2001] Zhou, P. and Goodson, K. E. (2001). Subpixel displacement and deformation gradient measurement using digital image/speckle correlation (disc). *Optical Engineering*, 40(8):1613–1620.

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