National Technical University of Athens

The Moving Least Squares Method in Mesh Deformation - Implementation in CUDA/C++ & Performance analysis

by

Evangelos D. Katsavrias

Dissertation
In partial fulfillment of the requirements for the degree of
Master of Science
in *Computational Mechanics*

February 2019, Athens

Approval committee

Acknowledgements

I would like to thank my thesis advisor Professor Kyriakos C. Giannakoglou of the School of Mechanical Engineering at National Technical University of Athens, for giving me the opportunity to work on this interesting subject.

I would also like to thank the two other members of the approval committee, who offered their expertise to assess my thesis. I am also grateful to all the professors, the course instructors, and the director of the MSc. programme Professor Andreas Boudouvis, for their excellent work.

Finally, I must express my very profound gratitude to my parents, my sisters, and my friends for providing me with unfailing support and continuous encouragement through the process of this thesis.

Evangelos D. Katsavrias

Abstract

There are plenty applications in computational mechanics, e.g. in fluid flows around bodies, contact solid mechanics, fluid-structure interaction analysis, etc., where the mesh deformation is part of the process. These problems are quite computationally intensive and very often solved on GPGPUs. Therefore, a mesh deformation method that would perform well on the GPGPU parallel environment, is something desirable to reduce the overall solution time of the problem. Such a mesh deformation method seems to be the Moving Least Squares (MLS) method, which exhibits full data independence in its coarse granularity level of computations, and a moderate parallelization potential in its finer granularity of computations.

Prior work on the MLS method for mesh deformation, see Τουρής [2016], had exposed the effectiveness and the good performance of the method in mesh deformation. In this work, a parallel execution of the Moving Least Squares (MLS) method is developed, on General Purpose GPUs (GPGPUs). Initially, the MLS method's computations are analyzed to expose the potential parallelism with the CUDA compute execution model. Further, CUDA algorithms are proposed to execute the computations efficiently and gain an optimal speedup. Additionally, the MLS method is enriched with the more general rational weighting functions (i.e. inverse distance weighting functions), which allow the utilization of low polynomial degree MLS interpolation, preserving the good quality results for low-medium degree of mesh deformations, see Witteveen and Bijl [2009]. The low polynomial MLS interpolation is an interesting case, because the computations are substantially reduced and the parallelization potential is increased.

The results of the developed algorithms expose good compute performance with very high speedups for low polynomial degree MLS interpolations, e.g. up to x20 in the case of the zero degree MLS method (known also as Inverse distance weighting method). The speedups for higher polynomial degrees are moderate, i.e. x5, but the utilized hardware comprised of a very strong CPU and a GPGPU with poor compute resources.

Contents

Chapter 1

Introduction

1.1 Scope and applications of the mesh deformation techniques

Modelling and solving problems in mechanics where the the analysis domain is moving or its shape is changing, can be encountered frequently. Some of the problems, are the following:

- Shape optimization
- Fluid-structure interaction analysis
- Aeroelastic analysis
- Hydroelastic analysis
- Elastic bodies with contact interfaces
- Combination of the above problems

In single domain shape optimization problems, there is the need to adapt the computational mesh to the updated geometry shape. In multi-domain problems of moving domains, or shape changing domains, the additional requirement is to propagated the deformations from one domain to the neighbour domains and preserve the compatibility on the interface. In all the aforementioned problems, one basic need is emerged; to adapt the corresponding computational mesh to the updated domain shape and position. A straightforward way to adapt a computational mesh to the domain shape is a naive regeneration. Though, that would prove expensive for large meshes, especially when the solution of the problem implicates solution steps and/or time steps. Instead, it is preferable to adapt the mesh by rapid deformations (i.e. node displacements), such that the mesh quality would not be affected severely and the mesh topology would be preserved. Therefore, the mesh deformation methods are suggested to be used as an efficient alternative, which can result in an adapted mesh of good quality in low computational cost.

A concrete application of the mesh deformation technique, is found in shape optimization of bodies immersed in flowing fluids, under certain requirements for their aerodynamic behavior, see Jameson [1988], Jameson [1995], Zhang et al. [2019], Karpouzas et al. [2016]. For instance, an objective can be to minimize the drag forces on the immersed body under the fluid flow, or to maximize the lift forces, or to maximize the efficiency of the flow around a blade of a turbine. In such problems, a CFD analysis is iteratively executed to determine the fluid flow around the updated shape of the immersed body. Thus, the computational mesh of the fluid domain, needs to be adapted to the updated shape of the solid body in every optimization step. In each mesh adaptation, it is crucial to preserve the good quality of the mesh for fast convergence in the CFD analysis. Furthermore, using a mesh deformation method over a mesh regeneration, keeps the mesh topology unchanged. The latter fact allows the use of the previous step CFD solution, in the new launched CFD analysis as an initial one, which accelerates the convergence. Another interesting application of the mesh deformation technique, is found in the Fluid-Structure Interaction (FSI) analysis with the Arbitrary Lagrange-Eulerian formulation Donea et al. [1982]. In FSI analysis, the domain of the fluid changes due to the deformation of the structure, and it is inconvenient to have a fixed fluid mesh. The issue is solved by using the Arbitrary Lagrangian-Eulerian (ALE) formulation, also known as the dynamic mesh formulation. In the ALE formulation the fluid mesh moves in compliance with the boundary deformation, therefore it is neither Eulerian (fixed) nor Lagrangian (moving with material particles).

1.2 Mesh deformation methods

The goal of every mesh deformation method is to compute the displacements of the free nodes (e.g. the interior mesh nodes) when the displacements of a number of fixed nodes are given (e.g. the boundary mesh nodes with prescribed displacements). The governing rule under which the node displacements should be evaluated, is the optimal quality of the resulted mesh, or equivalently, the objective is the minimum distorsion of the mesh elements. Other, application dependent requirements, can be the preservation of the orthogonality on the boundaries, the high rigidity of the elements close to the boundaries, etc.

The minimum distorsion requirement, is approached differently in each mesh deformation method. Nevertheless, two main classes can be identified. The first class, consists of methods where a continuous displacement field is constructed, under the constraint to satisfy the displacements at some given points and the basic requirement of low local distorsions. The displacements of the free nodes, are probed from the continuous displacement field, as soon as the two domains are overlapping. The methods of the second class, are expressing the distorsion of the mesh elements explicitly, through certain geometric quantities of the mesh (e.g. element angles, edge lengths, etc.) which are linked directly to the node displacements. These geometric quantities are interrelated within the mesh, and they allow for the expression of constraint equations, which subsequently can lead to a system of equations of all the unknown nodal displacement components.

Finally, the implementation of each method, should be based on computational models which are able to provide the required output, for some given input data and constraints, in a consistent and robust manner. In the subject computational models, the consistency is related to the prescribed node displacements and the required smoothness, and the robustness is referring to the sustainability to various displacement modes, as required from the applications.

In the following subsections, the most important mesh deformation methods are presented. For further reading, Selim and Koomollil published recently an extensive survey on the existing mesh deformation methods (Selim and Koomullil [2016]). There, the most significant techniques are reviewed and compared, allowing a researcher to adopt the most efficient scheme or to propose improvements.

1.2.1 Methods based on a continuous deformation field

The first class of methods, is based on existing abstract computational models, which evaluate scalar fields or vector fields while satisfying the following conditions: a) some prescribed values on part of the domain (e.g. boundary values, values on sample points, etc.), and b) some value variations with respect to the spatial variables (e.g. partial differential equations, distributions, etc.). These computational models can be subclassed as following:

1) The computational model is based on a mathematical model of an appropriate PDE, which describes smooth fields, while complying to a given boundary and some prescribed boundary values (i.e. a Dirichlet problem), and combined with a certain computational method (e.g. weighted residual methods, finite differences, finite volumes, etc.). Concrete computational models of this family, are the one developed for elasticity problems (i.e. biharmonic PDEs), and the corresponding mesh deformation methods are termed as elastic pseudo-structural, or physical analogy methods.

2) The computational model is a general function fitting method, which is able to produce a sufficiently smooth function that interpolates some given sample data. For better results, some of these methods are imitating/approximating numerically the general solution of a Dirichlet problem, or a specific computational method solving a specific Dirichlet problem, while being parametrized by any number of sample data.

The common feature of these methods, is that the unknown displacement field is found from a linear space of functions, which is spanned by a basis of primative functions. Thus, the coefficients of the linear combination of the basis functions, are the only unknown parameters to be determined. Another common feature of these methods, is that the topological data of the mesh are not needed, and they can apply to any type of mesh, i.e. structured, unstructured or hybrid.

What makes special each method, is the different type of basis functions that defines a different function space of solutions. Furthermore, the different optimality expressions (e.g. on the variations of the displacement field, or the interpolation of the given data) result a different linear system of equations of the unknown parameters.

Some of the mesh deformation methods of this class are presented and described in the following text. Other methods not explained below, are the Algebraic Damping method proposed by Zhao and Forhad (Zhao and Forhad [2003]), and for structured meshes the Transfinite Interpolation (TFI) method that is quite efficient and suggested for many applications with moving boundaries Sen et al. [2017].

1.2.1.1 Elastic pseudo-structural methods

It is a physical analogy method, where the domain spanned by the subject computational mesh is seen as an elastic body. The linear elasticity field equations (which can combine and sum up to a biharmonic equation), are describing a highly smooth displacement field, from which the free mesh nodes can inquire the displacement values, under the constraint to satisfy the prescribed Dirichlet boundary conditions and/or other given node displacements (Lynch and O'Neil [1980]). The deformation can be controlled with certain physical parameters that describe the body stiffness in various affine transformations (e.g. shear, normal stretches, volume change, etc.). Several improvements has been proposed from Stein et al. [2003], for the best combination of the elastic parameters (e.g. the Lame parameters), in order to achieve very large deformations. Another improvement, is to variate the stiffness parameters for selected elements (e.g. small elements, or elements close to the boundaries), to preserve some mesh properties. The elastic problem is treated numerically with the finite elements method on the same subject mesh, and the components of the vector displacement field are coupled. Thus, a very large system of equations needs to be solved, but the stiffness matrix is sparse, and allows for efficient memory storage models and solvers (e.g. direct or iterative with preconditioning).

1.2.1.2 Methods describing the displacement field with general PDEs

The elastic pseudo-structural methods, based on a biharmonic PDE, are just a concrete example of the abstract class of PDEs, which can be used to describe an appropriate displacement field with low local distortions. Using the biharmonic operator to describe a field, larger deformations can be allowed compared to second order PDEs (e.g. based on the Laplacian operator) (Helenbrook [2003]). Furthermore, the biharmonic equations have the advantage, against the Laplacian, to preserve the orthogonality of the mesh on the boundaries. The main drawback of the methods based on a biharmonic description, is the high computational cost.

The Laplacian operator is another efficient suitable way to result a smooth mesh deformation (Burg [2006]). The traditional Laplacian method is based on the Laplace equation, though the modified Laplacian, incorporating a diffusion coefficient, can perform better by adjusting the coefficient value. The diffusion coefficient value, can be variable and based on the distance from the boundaries, or based on a mesh property (i.e. orthogonality and skewness). Finally, the method can handle very large deformations of single frequency (i.e. rigid translations or rotations), by adjusting appropriately the exponent of the diffusion coefficient (Selim and Koomullil [2016]).

1.2.1.3 Function fitting methods with Radial basis functions

The Radial Basis Functions (RBF) method (de Boer et al. [2007]), is a function fitting method, where the fitting process is taking place globally, for the full set of the nodes with prescribed displacements (i.e. fixed nodes). The prescribed data of each fixed node, applies an influence on the resultant fitted function, which influence follows a distribution over the domain that fades out smoothly as the distance from the subject fixed node increases. The distribution of the influence (i.e. on the fitted field) is described with a radial function, and the intensity (magnitude) of the influence is an unknown parameter to be evaluated. The resultant fitted function (i.e. unknown field) is described as the sum of the influences from all the fixed nodes, i.e. the linear combination of the evaluated radial functions on a certain domain point, with coefficients the corresponding magnitudes of the radial functions. One radial function is centered on each fixed node, spanning over the domain with a sufficiently large radius, such that it includes a sufficient number of fixed nodes. Every fixed node with its linked fixed nodes, falling in the radial function span, will co-influence a certain subdomain, which is defined by the intersection of the corresponding radial functions' span. In case where the radius of the functions is small, the occured fitting process is more local, allowing only few fixed nodes to effect on points in the domain, as well as assigning small influence for fixed nodes far from the close vicinity. The latter fact, facilitates a better and smoother interpolation around the boundaries. In contrast, too small influence radius can result a non robust fitting method, resulting poor smoothness far from the fixed nodes or even failing to give a solution.

The unknown magnitude of the influence of each fixed node, is determined after constructing a linear system of equations. The number of equations is equal to the number of the unknowns, i.e. the number of the fixed nodes, where each equation is the field description (through the sum of the influences) satisfied on one of the fixed nodes by interpolation. The latter process is frequently termed as "network trainning", where the term network refers to the expressed interaction of the fixed nodes.

The RBF method can result dense system matrices when the influence radius is large, such that a large number of fixed nodes are co-influencing, or sparse when the radius is reduced. Thus, the method can be fast with efficient solvers for sparse matrices, whereas the method becomes slow for a large number of boundary nodes and large influence radii. The efficiency of the method can be further improved with iterative solvers and preconditioning. Further efficiency improvements are reported in Selim and Koomullil [2016].

The RBF method is easy to implement and the resulted mesh quality is good preserving the orthogonality on the boundaries. Finally, the quality of the results depends on the type of the radial basis functions and the chosen influence radius.

1.2.1.4 Free form deformation methods (FFD)

The FFD methods are describing and controlling a domain (i.e. an infinite closed set of points) through a finite set of control points (Sederberg and R. Parry [1986]). The FFD methods in mesh deformation, are describing directly the domain where the computational mesh is defined (e.g. the fluid domain), through a set of control points placed on the boundaries (static or dynamic). Thus, as the mesh nodes are a finite subset of the domain points, their position is given directly through the FFD description. The FFD descriptions are based on linear function spaces, parametrized by the control points (i.e. boundary points) and spanned by a basis of an equal size. An important requisite property for the used linear function space, is the linear combination convexity, i.e. the combination coefficients should be non-negative and their sum at any point within the domain should equal to one. The latter property allows for convex descriptions, such that the control lattice formed by the control points, being the convex hull of the described domain. Futhermore, to achieve smooth desctiptions, the basis functions are chosen to be sufficiently smooth, e.g. harmonic functions (Μαυρονικόλα [2017], Ζέρβας [2018]), Bernstein polynomials (Becker et al. [2011], Παπαδημητράκης [2016]), etc.

1.2.1.5 Local function fitting methods with weighted least squares (WLS) and moving Least Squares (MLS)

The WLS and MLS methods are both very well known methods, with applications in regression analysis, function fitting, numerical analysis of PDEs, image morphing, etc. The two methods implement a generalized linear least squares method, which results a smooth function fitted to some given data (i.e. a displacement field fitted to some given displacements in mesh deformation), by a linear combination of smooth basis functions. The generalization of the method comes from that the minimized squared quantities are weighted in a desired way by a distribution (i.e. a kernel, or a mask function). In fact, the ordinary least squares method is a special case of the WLS, where the weighting distribution is a constant function. The improved results of applying the two methods, are the outcome of choosing a weighting function with local support (span), i.e. positive on part of the domain and zero (or almost zero) on the rest of the domain. These weighting functions are cutting off the influence of far distant data, resulting a locally fitted function that is relied on a subset of the most closely distant data. The degree of the locality of the fitting, is controlled through the weighting function span and the speed that it vanishes. If the weighting function vanishes very fast from a defined center point in the domain, then the resulted least squares fitting is considering only the very close distant data around the center point. Furthermore, in order to keep smooth the resultant field function, which is comprised of several local evaluations, the weighting function is required to vanish very smoothly as approaching the bounds of the local support. As for a short mathematical insight to the method, the local fitting of the displacement field to the prescribed displacements, is the solution of a minimization of the total weighted interpolation error (i.e. expressed in the L^2 norm). Searching for a solution in a linear function space, i.e. parametrizing the solution function with a finite number of basis functions and unknown coefficients, the minimization problem can be solved with the Ritz method, from a set of linear equations known as *normal equations*. Considering a function space spanned by the monomial basis, one of the unknown coefficients represents the field value (i.e. the coefficient corresponding to the constant function) on the center of the weighting function, which is the domain point where the field value is queried. The rest of the evaluated parameters, deliberate the corresponding local derivatives of the field function, but in an omni-directional sense.

The MLS method (Lancaster and Salkauskas [1981], Bos and Salkauskas [1989]), implements an infinite number of the above described local fittings for a given continuous domain, i.e. for every point of the domain an individual local least squares fitting is performed. The latter fact implies, that performing a local fitting of a function with a basis of monomials, only the coefficient corresponding to the constant function is in interest, which represents the field value at a specific domain point.

The WLS method, implements the local least squares fitting on a finite number of points within the domain, termed as stationary points (Nealen [2004]). The evaluated data on the stationary points, i.e. the field value and some derivatives, constitute the data to evaluate the field values at any other point in the domain. As a method for these evaluations, it is proposed (Nealen [2004]) the MLS method of 0-degree polynomial (i.e. the constant function is the only member of the basis functions), using as data only the one available on the stationary points.

The MLS method applied in mesh deformation, performs one local least squares fitting per adapted mesh node and per displacement component (i.e. 2 evaluations per node for a 2D mesh and 3 evaluations for a 3D mesh). Moreover, for the evaluation of a specific displacement component on a node, i.e. for a point evaluation of a scalar field function, only the data of the same displacement component are considered. Furthermore, to apply the MLS method in mesh deformation, an appropriate weighting function and a span size should be selected. Small span sizes and rapid decaying weighting functions, enhance the locality of the function fitting, which improves the interpolation to the given displacements and the mesh quality in the vicinity of the fixed nodes (i.e. the mesh nodes with prescribed displacements). Though, small weighting function spans can cause high distorsions away from the fixed points, or numerical accuracy problems when all the points are assigned very small weights, or even lack of robustness, especially when a higher degree polynomial is used and the data points are sparse long distant. Nevertheless, when the nodes with prescribed displacements are short distant, and the displacements follow highly variable patterns, small span sizes and/or higher degree polynomials are necessery to interpollate the given data and avoid high distorsions around the fixed nodes.

A special case of the MLS method, where the solution function space is spanned by one basis function, i.e. the constant function, and the weighting function is an inverse distance function (IDW), is proved to be very efficient, thus attracting a high interest for many applications (Gordon and Wixom [1978], McLain [1974]). This method has been first proposed from Shepard (Shepard [1968]) for surface reconstruction applications by interpolating the given sample points in 3D, and is frequently called the Shepard's method. In this method, the IDW functions can achieve a perfect interpolation, due to the very high weighting values that explode to infinity as approaching the center of the distribution, and the rapid decay of the weighting values as getting away from the center. The evaluation of such a local least squares fitting is simplified to the weighted average method, which is computationally very cheap. In mesh deformation, the method has been proposed from Witteveen (Witteveen and Bijl [2009], Uyttersprot [2014], Sen et al. [2017]) demonstrating very good mesh qualities for a low computational cost. Additionally, this very simplified form of the MLS method, facilitates greatly the algorithm design in parallel processing, requiring minimal communication between the computing threads and allowing the efficient linear speed up. Moreover, this special case of the MLS method, because of the very rapid decaying inverse distance weighting functions and the constant fitted function, results a zero variation of the field values in the close proximity of the nodes with prescribed displacements (clamping effect). Therefore, in the most common cases where the prescribed displacements are on the boundary nodes, it results meshes where the elements close to the boundaries preserve their initial shape, i.e. the close distant mesh nodes are rigidly clamped on the boundaries.

The MLS method is used in many other applications, in computational mechanics (Nayroles et al. [1992], Belytschko et al. [1994], Liu [2002], Quaranta et al. [2005]) and physics (Maisuradze et al. [2003]). Furthermore, it is used in general where a geometry deformation or image morphing (Schaefer et al. [2006]) is required, e.g. CAGD, computer graphics, animation industry, medical image processing, etc. Other applications are where a function reconstruction is required, based on some sample data, with applications in statistical modeling (Cleveland and Devlin [1988], Cleveland [1979], Strutz [2010]), geo-sciences, geometry reconstruction, etc.

1.2.2 Methods based on explicit mesh distortion constraints

The mesh deformation methods of this class are based on concepts where the mesh quality is controlled explicitly, through geometrical quantities and imposed constraints. For instance, such constraints can force the mesh elements to follow rigid motion modes as rigid particles, to satisfy an equilibrium of fictitious forces and moments (physical analogy methods), constraints on the angles between the edges, etc. Thus, some of the mesh deformation methods in this class are:

a) the rigid body motion method,

b) the linear springs analogy method,

c) the rotational springs analogy method,

d) the delaunay graphs method.

The above methods can function for structured or unstructured meshes.

1.2.2.1 Springs analogy methods

In this method the mesh is modeled as a system of springs being joint on the nodes. The mesh deformation is the result of the solution of a system of equations, which describe the equilibrium of the interconnected nodes, for given displacements of some of the nodes (e.g. the boundary nodes displacement). There are two pseudo-structural approaches to interrelate the nodes with springs of appropriate stiffness, limiting their relevant displacement. In the first approach, each edge within the mesh is emulated with a linear spring of stiffness inverse proportional to the length (Batina [1991]), i.e. short lengthed edges are assigned with higher stiffness value. Alternatively, each angle between two adjacent mesh edges, is assigned with a rotational spring of stiffness proportional to the declination of the angle from the proper angle of the element (e.g. 60deg for triangular elements, 90deg for quadrilateral elements, etc.). The two approaches, are the base of the spring analogy methods, i.e. the linear spring analogy and the rotational spring analogy methods (Farhat et al. [1998], Degand and Farhat [2002]). The method of the linear springs is easy to apply, but it performs poorly under big displacements and dense meshes, degenerating the mesh elements. In such cases, a method of rotational springs analogy with corrections on the boundary nodes, will perfom better to avoid the mesh elements' degeneration J. Blom [2000], Αποστόλου [2015].

1.2.2.2 Adaptation with the Rigid body movement model

This method is implemented with the assistance of fictitious cells formed around each interior node that need to be adapted. The method is of a physical analogy type, as each cell is modeled as a perfectly rigid particle (Κοντός [2018]). For each interior node, the adjacent circumferential neighbor nodes are found to form a cell, which cell is forced to be rigid and undeformed under the mesh deformation process. In other words, the goal is to displace the cells without changing their shape, thus it is algebraically expressed some constraint equations on the nodes displacements, which are solved for the unknown node displacements. The resulted mesh quality is similar to the initial, and the method is fast enough although it is iterative (Κοντός [2018]).

1.2.2.3 Adaptation with Delaunay graphs

The mesh deformation method with Delaunay graphs parametrization (Delaunay graph mapping method), is an algebraic method proposed from Liu (Liu et al. [2006]). The method requires an ascillary initial coarse graph that will be defined on the given mesh elements. It is important to create an initial graph with as high as possible good quality in order to sustain large deformations, i.e. the graph elements has to be as close as possible to the proper shape. Though, such a proper shape of the graph elements is generated through the Delaunay triangulation procedure, where one of the fundamental properties is that the minimum angle per graph element is the maximum possible.

The nodes of the given mesh, mapped to the barycentric points of the elements of the ascillary graph, and the connectivities between the mesh nodes, are sufficient topological data for the ascillary graph. The new positions of the boundary displacements of the deformed mesh, are redefining the ascillary graph, but the mapped topological data are preserved. Thus, the new positions of the barycentric points of the graph elements are the required adapted positions of the deformed computational mesh. The mesh deformation method using Delaunay graphs can result meshes of good quality efficiently, even for large deformations Τσολοβίκος [2018].

1.3 Introduction to high performance computing and parallel processing

There are many applications which can effectively use computing speeds in the trillion operations per second range. Some of these are:

1) Numerical simulation to predict the behaviour of physical systems.
2) High performance graphics, e.g. visualization, animation, etc.
3) Big data analytics for strategic decision making and finance.
4) Synthesis of molecules for designing medicines.

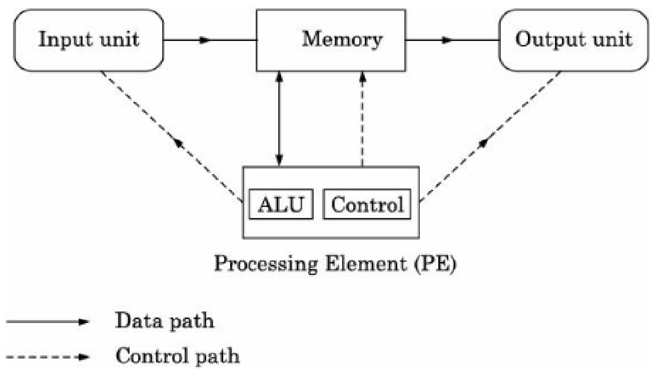


Figure 1.1: The von Neumann computer architecture.

There are two methods of increasing the speed of computers. One method is to build sequential processing elements (scalar processors) using faster semiconductor components (frequency rate), and the other is to improve the architecture of computers using parallel models, i.e. temporal (or task) parallelism and/or data parallelism. The temporal parallelism implies the decomposition of the process in several sub-processes (e.g. instructions), which can be executed by individual specialized execution units in an overlapped manner (e.g. instruction level parallelism, pipeline streams, multi-threading, memory hierarchy, data transferring bus lines, etc.), for as long as no data dependencies occur between the overlapped sub-processes. Data parallelism is achieved by executing the same process (no matter the operation complexity, e.g. an instruction, or a program, etc.) on n sets of independent operands, simultaneously on n corresponding execution units. Such processing units to execute entire programs (i.e. coarse grain of parallelism) can be inter-connected cores, CPUs, GPUs, multiprocessor servers or computer clusters, known as parallel computers. When instruction level parallelism exists in a micro-processor, then examples of execution units are the ALUs, FPUs, LSUs, BEUs, AGUs, etc., which can be installed multiple times in a core to implement further temporal or data parallelism. The latter type of parallelism is known in the literature as very fine grained parallelism, as the smallest grain is a machine instruction in a program.

The speed of a sequential computer (see Figure 1.1) is limited by the speed at which a processor can retrieve instructions and data from the memory, and the speed at which it can process the retrieved data. To increase the speed of processing of data, one may increase the speed of the processors by increasing the clock speed. Though, the heat dissipation in the processors is directly analog to the frequency, which puts a limit to the frequency at which processors operate, which is known as the frequency wall in the literature.

In 1965 Moore had predicted that the number of transistors in an integrated circuit chip will double (per unit area of chip) approximately every two years, and without increasing the product cost (known as the Moore's law). The Moore's prediction and the actual advances are a result of constant tecnhological developement (e.g. advances in lithography process), market and economy laws, and more precisely nowdays, complemented from the Kimmey's Law, Dennard's Law, Metcalfe's Law and the Emergent Behavior (Hutcheson [2018]). The increase in the number of transistors allowed processor designers to install more execution units on chip, combined with control units, achieving complex pipelined, superscalar, multi-threaded processors that exploited the available instruction level parallelism (ILP). However, even if many arithmetic elements are equipped in one processor, there was not enough instruction level parallelism available in threads to use them. Further as the complexity of the processor increased design errors crept in, debugging complex processors on a single chip with too many transistors was difficult and expensive. Thus, instead of using the extra transistors to increase the number of integer units, floating point units, and registers, which reached a point of diminishing returns, architects had to explore other ways of using the extra transistors. There are two other ways of using transistors. One is to increase the size of the on-chip memory called the L1 cache to exploit memory hierarchy speedups, and the other is to put several processors in a single chip and make them cooperate to execute a program. There is a limit beyond which increasing the L1 cache size has no advantage as there is an upper limit to locality of reference to memory. Nevertheless, installing n processors (processing cores) in a chip, can very often yield $n-fold$ speedup and in some special cases even super-linear speedup. The number of cores has been increasing in step with the increase in the number of transistors in a chip, i.e. the number of cores is doubling almost every two years. An architecture with multiple cores, implements an on-chip shared memory for the parallel working cores, or a network of cores, where each core has its own memory.

1.3.1 Parallel computers

The computer speed relies mostly on the processor speed, though other computer units, namely, the memory, and I/O units, can contribute when their speed is increased, or the computer architecture is improved. For example, while the processor is computing, data can be transferred asynchronously from the computer memory to the processor memory and from the I/O units to the computer memory, for later use. Such an overlap of operations is achieved by using both software and hardware features. To further increase the processing speed, many such computers may be inter-connected to work cooperatively to solve a problem. Research in parallel computing started in the late 70s, by examining how several independent processors could be inter-connected to work in parallel. Many scientific groups and companies were active in building parallel machines during the 80s and 90s upto early 2000, but parallel computing had never been used for general purpose (Rajaraman and Murthy [2016]). It was the multiple cores architecture, by implementing parallel computing paradigms, that brought the parallel computing in everyday use. Nevertheless, in each case of processor architecture, computer architecture, or parallel computer architecture, the main subject is to improve the inter-connection network of the sub-units, by implementing temporal and data parallelism.

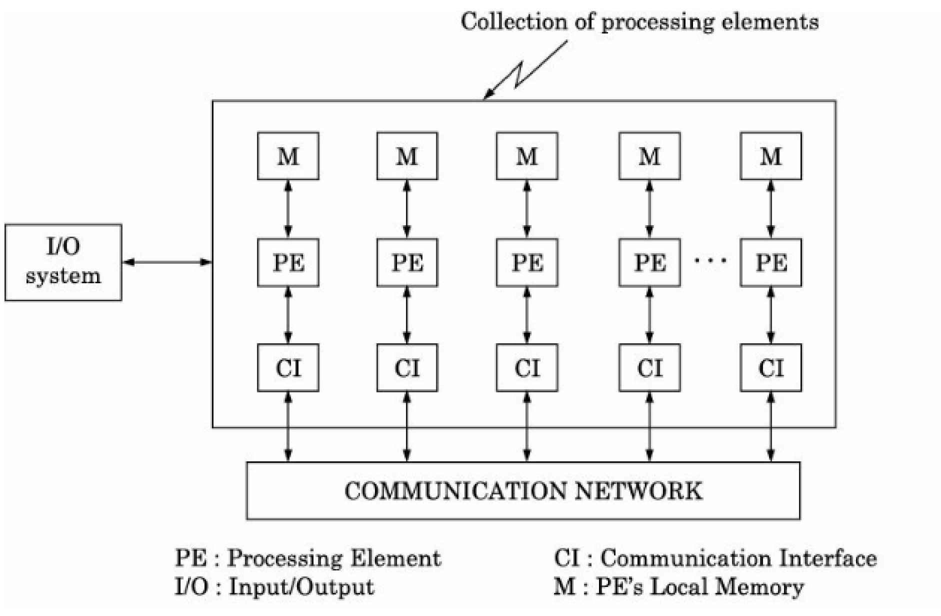


Figure 1.2: General diagram of a parallel computer (Rajaraman and Murthy [2016]).

The heart of a parallel computer is a set of processing elements (PE) interconnected by a communication network (see Figure 1.2), and the general structure is complemented by PE local memory and an I/O system. This general structure can have many variations based on the type of PEs, the memory available to PEs to store programs and data, and how memories are connected to the PEs, the type of communication network used and the technique of allocating tasks to PEs and how they communicate and cooperate (Rajaraman and Murthy [2016]). The variations in each of these lead to a rich variety of parallel computers.

The vast variety of parallel computer architecture may be classified based on the following criteria:

1) How do instructions and data flow in the system? This idea for classification is extensively used in literature, being also one of the earliest proposals, known as Flynn's classification (Flynn [1972]).

2) What is the coupling between PEs? Coupling refers to the way in which PEs cooperate with one another. The autonomy enjoyed by the PEs while cooperating with one another during problem solving determines the degree of coupling between them. A tightly coupled parallel computer, shares a common main memory, thus communication among PEs is very fast and cooperation may be even at the level of instructions.

3) How do PEs access memory? Accessing relates to whether data and instructions are accessed from a PE's own private memory or from a memory shared by all PEs.

4) What is the quantum of work done by a PE before it communicates with another PE? This is commonly known as the grain size of computation. The grain sizes are classified as very fine grain (a single machine instruction), fine grain (a thread, i.e. a very small sequence of machine instructions, managed independently), medium grain (e.g. a subroutine, or a procedure less than 1000 machine instructions), and coarse grain (a complete program). The grain size determines the frequency of communication between PEs during the solution of a problem.

Flynn classified parallel computers into four categories based on how instructions process data. A computer with a single (scalar) processor with no temporal and data parallelism, is called a Single Instruction stream, Single Data stream (SISD) Computer. This type of computer implements the von Neumann architecture. A class of computers which have multiple processors is the Single Instruction stream, Multiple Data stream (SIMD) computer (see Figure 1.3). These computers (e.g. array computers) expose data parallelism, but no temporal parallelism, i.e. a specific instruction is issued (dispatched) to every processor from one central control unit (instruction scheduler), and every processor operates on a different set of operands, in a lock-step-fashion (simultaneously) using data from their local memory. In this model there is no explicit communication among processors. However, data paths between nearest neighbours, or grids, are used in some structures to allow local communication. SIMD computers are used to solve many problems in science which require identical operations to be applied to different data sets synchronously, e.g. adding and multiplying the elements of a set of arrays simultaneously. If instead of a single instruction, the PEs are issued a program to process multiple data streams, such a parallel computer is called a Single Program Multiple Data (SPMD) Computer. The third class of computers according to Flynn's classification, is known as Multiple Instructions stream, Single Data stream (MISD) computers. Observe in Figure 1.4 that in this structure multiple instruction schedulers exist, where each one is controlling a different stream of instructions dispatched to an individual PE, and operating on shared data. The MISD parallel computing model, fascilitates the implementation of temporal parallelism, e.g. the pipeline processing is a special case of this mode of computing. In pipeline processing (e.g. in vector computers) the data processed by PE1 (see Figure 1.4), namely, R1 is fed to PE2, R2 to PE3 etc., and DM contents will be input to only PE1. This type of processor may be generalized using a 2-dimensional arrangement of PEs, which is known as a systolic processor (e.g. VLSI circuits). The last and the most general model, according to Flynn's classification, combines the temporal and the data parallelism, and is termed as the Multiple Instructions stream, Multiple Data stream (MIMD) computer (see Figure 1.5). MIMD is the most frequent architecture for the mainstream processors, and parallel computers (e.g. Message passing multi-computer, cluster computers, shared memory computer using bus or interconnection network communication, distributed shared memory systems, etc.).

The classification based on the mode of accessing memory, finds the class of Shared Memory (SM) computers, and the class of Distributed Shared Memory (DSM) computers. In a

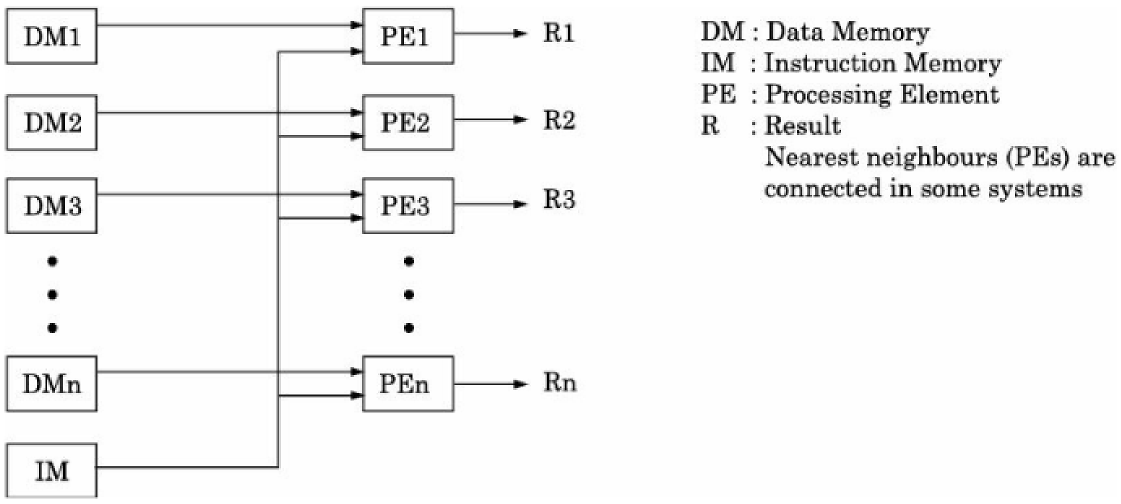


Figure 1.3: General diagram of a SIMD computer (Rajaraman and Murthy [2016]).

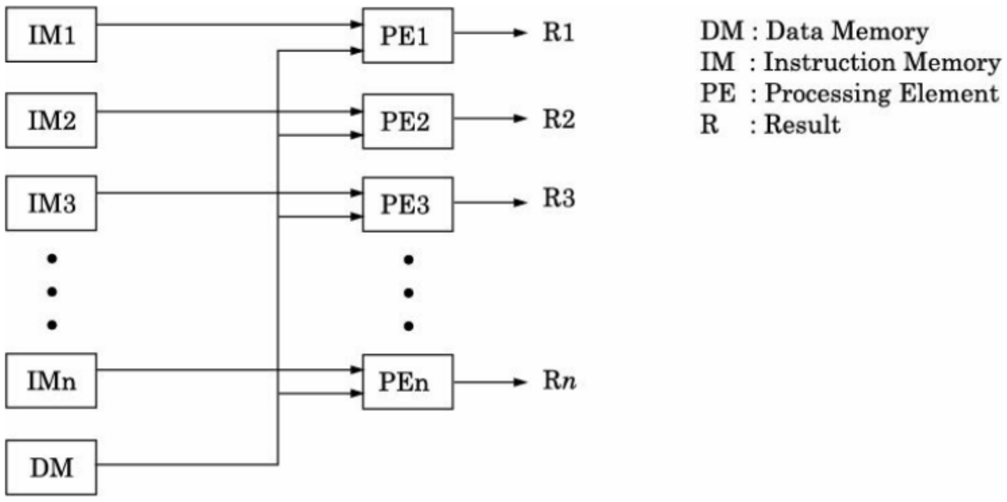


Figure 1.4: General diagram of a MISD computer (Rajaraman and Murthy [2016]).

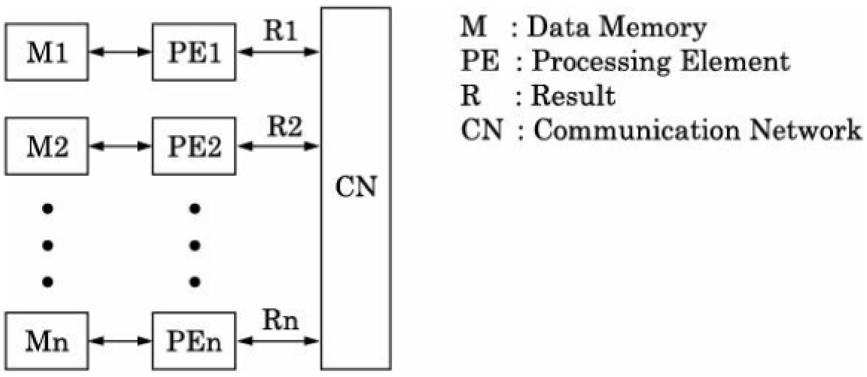


Figure 1.5: General diagram of a MIMD computer (Rajaraman and Murthy [2016]).

shared memory computer all the processors share a common global address space, or in other words, programs are written for this parallel computer assuming that all processors address a common shared memory. The latter memory model has a Uniform Memory Access

(UMA), i.e. the time to access a word in the memory is constant for all the processors. Shared memory machines may be of the bus-based, extended, or hierarchical type. In distributed shared memory (DSM) systems, each processor may have its own local memory and may or may not share a common memory. A distributed shared memory computer has a Non Uniform Memory Access (NUMA), explaining, the time taken to access a word in its local memory is smaller than the time taken to access a word stored in the memory of another computer or a common shared memory. For example, if a remote memory is accessed by a PE using a communication network, it may be 10 to 1000 times slower than accessing its own local memory. Distributed memory machines may have hypercube or mesh interconnection schemes.

1.3.2 Chip Multiprocessors

When the performance of single processors reached saturation around 2005, chip designers started integrating many processors (cores) on a single chip to continue to build processors with better performance. The parallel processors built on a single integrated circuit chip, and following known architectures from parallel computers, are called Chip Multiprocessors (CMP), allowing a core level parallelism to speedup execution of programs, or to execute a number of independent programs simultaneously. However, there are important differences between the architectural issues in designing a parallel computer of independent processing units, and designing a CMP. On the negative side, individual cores heat up at high clock frequency and may affect the performance of physically adjoining cores. Another problem arises due to threads or processes running on different cores and sharing an on-chip cache memory, leading to contention and performance degradation unless special care is taken both architecturally and in programming. Furthermore, whereas in parallel computers wires interconnecting computers had no constraint on the number of wires, the wires inside a chip occupy chip area and thus their number has to be reduced. Also, the switches used for inter-processor communication need chip space and their complexity has to be reduced. Considering in advantages, as the processors are closely packed, the delay in transferring data between cooperating processors is small. Thus, very lightweight threads (i.e., single machine instruction or very short sequence of machine instructions, managed independently) can run in the processors and communicate more frequently without degrading performance. As cores are constantly duplicated, the design of a single core may be replicated without incurring extra design cost. Besides this, depending on the requirement of the application, the power budget, and the market segment, processors may be tailored with a necessary number of cores without incurring a high cost.

Figure 1.6 illustrates a generalized structure of CMPs, where a set of processing cores interconnected by a communication network is the standard essential part of the chip. The independent processing cores share common on-chip cache memory to cooperate and solve problems in parallel. The general structure given in Figure 1.6 can have many variations (similar to parallel computers) based on the type of cores, the memory system used by the cores, the type of communication system interconnecting the cores, and how the cores communicate and cooperate.

The type of the processing cores, can vary from one CMP to another, or from core to core within a particular CMP. Some of the most common core processor types are, the single threaded scalar processor (SISD, von Neumann computer), a multithreaded superscalar processor, a simplistic processor consisting only of an ALU, a general purpose processor, etc.

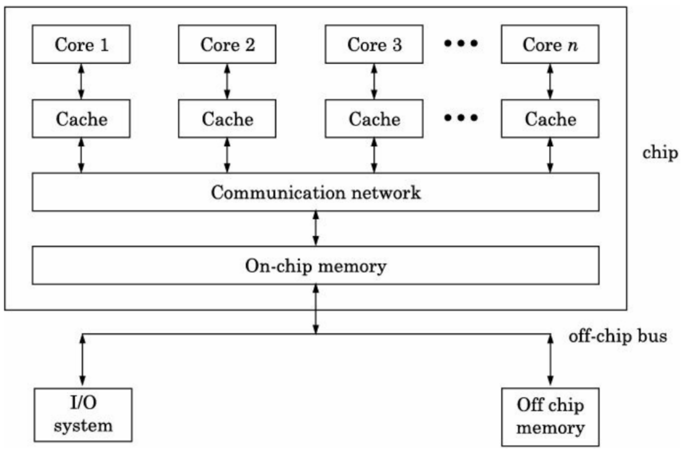


Figure 1.6: A typical architecture of a chip multiprocessor (CMP). (Rajaraman and Murthy [2016])

As mentioned above, CMPs are designed according to known parallel computing models, and some of them are described below,

a) the SPMD model, operates in parallel all the cores with copies of the same running program, but on different data sets,

b) the SIMD, according to which an instruction dispatcher issues (broadcasts) a single instruction to all the supervised core processors, i.e. copies of an instruction are dispatched, but with different data per core.

c) the SIMT (single instruction, multiple threads) model, is a generalization of the SIMD model that was innovated from NVIDIA for their general purpose core processors known as SMs (shader or streaming multiprocessors). In a SIMT architecture, rather than a single thread issuing vector instructions applied to data vectors (SIMD model), multiple threads issue common instructions to arbitrary data. The benefits of SIMT for programmability led NVIDIA's GPU architects to coin a new name for this architecture, rather than describing it as SIMD.

d) the MIMD model, where all the instructions and the data are stored in the shared memory and managed by scheduled threads. A free core selects a thread to execute and deposits the results back in the shared memory for use by other cores. each core is assigned an independent task, and all the cores work simultaneously on the assigned tasks, known as the request level parallel processing.

Concerning the memory system, there are few memory hierarchy patterns used, of 2 or more cache memory levels. Such patterns variate the location of the higher (further) cache levels, i.e. the L2 cache can be placed locally in each core, or shared among the cores and installed on-chip. If L3 cache memory exists, it can be placed either on-chip, when the L2 cache memory is on core, or off-chip. Shared cache memories (or off-chip memories) reduce the occupied on-chip area and power consumption, allowing for larger memory capacities and the cooperation of the processors. From the other hand, cache memories placed in processing cores accelerate the processing of independent tasks per core. With the chips being populated with more and more transistors, higher capacities of cache memories are possible, and higher level cache memories can immigrate on-chip closer to the processing cores. The use of independent caches from the cores, leads to the cache coherence problem, i.e. inconsistency in the stored data from a cache memory level to another level. The problem of cache coherence is solved with various techniques developed for parallel computing architectures, by updating the data of a given address (unified memory address space) in all the cache memories. The memory consistency model and the need to write race free programs are also applicable to CMPs.

The design of communication networks among the cores and the shared memories in CMPs, follows various architectures from parallel computers, e.g. a bus interconnection, or a ring bus interconnected cores (with caches), a fixed interconnection network such as a ring, or a grid, or crossbar switch with appropriate routers. A point of difference, compared to the communication networks of conventional parallel computers, is that in CMPs a bus or an interconnection network is an integral part of a single microprocessor chip, offering higher speeds and lower latencies.

Nevertheless, the most difficult problem is to perceive parallelism in algorithms and develop a software environment which will enable application programs to utilize this potential parallel processing power. Unlike multithreaded processors in which the hardware executes multiple threads without a programmer having to do anything, multicore processors require careful programming. The problem of the programming model in GPU multi-core processors, where the number of cores is very large (many-cores processors), is specially handled with a large pool of virtual threads that are programmed in the software level. The large pool of virtual threads are managed by an efficient hardware execution model (i.e. ATI/AMD Ultra-Threaded Dispatch and GCN system, or NVIDIA GigaThread and CUDA system), which in first priority services the need of high processing throughput and dynamic branching.

1.3.3 High compute throughput with GPGPUs

Graphic processor units (GPUs) since '90s offered a full acceleration for 2D and 3D graphics, dismissing any workload from the CPU. They utilized a classical processing pipeline, illustrated in Figure 1.7, which is still valid without considerable changes.

The graphics pipeline contains all the necessary stages from the moment that the vertex data arrive to the GPU until a pixel is finally drawn on the screen. Initially, each stage of the pipeline was supported by specialized (fixed-function) parallel processing elements on chip. The parallel processing elements per pipeline stage, were designed to operate with high throughput on a large amount of independent data (e.g. vertices, or fragments-pixels), covering the workload per stage (i.e. traditionally the fragment-pixel stage had a much larger number of elements to process, compared to the vertex stage).

In the early 2000, there was a demand from GPUs to offer more complex and flexible processing features in the most important pipeline stages, i.e. the vertex processing stage and the fragments-pixel processing stage. Therefore, the corresponding processing units became gradually programmable, starting with the vertex stage by implementing the programmable parallel vertex shaders, and then the programmable parallel pixel shaders. A constant demand for better control and more dynamic flow over the graphics pipeline, was pushing the GPU architectures to unify the programmable shaders, into general purpose parallel processors, that could be shared dynamically for the various graphics pipeline stages, see Figure 1.8. The latter GPU architectures with unified parallel processors, were developed along with dynamic multi-threading execution systems (i.e. ATI/AMD Ultra-Threaded Dispatch

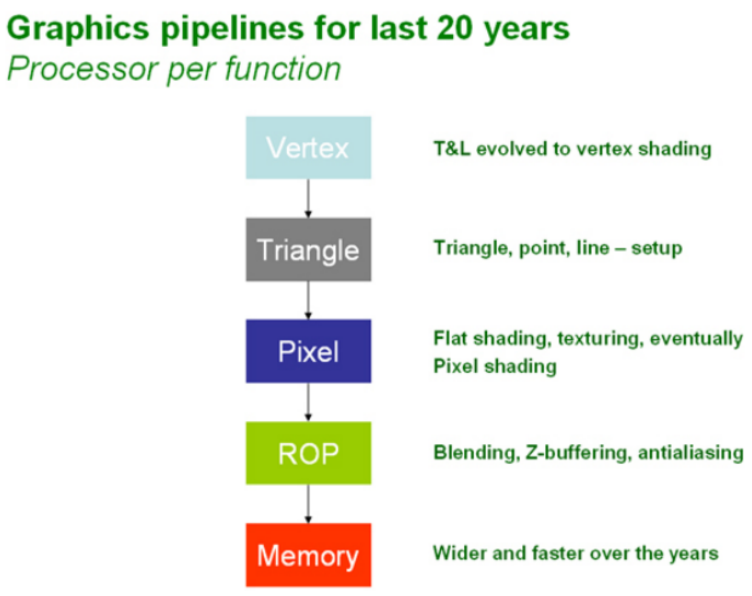

Graphics pipelines for last 20 years
Processor per function
Vertex
Triangle
Pixel
ROP
Memory
T&L evolved to vertex shading
Triangle, point, line – setup
Flat shading, texturing, eventually Pixel shading
Blending, Z-buffering, antialiasing
Wider and faster over the years

Figure 1.7: Classic graphics pipeline with a specialized processor per function. (NVIDIA)

and GCN system, or NVIDIA GigaThread and CUDA system), which in first priority are servicing the need to retain the processing throughput in the highest possible level at every moment, while sharing the available compute resources among many data and many pipeline stages.

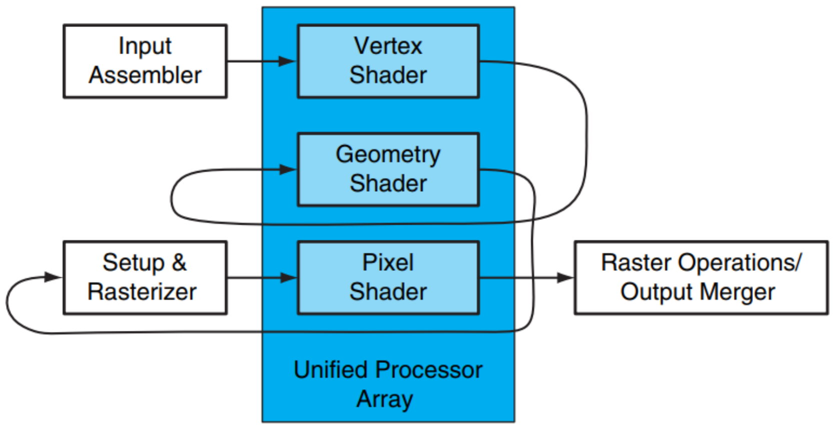


Figure 1.8: Unified shader processor array, executing the logical graphics pipeline stages. (Patterson and Hennessy [2013])

The dynamic multi-threading execution systems are implemented by many flying virtual threads, which are in standby to be instantiated by hardware threads for execution, when the scheduling units will find the optimum time instant. In a typical system, thousands of threads are queued up for work. If the GPU must wait on one group of threads, it simply begins executing work on another. The flying virtual threads concept, facilitates greatly the parallel programming models (e.g. CUDA, OpenCL, etc.), by letting the execution control of the hardware computing threads to the hardware itself (or compiler sometimes), and maintaining a scalable software for future or current hardware with variant amount of compute resources. These abstract programming models, combined with the general purpose parallel processors compliant with the IEEE 754 standard (a standard of precision and numerical accuracy of floating point operations), gave the ability to use the GPUs as general purpose processing units (GPGPUs) for any sufficiently parallel problem. GPGPU computing is the use of a GPU as a co-processor to accelerate CPUs for general-purpose computing. The GPU accelerates applications running on the CPU by offloading some of the compute-intensive portions of the code where data are independent and the operations are the same. Additionally, computations on GPGPUs demonstrate a high potential for good energy efficiency.

Parallel computing devices such as GPGPUs, tend to be hit for any single threaded application and frequently are poor performers for extremely branch-intensive, unpredictable or too small problems. Very small problems lack the parallelism needed to use all the threads on the GPU and/or could fit into a low-level cache on the CPU, substantially boosting CPU performance. Unpredictable problems have too many meaningful branches, which can prevent data from efficiently streaming from GPU memory to the cores or reduce parallelism by breaking the SIMD parallel model that exists in the fine grain execution model of the GPGPUs. This problems does not exist in MIMD parallelism, which is implemented for the coarse grain execution model of the GPGPUs, i.e. each process/instruction has its own 'locus of control'.

GPU chip multiprocessors, which originate from the special-purpose graphics co-processors, are optimized for data-parallel tasks with simpler control logic, focusing on the throughput of parallel programs. These programs in graphics, are based on a data-parallel map idiom of large single-value data. Thus, other applications which possess the map idiom, basically map operating on multiple-dimension arrays, are ideal for peak performance. Loops on regular data structures is another example of map idiom, offering high parallelism degree, even for fine grain calculations. The GPGPUs are not necessarily limited to map computations, stencils may be target computations, as well as complex reduce and parallel prefix. GPGPUs rely heavily on the underlying architecture with shared memory, so many stencils can be modeled as map & barrier.

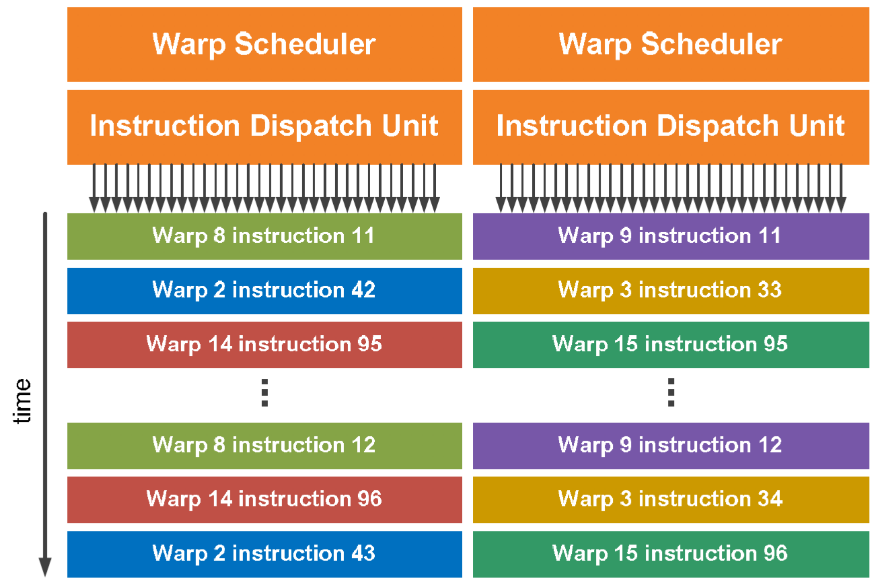
GPU multi-core processors are heavily populated with cores, and are known as many-core processors. The many core architecture is organized in a two level hierarchy that focuses primarily on achieving high compute throughput on data parallel workloads, by sacrificing single-thread performance and execution latency. This is in contrast to general purpose processors, and multi-core architectures, which focus primarily on single-thread performance, with low execution and communication latency, with a secondary focus on high compute throughput.

The GPGPUs architectures, implement a coarse grain parallel model among clusters of execution units, and a fine grain parallel model among the execution units within each cluster. Both of the parallel granularity levels are participating to achieve high compute throughput, through the dynamic multi-threading execution system, as introduced in the text above. NVIDIA names the clusters of the execution units stream multiprocessors (SM), and AMD calls them compute units (CU).

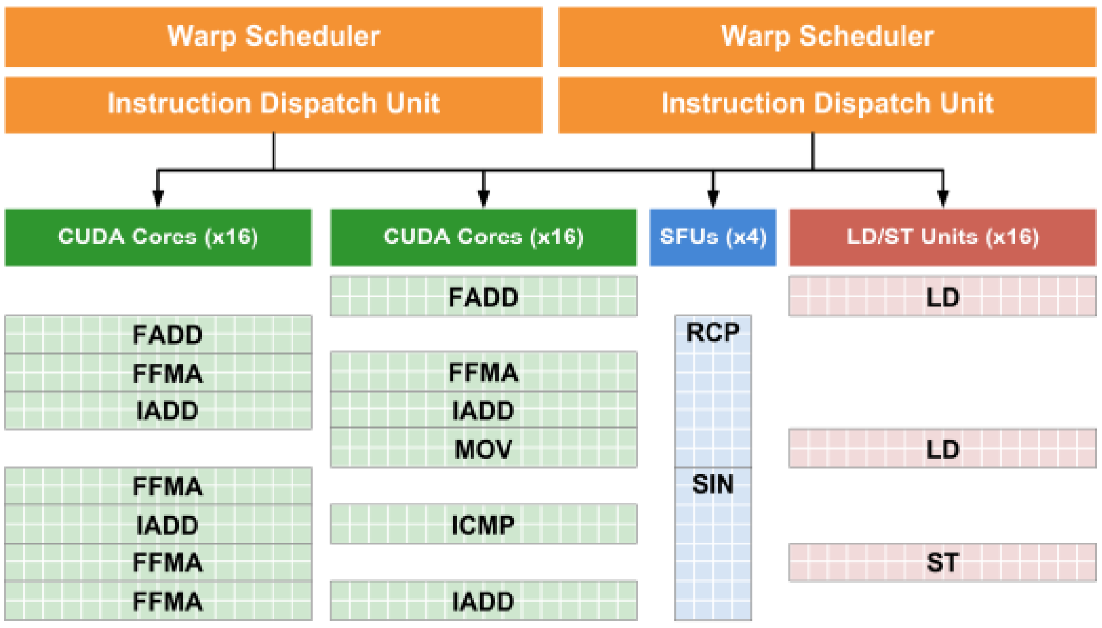
A CU or a SM loosely corresponds to a core processor in a modern microprocessor. The GPGPU chip multiprocessors consist of, core processors (i.e. SMs or CUs) on an interconnection network, high-bandwidth DRAM channels, on-chip L2 cache, and a dynamic virtual thread manager. The number of SMs and cores per SM, varies as per the price and target market of the GPU. Among the core processors, a bus interconnection is usually preferred, as it is simple and easy to fabricate, but it can become a bottleneck for a many-core processor, when many of them will try to access the cache memory on chip simultaneously. Therefore, an interconnection network is preferred, with a message passing cooperation system to avoid the cache coherence problem, or a shared memory architecture with a ring bus and directory based cache coherence protocol (Rajaraman and Murthy [2016]). GPGPUs coarse grain parallel computing executes with the MIMD parallel model. The execution model is implemented with dynamic managers of groups of virtual threads (i.e. the ATI/AMD Ultra-Threaded Dispatch system, or the NVIDIA GigaThread system), which try to keep a workload equilibrium among the SMs or CUs. The managed groups of virtual threads are known as *thread blocks*. The thread blocks are distributed to the SMs (or CUs) for further brakedown structuring in groups of, *warps* in CUDA terminology, or *wavefronts* in the AMD's GCN architecture, which groups have a hardware fixed size. The warps (or wavefronts) are executed concurrently in a fine grained parallel fashion of the SIMT model.

Each core processor (i.e. SM or CU) has its own central front-end, complete with instruction fetch-decode-issue logic (i.e. warp schedulers and instruction dispatch units), a large amount of execution units, registers, L1 cache memory, and an interconnection network. NVIDIA and AMD like to call their arithmetic execution units (ALUs/FPUs) 'cores', so that they can claim to have hundreds of cores. In reality, GPU chip multiprocessors have more like tens of cores, i.e. the SMs or CUs, but are able to provide more computate power by using vectors, which lower the amount of the control overhead. However, it is clear that Stream Processors (SP) in NVIDIA's terminology, are not truly independent processor cores. Each SP has a register file (at least a portion of one) and an independent instruction pointer, but the SPs lack a complete front-end that can fetch and schedule instructions independently. In that regard, the SPs most closely correspond to an issue pipeline in a modern multi-threaded CPU. The fine grain parallel computing is implemented with the SIMT (generalization of SIMD) parallel model, with a fixed number of hardware threads instantiating the virtual flying threads (i.e. context switching) and dispatching a common instruction to the execution units, per processor clock cycle. For instance, in CUDA the latter cluster of hardware threads consists of 32 threads. The corresponding CUDA clusters of 32 virtual threads, which are instantiated at a clock time are called *warps*. Thus, the virtual flying threads are not totally unordered, but structured in groups of a fixed number of members, i.e. the warps. The threads in the warps should ideally possess the same sequence of instructions, as they are executed under the SIMT model. If there are threads in a warp with divergent instructions, these instructions will be issued only for the active threads, while the rest will remain idle as waste compute resources. This thread diverging problem requires careful programming, accounting for how the virtual threads are grouping in warps. The grouping of the virtual threads is augmented by multi-dimensional grid structures, where each virtual thread has a flying identity that corresponds to its position in the structured grid. In that grid, which is a fundamental element of the CUDA programming model, 32 consecutive virtual threads are comprising a warp. Each warp has its own program counter to keep a register of which instructions need to be issued for execution, common and divergent in total. Furthermore, every warp is able to access its own registers, to load and store from divergent addresses, and to follow divergent control flow paths. When a warp is being instantiated by the hardware threads for its current instruction dispatch at a certain clock cycle, at the next clock cycle it may proceed in dispatching its next instruction (if any remains), or may not. The decision is taken by a central instruction control unit (known as a warp scheduler in CUDA architecture), which analyzes a number of concurrent warps, currently residing in the SM. Thus, warps will reside in the SM, until they finish the issuing of their instructions, and will be instantiated according to a schedule (i.e. context switching) in various clock cycles, see Figure 1.9a. Current CUDA architectures support multiple (e.g. 2 or 4) warps instantiation per clock cycle, and furthermore, multiple (e.g. 2) consecutive instructions to be issued for execution from each instantiated warp, when possible. These, allow higher performance with instruction level parallelism, i.e. instruction pipelining, and superscalar execution (i.e. IPC> 1, when enough execution units are available). The execution units in each SM are structured in blocks, e.g. in Figure 1.9b the Fermi SM execution blocks are illustrated, where we see two blocks of 16 ALUs (arithmetic logic units) each, one block of four SFUs (special function units) and one block of 16 LSUs (load/store units). Therefore, for the Fermi SM, a total of 32 instrcutions from one or two warps can be dispatched in each cycle to any two of the four execution blocks within the SM. In case where the available number of execution units is greater or equal to the size of the warp (i.e. 32), and the warp threads do not diverge in the execution path, all the threads execute the scheduled instruction in one clock cycle, offering the maximum throughput. Otherwise, the number of cycles increases accordingly.

Employing the SIMT execution model, the control and logic hardware units get simplified and centralized to service a large number of simple ALUs that lack complex control and logic. The simple processing elements, need less transistors and integrated chip area, thus allowing to install a large number of them, resulting a substantially increased processing throughput.



(a) Illustrated are the instruction dispatching pipelines, one per warp scheduler, as issued from the central instructions control unit and the private program counter of each warp. (Source: NVIDIA)



(b) The figure shows the structure of the execution units in blocks, within a Fermi SM. The dispatched warp instructions are issued for execution per clock cycle in clusters, of size equivalent to the execution block. (NVIDIA)

Figure 1.9: In the two pictures above, the instructions dispatching mechanism is shown, along with the block structures of the execution units.

1.4 Work scope and outline

The various applications mentioned in section 1.1, where the mesh deformation is part of the process, are quite computationally intensive and very often solved on GPGPUs. Therefore, a mesh deformation method that would perform well on the GPGPU parallel environment, is something desirable to reduce the overall solution time of the problem. Such a mesh deformation method seems to be the MLS method, which exhibits full data independence in its coarse granularity level of computations, and a moderate parallelization potential in its finer granularity of computations.

Prior work on the MLS method for mesh deformation, see Τουρής [2016], had exposed the effectiveness and the good performance of the method in mesh deformation. The subject of this work is to develop a parallel execution of the Moving Least Squares (MLS) mesh deformation method, on General Purpose GPUs (GPGPUs). Initially, the MLS method's computations are analyzed for various polynomial degrees, to expose the potential parallelism with the CUDA compute execution model. Further, CUDA algorithms are proposed to execute the computations efficiently and gain an optimal speedup.

Additionally, the MLS method is enriched with the more general rational weighting functions (i.e. inverse distance weighting functions), which allow the utilization of low polynomial degree MLS interpolation, preserving the good quality results for low-medium degree of mesh deformations, see Witteveen and Bijl [2009]. The low polynomial MLS interpolation is an interesting case, because the computations are substantially reduced and the parallelization potential is increased.

The breakdown structure of this work is as following:

1. The second chapter introduces to the reader the mathematical background of the MLS method, combined with elements from the approximation theory which can be found in Appendix A. This mathematical background is essential to realize that the MLS method is a least squares method, with the ability to fit a function locally by cutting off the effect of far data. The elements from the approximation theory are useful to comprohend numerous publications on the MLS method with many applications.
2. The third chapter contains a thorough analysis of how the computations of the MLS method can fit and exploit the CUDA architecture. Furthermore, three variants of CUDA algorithm designs are presented to implement the MLS method, which are utilized in some test cases as presented in chapter four.
3. The fourth chapter demonstrates the parallel execution speedups that result each of the CUDA algorithm designs for some test cases. Additional results are included for simple parallelizations with CPU multithreading-multicore hardware and heteregeneous execution (CPU+GPGPU).

Chapter 2

The least squares methods

2.1 Introduction

In this chapter the mathematical background is shown for the family of the least squares methods. This chapter is augmented with many references to the Appendix A, where various important elements from approximation theory are presented.

The mathematical tool of this work, the Moving Least Squares method (MLS), is shown to be a special implementation of the generalized weighted least squares. Using appropriate weighting functions, the generalized weighted least squares can perform local approximations, which is the key element of the MLS method.

The importance of the weighting function in the results of the MLS and WLS methods is recongnised, especially in cases where the methods are required to interpolate the data. Further analysis on the weighting functions can be found in Appendix B, along with several parametric analyses which exposing the effect of the various function parameters on the quality of the interpolation. The intepolation property of the MLS, has been proved quite essential in our problem of mesh deformation, improving substantially the results in the close vicinity of the boundaries, even with the 0-degree polynomial.

2.2 Least Squares Approximation

The least squares approximation, is a method that gives the best approximation of a function, or the best approximation of a set of nodes. The best approximation implies an objective function, which minimizes the total approximation error. The objective functions can actually be any reliable metric functions, measuring the total approximation error, and can prove robust in a minimization process. In the case of the least squares method, that objective function is the norm-2 metric. The latter metric function, is associated with the inner product operation, thus when minimized, the orthogonal projection of the total error on the solution space is implied. More details can be found in Davis [1975] and Atkinson and Han [2009], or in Appendix A.2 and A.3.

Least Squares Approximation
Let $\mathcal{C} = L^2(\bar{\Omega})$ be a closed space, on $\Omega = \mathbb{R}^d$, equiped with the norm $\|.\|_2$, and $f \in \mathcal{C}$. Assume $\mathcal{S} = \mathbb{P}_n$ is a finite-dimensional convex subset of $\mathcal{C}$ with $n \geq 0$ and a basis $span\{\phi_0, \phi_1, ..., \phi_n\}$. Then there is a unique best approximation $P_S(f) = \hat{f}_b = \sum_{i=0}^{n} a_i\phi_i \in \mathbb{P}_n$, defined by a unique set $\{a_i\}_{i=0}^{n}$, $a_i \in \mathbb{R}$, from the minimization problem

$$\min_{\hat{f}\in\mathbb{P}_n} \left\| f - \hat{f} \right\|_2^2 \tag{2.1}$$

$$\Leftrightarrow E(f; a_i) = \min_{a_i} \int_{\bar{\Omega}} \left| f(x) - \sum_{i=0}^{n} a_i\phi_i(x) \right|^2 dx \tag{2.2}$$

The necessary condition to find the set $\{a_i\}_{i=0}^{n}$, which minimizes the above equation, is yielding a set of $n+1$ in number equations, known as *normal equations.*

$$\begin{aligned} \frac{\partial E(f; a_i)}{\partial a_m} &= 0, \quad m = 0, 1, 2, ..., n \\ \Leftrightarrow \frac{\partial}{\partial a_m} \int_{\bar{\Omega}} \left| f(x) - \sum_{i=0}^{n} a_i\phi_i(x) \right|^2 dx &= 0 \\ \Leftrightarrow \int_{\bar{\Omega}} \sum_{i=0}^{n} \phi_m(x) a_i \phi_i(x) dx &= \int_{\bar{\Omega}} \phi_m(x) f(x) dx \end{aligned} \tag{2.3}$$

The normal equations in Eq. 2.3 can be expressed in a matrix form

$$\mathbf{A}\mathbf{a} = \mathbf{f}$$

where $\mathbf{A}$ is the following Gramian matrix

$$\mathbf{A} = \int_{\bar{\Omega}} \begin{bmatrix} \phi_0(x)\phi_0(x) & \phi_0(x)\phi_1(x) & ... & \phi_0(x)\phi_n(x) \\ \phi_1(x)\phi_0(x) & \phi_1(x)\phi_1(x) & ... & \phi_1(x)\phi_n(x) \\ ... & ... & ... & ... \\ \phi_n(x)\phi_0(x) & \phi_n(x)\phi_1(x) & ... & \phi_n(x)\phi_n(x) \end{bmatrix} dx \tag{2.4}$$

and the vectors $\mathbf{a}$ of the set of unknowns a_i and $\mathbf{f}$ of the RHS

$$\mathbf{f} = \int_{\bar{\Omega}} \begin{bmatrix} f(x)\phi_0(x) \\ f(x)\phi_1(x) \\ ... \\ f(x)\phi_n(x) \end{bmatrix} dx, \quad \mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{bmatrix}$$

For function reconstruction problems, we have the following corollary for the best approximation with distinct data provided.

Least Squares Approximation on Distinct Points
Let $\mathcal{C} = L^2(\bar{\Omega})$ be a closed space, on $\Omega = \mathbb{R}^d$, equiped with the discrete norm $\|.\|_2$, and $f \in \mathcal{C}$. Assume $\mathcal{S} = \mathbb{P}_n$ is a finite-dimensional convex subset of $\mathcal{C}$ with $n \geq 0$ and a basis *span*$\{\phi_0, \phi_1, ..., \phi_n\}$. Let a set of sample values $\mathcal{F} = \{f(\hat{x}_i)\}_{i=1}^{M}$ is given on the set of sample points $\mathcal{P} = \{\hat{x}_i\}_{i=1}^{M}$, with $M \geq n+1$. Then there is a unique best approximation $P_S(f) = \hat{f}_b = \sum_{i=0}^{n} a_i\phi_i \in \mathbb{P}_n$, defined by a unique set $\{a_i\}_{i=0}^{n}$, $a_i \in \mathbb{R}$, from the minimization problem

$$\begin{aligned} &\min_{\hat{f}\in\mathbb{P}} \left\| f - \hat{f} \right\|_2^2 \qquad (2.5)\\ \Leftrightarrow &\min \sum_{i=0}^{M} |f(\hat{x}_i) - \sum_{j=0}^{n} a_j\phi_j(\hat{x}_i)|^2 \end{aligned}$$

The necessary condition to find the set $\{a_i\}_{i=0}^{n}$, which minimizes the above equation, is yielding a set of $n+1$ in number equations, known as *normal equations.*

$$\begin{aligned} &\frac{\partial E(f;a_i)}{\partial a_m} = 0, \quad m = 0,1,2,...,n \qquad (2.6)\\ \Leftrightarrow &\frac{\partial}{\partial a_m}\left[\sum_{i=0}^{M} |f(\hat{x}_i) - \sum_{j=0}^{n} a_j\phi_j(\hat{x}_i)|^2\right] = 0\\ \Leftrightarrow &\sum_{i=0}^{M}\sum_{j=0}^{n} \phi_m(\hat{x}_i)\phi_j(\hat{x}_i)a_j = \sum_{i=0}^{M} f(\hat{x}_i)\phi_m(\hat{x}_i) \end{aligned}$$

The Gramian matrix of the discrete normal equations is defined as

$$\mathbf{A} = \sum_{i=0}^{M} \begin{bmatrix} \phi_0(\hat{x}_i)\phi_0(\hat{x}_i) & \phi_0(\hat{x}_i)\phi_1(\hat{x}_i) & ... & \phi_0(\hat{x}_i)\phi_n(\hat{x}_i)\\ \phi_1(\hat{x}_i)\phi_0(\hat{x}_i) & \phi_1(\hat{x}_i)\phi_1(\hat{x}_i) & ... & \phi_1(\hat{x}_i)\phi_n(\hat{x}_i)\\ ... & ... & ... & ...\\ \phi_n(\hat{x}_i)\phi_0(\hat{x}_i) & \phi_n(\hat{x}_i)\phi_1(\hat{x}_i) & ... & \phi_n(\hat{x}_i)\phi_n(\hat{x}_i) \end{bmatrix} \qquad (2.7)$$

and the vectors $\mathbf{a}$ of the set of unknowns a_i and $\mathbf{f}$ of the RHS

$$\mathbf{f} = \sum_{i=0}^{M} \begin{bmatrix} f(\hat{x}_i)\phi_0(\hat{x}_i)\\ f(\hat{x}_i)\phi_1(\hat{x}_i)\\ ...\\ f(\hat{x}_i)\phi_n(\hat{x}_i) \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_0\\ a_1\\ ...\\ a_n \end{bmatrix}$$

□

2.3 Weighted Least Squares approximations

The classic least squares approximation, as introduced in the previous section with the minimization of the total approximation error, can be generalized. The generalization is introduced with a weighting mask function applied on the measured errors per point. In fact, the classic least squares method is a special case of the generalized, where the weighting mask function is the global unit function that considers every point error with the same importance in the total error.

Weighted Least Squares Approximation
Let $\mathcal{C} = L^2_w(\bar{\Omega})$ be a closed space, on $\Omega = \mathbb{R}^d$, equiped with the norm $\|.\|_{0,w}$, and $f \in \mathcal{C}$. Assume $\mathcal{S} = \mathbb{P}_n$ is a finite-dimensional convex subset of $\mathcal{C}$ with $n \geq 0$ and an orthogonal basis $span\{\phi_0, \phi_1, ..., \phi_n\}$. $w(\Omega)$ is a positive function and integrable on $\bar{\Omega}$. Then there is a unique best approximation $P_S(f) = \hat{f}_b = \sum_{i=0}^n a_i\phi_i \in \mathbb{P}_n$, defined by a unique set $\{a_i\}_{i=0}^n$, $a_i \in \mathbb{R}$, from the minimization problem

$$\min_{\hat{f}\in\mathbb{P}_n} \left\| f - \hat{f} \right\|^2_{0,w} \tag{2.8}$$

$$\Leftrightarrow E(f; a_i) = \min_{a_i} \int_{\bar{\Omega}} w(x) \left| f(x) - \sum_{i=0}^n a_i\phi_i(x) \right|^2 dx \tag{2.9}$$

The necessary condition to find the set $\{a_i\}_{i=0}^n$, which minimizes the above equation, is yielding a set of $n+1$ in number equations, known as *normal equations.*

$$\frac{\partial E(f; a_i)}{\partial a_m} = 0, \quad m = 0, 1, ..., n \tag{2.10}$$

$$\Leftrightarrow \frac{\partial}{\partial a_m} \int_{\bar{\Omega}} w(x) \left| f(x) - \sum_{i=0}^n a_i\phi_i(x) \right|^2 dx = 0$$

$$\Leftrightarrow a_m = \frac{\int_{\bar{\Omega}} \phi_m(x)w(x)f(x)dx}{\int_{\bar{\Omega}} \phi_m(x)^2 w(x)dx}$$

The discrete case is similar to (2.6) and (2.10)

$$\frac{\partial E(f; a_i)}{\partial a_m} = 0, \quad m = 0, 1, ..., n \tag{2.11}$$

$$\Leftrightarrow \frac{\partial}{\partial a_m} \left[\sum_{i=0}^M w(\hat{x}_i)|f(\hat{x}_i) - \sum_{j=0}^n a_j\phi_j(\hat{x}_i)|^2 \right] = 0 \tag{2.12}$$

$$\Leftrightarrow \sum_{i=0}^M \sum_{j=0}^n \phi_m(\hat{x}_i)w(\hat{x}_i)\phi_j(\hat{x}_i)a_j = \sum_{i=0}^M \phi_m(\hat{x}_i)w(\hat{x}_i)f(\hat{x}_i) \tag{2.13}$$

$$\Leftrightarrow a_m = \frac{\sum_{i=0}^M \phi_m(\hat{x}_i)w(\hat{x}_i)f(\hat{x}_i)}{\sum_{i=0}^M \phi_m(\hat{x}_i)^2 w(\hat{x}_i)} \tag{2.14}$$

2.4 Local Least Squares approximations

When we are dealing with problems that a global single polynomial approximation, gives insufficient quality of results, local approximations are employed for better accuracy and stability. Many of the local approximations are piecewise polynomials over a set of subdomains. Though, there are alternative methods that necessitate only given points in the domain, and have a great success in many computational fields. Here we introduce the local least squares approximation on weighted inner product function spaces, which is the base for the weighted local least squares (WLS) and moving least squares (MLS) methods. Both of the methods give a global approximation for a function f, through several local approximations over a set of given domain points.

The concept of the presented local approximations, is based on the weighted inner product $(.,.)_w$ operating on the residual $E(\hat{f})$, as introduced in the previous sections with an appropriately selected weighting function w. Such an appropriate weighting function should be semi-positive in a compact support, integrable, and rapidly decaying. Hence, trying to give a notion of the method in simple words, projecting the error function $E(\hat{f})$ on a compactly supported weighting function w, will result a weighted error on a compact subdomain. Subsequent minimization of the latter weighted residual, results a local(compact) approximation.

In the rest of the text, the compact support of the weighting function will be represented by an infuluence ball $B(\bar{x}, r)$,

$$B(\bar{x}, r) = \{x \mid x \in \Omega,\ \|x - \bar{x}\|_2 \leq r\}$$

where $\bar{\Omega} \subset \mathbb{R}^d$ is the given d-dimensional domain of a function f, $r \in \mathbb{R}^+$ is the half span defined as a parameter of the weighting function, and $\bar{x}$ is the center(origin) coined as *stationary point*. More details on the weighting functions are provided in Appendix B.

According to the above, applying Eq. (2.9), which is the generalized WLS, we get the following expression of the minimization problem for the best approximation

$$\begin{aligned} E(f; a_i)|_{\bar{x}} &= \min_{a_i} \int_{B(\bar{x},r)} w(x - \bar{x}) R^2(x) dx \\ &= \min_{a_i} \int_{B(\bar{x},r)} w(x - \bar{x}) \left| f(x) - \sum_{i=0}^{n} a_i \phi_i(x) \right|^2 dx \end{aligned} \tag{2.15}$$

which is a minimization problem of a convolved function $(w * R^2)(\xi)$, $\xi \in B(\bar{x}, r)$. The convolution consists of the kernel-mask $w(x - \bar{x})$ and the squared inner product measured error $R^2(x)$.

The above convolution is considered to preserve the convexity properties of the finite dimensional solution space $\mathcal{S} \subset \mathcal{C} = L^2_w(\bar{\Omega})$, when the weighting function is semi-positive in a compact support, integrable, and rapidly decaying. Such functions are presented in Appendix B, illustrating their properties and results.

As in section 2.3, we assume the function space $\mathcal{C}$, the solution space $\mathcal{S} = \mathbb{P}_n$ with a non-orthogonal basis $\{\phi_i\}_i^n$, the best approximation $P_S(f)|_{\bar{x}} = \hat{f}_b|_{\bar{x}} = \sum_{i=0}^n a_i|_{\bar{x}}\phi_i$, and additionally a weighting function $w_{B(\bar{x},r)}$ with the above stated properties. Then the local approximation can take place from the normal equations of (2.10)

$$\frac{\partial E(f;a_i)|_{\bar{x}}}{\partial a_m} = 0, \quad m = 0,1,2,...,n \tag{2.16}$$

$$\Leftrightarrow \int_{B(\bar{x},r)} w(x-\bar{x}) \sum_{i=0}^n \phi_m(x)\phi_i(x)a_i|_{\bar{x}}dx = \int_{B(\bar{x},r)} w(x-\bar{x})\phi_m(x)f(x)dx$$

Note that to avoid numerical instabilities when computating each local approximation, it is convenient to consider the origin of the the polynomials $\phi_i(x)$ shifted to the stationary point position, i.e. $\phi_i(x-\bar{x})$. In this way, the resulted normal equations are of better condition to process and solve, with smaller effect of the machine limitations on precision and numerical accuracy. The eq. (2.16) rewritten

$$\begin{aligned}\int_{B(\bar{x},r)} w(x-\bar{x}) \sum_{i=0}^n \phi_m(x-\bar{x})\phi_i(x-\bar{x})a_i|_{\bar{x}}dx \\ = \int_{B(\bar{x},r)} w(x-\bar{x})\phi_m(x-\bar{x})f(x)dx\end{aligned} \tag{2.17}$$

The values $\hat{f}(x)|_{\bar{x}}$ of a local approximation in the vicinity of the stationary point $\bar{x}$, considering the shifted basis functions $\phi_{i|\bar{x}_j}$, are estimated from the relation

$$\hat{f}(x)|_{\bar{x}} = \sum_{i=0}^n \phi_i(x-\bar{x})a_i|_{\bar{x}} \tag{2.18}$$

where $a_i|_{\bar{x}}$ are the local coefficients estimated from the minimization problem (2.17).

For the discrete case of the local least squares approximation, as in section 2.2, we consider the data of the reconstructed function, $\mathcal{F} = \{f(\hat{x}_i)\}_{i=1}^M$, $\mathcal{P} = \{\hat{x}_i\}_{i=1}^M \subset \bar{\Omega}$, and the subsets of them

$$\mathcal{P}|_{\bar{x}} = \{\hat{x}_i \mid \hat{x}_i \in B(\bar{x},r),\ 1 \le i \le s,\ n+1 \le s \le M\} \subset \mathcal{P}$$
$$\mathcal{F}|_{\bar{x}} = \{f(\hat{x}_i)\}_{i=1}^s \subset \mathcal{F}$$

The latter subsets are containing only the data within the support of the weighting function, i.e. in the influence ball $B(\bar{x},r)$. Moreover, the weighting function as considered to be radial in d-dimensions (i.e. same value at a certain distance from the center $\bar{x}$), we equip it with the distance metric $\|.\|_2$. Then we have the following local discrete normal equations

$$\begin{aligned}\sum_{i=0}^M \sum_{j=0}^n w(\|\hat{x}_i-\bar{x}\|_2)\phi_m(\hat{x}_i-\bar{x})\phi_j(\hat{x}_i-\bar{x})a_j|_{\bar{x}} \\ = \sum_{i=0}^M w(\|\hat{x}_i-\bar{x}\|_2)\phi_m(\hat{x}_i-\bar{x})f(\hat{x}_i), \quad m = 0,...n\end{aligned} \tag{2.19}$$

The local approximations as defined above, can be expressed in a matrix form as following

$$\mathbf{A}_{\bar{x}}\mathbf{a}_{\bar{x}} = \mathbf{f}_{\bar{x}} \tag{2.20}$$

where

$$\mathbf{A}_{\bar{x}} = \sum_{i=0}^{M} w(\|\hat{x}_i - \bar{x}\|_2) \begin{bmatrix} \phi_0(\hat{x}_i - \bar{x})\phi_0(\hat{x}_i - \bar{x}) & \phi_0(\hat{x}_i - \bar{x})\phi_1(\hat{x}_i - \bar{x}) & ... & \phi_0(\hat{x}_i - \bar{x})\phi_n(\hat{x}_i - \bar{x}) \\ \phi_1(\hat{x}_i - \bar{x})\phi_0(\hat{x}_i - \bar{x}) & \phi_1(\hat{x}_i - \bar{x})\phi_1(\hat{x}_i - \bar{x}) & ... & \phi_1(\hat{x}_i - \bar{x})\phi_n(\hat{x}_i - \bar{x}) \\ ... & ... & ... & ... \\ \phi_n(\hat{x}_i - \bar{x})\phi_0(\hat{x}_i - \bar{x}) & \phi_n(\hat{x}_i - \bar{x})\phi_1(\hat{x}_i - \bar{x}) & ... & \phi_n(\hat{x}_i - \bar{x})\phi_n(\hat{x}_i - \bar{x}) \end{bmatrix} \tag{2.21}$$

$$\mathbf{f}_{\bar{x}} = \sum_{i=0}^{M} w(\|\hat{x}_i - \bar{x}\|_2) \begin{bmatrix} f(\hat{x}_i)\phi_0(\hat{x}_i - \bar{x}) \\ f(\hat{x}_i)\phi_1(\hat{x}_i - \bar{x}) \\ ... \\ f(\hat{x}_i)\phi_n(\hat{x}_i - \bar{x}) \end{bmatrix}, \quad \mathbf{a}_{\bar{x}} = \begin{bmatrix} a_0 \\ a_1 \\ ... \\ a_n \end{bmatrix} \tag{2.22}$$

Alternatively, $\mathbf{A}_{\bar{x}}$ and $\mathbf{f}_{\bar{x}}$ can be expressed with the Vandermonde matrix $\mathbf{V}_{\bar{x}}$

$$\mathbf{A}_{\bar{x}} = \mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{V}_{\bar{x}} \tag{2.23}$$

$$\mathbf{f}_{\bar{x}} = \mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{F} \tag{2.24}$$

$$\mathbf{V}_{\bar{x}} = \begin{bmatrix} \phi_0(\hat{x}_1 - \bar{x}) & \phi_1(\hat{x}_1 - \bar{x}) & ... & \phi_n(\hat{x}_1 - \bar{x}) \\ \phi_0(\hat{x}_2 - \bar{x}) & \phi_1(\hat{x}_2 - \bar{x}) & ... & \phi_n(\hat{x}_2 - \bar{x}) \\ ... & ... & ... & ... \\ \phi_0(\hat{x}_M - \bar{x}) & \phi_1(\hat{x}_M - \bar{x}) & ... & \phi_n(\hat{x}_M - \bar{x}) \end{bmatrix} \tag{2.25}$$

and the diagonal matrix of the weighting function evaluations on the sample points

$$\mathbf{W}_{\bar{x}} = \begin{bmatrix} w_0(\hat{x}_1 - \bar{x}) & 0 & ... & 0 \\ 0 & w_1(\hat{x}_2 - \bar{x}) & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & w_M(\hat{x}_M - \bar{x}) \end{bmatrix} \tag{2.26}$$

$\mathbf{F}$ is the vector with the sample values

$$\mathbf{F} = \begin{bmatrix} f(\hat{x}_1) \\ f(\hat{x}_2) \\ ... \\ f(\hat{x}_M) \end{bmatrix} \tag{2.27}$$

The Vandermonde matrix is constructed as

$$\mathbf{V}_{\bar{x}} = \begin{bmatrix} \mathbf{b}(\hat{x}_1)_{\bar{x}}^T \\ \mathbf{b}(\hat{x}_2)_{\bar{x}}^T \\ ... \\ \mathbf{b}(\hat{x}_M)_{\bar{x}}^T \end{bmatrix}, \quad \mathbf{b}(x)_{\bar{x}} = \begin{bmatrix} \phi_0(x - \bar{x}) \\ \phi_1(x - \bar{x}) \\ ... \\ \phi_n(x - \bar{x}) \end{bmatrix} \tag{2.28}$$

The form with the Vandermonde matrices facilitates the expression of the coefficients in matrix form

$$\mathbf{a}_{\bar{x}} = \left[\mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{V}_{\bar{x}}\right]^{-1} \mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{F} \tag{2.29}$$

Thus, (2.18) can be expressed as

$$\hat{f}(x)_{\bar{x}_j} = \mathbf{b}(x)_{\bar{x}}^T \left[\mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{V}_{\bar{x}}\right]^{-1} \mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{F} = \mathbf{q}(x)\ \mathbf{F} \tag{2.30}$$

$$\mathbf{q}(x) = \mathbf{b}(x)_{\bar{x}}^T \left[\mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \mathbf{V}_{\bar{x}}\right]^{-1} \mathbf{V}_{\bar{x}}^T \mathbf{W}_{\bar{x}} \tag{2.31}$$

$\mathbf{q}(x)$ can be seen as the local shape functions of the sample values $\mathbf{F}$.

An interesting and insightful version of the above local least squares, is given from the 0-degree polynomials, i.e. $\{\phi\} = \{1\}$. In such a case, solving for the only coefficient a_0 in (2.19), we get

$$a_0|_{\bar{x}} = \frac{\sum_{i=0}^{s} w_i f_i}{\sum_{i=0}^{s} w_i} \tag{2.32}$$

where $w_i = w(\|\hat{x}_i - \bar{x}\|_2)$ and $f_i = f(\hat{x}_i)$. Furthermore, the coefficient a_0 is the value of the approximation $\hat{f}(\bar{x}) = a_0|_{\bar{x}}$, as defined in eq. (2.18).

This is the simplest notion of the local least squares, providing the simple form of a weighted average. Additionally, the single local shape function $q(x)$ is constant for a given stationary point $\bar{x}$, but variable with the stationary points,

$$q(\bar{x}) = \frac{w_i(\bar{x})}{\sum_{i=0}^{s} w_i(\bar{x})} \tag{2.33}$$

This form of the weighted average, can make clear the fundamental role of the weighting function and the impact on the resulted approximation. For instance, for the sake of better local approximation, the eq. (2.32) makes apparent the need to assign larger weights to sample points closer to the stationary point, and after a certain distance assigning negligible or zero weights. That explains the need of a rapid decaying function with compact support, both related to the order of approximation (i.e. the quality of the local approximation). As the coefficient a_0 defines the weighted average of the value $f(\bar{x})$, for higher degree local approximations, the coefficient a_1 will define the local slope on the stationary point (as in the case of the taylor series), the coefficient a_2 the local curvature, etc.

Another interesting observation, is that the interpolation with appropriate weighting functions becomes straightforward in eq. (2.32), i.e. we need to provide a weighting function that will assign very large weights to closely distant sample points, and very small values to far sample points, so that the close ones could dominate and strike out the effect of the distant. Hence, when the stationary point is on the sample point position, we need a weighting function that will assign a very large weight on that overlapped sample point, and very rapidly decayed to assign very small weights (by some orders) to other close distant sample points. In cases were the stationary point is overlapping a sample point, and the close distant sample points experience small variations in value, the interpolation can be an easy task for most of the weighting functions. Though, when close sample points experience large variations in value, the weighting function should be powerful enough, in the sense of the dirac function. By analogy, the same technique is applied in the penalty methods, to introduce constraints on a described system.

Finally, all the above remarks, apply smoothly in the global approximation with the MLS method (introduced in section 2.6), fact that can be seen by observing eq. (2.33) which variates smoothly with the stationary points, and considering that the weighting function is rapidly decayed, which permits a smooth fade-in and fade-out of sample points between various adjacent stationary points in their influence balls $B(\bar{x}, r)$.

2.5 Weighted Local Least Squares method (WLS)

One can exclusively use the local approximations, as analysed in section 2.4, around the stationary points to evaluate the function $\hat{f}(x)$ on other domain points $x \in \bar{\Omega}$. Though, especially for domain points far from the close vicinity of the stationary points, one can get improved results and a global smoothness if the effect of multiple stationary points is considered. In this case we employ a global approximation, based on the local approximations that take place on the $\bar{x}$ stationary points.

Assume that we need to approximate globally a function $f(x)$, which is defined on a real d-dimensional domain $\bar{\Omega} \subset \mathbb{R}^d$. Moreover, we have a set of N stationary points $\bar{x} = \{\bar{x}_i\}_{i=1}^N$, such that $\forall \bar{x}_i \in \bar{\Omega}$ an appropriate weighting function $w(B(\bar{x}_i, r_i))$ is defined. Then, the local least squares approximation exactly on each stationary point $\bar{x}_i$, is the best approximation $P_S(f)$ under the weighted inner product $(.,.)_w$, as defined in section 2.4. For the rest of the domain and any specific $x \in \bar{\Omega}$, the evaluation of the function $\hat{f}(x)$ will be based on a subset of best local approximations. That subset corresponds to effective stationary points $\bar{x}|_x = \{\bar{x}_i\}_{i=1}^s$, $1 \leq s \leq N$, within a positive range $r_e > 0$.

For any evaluation point $x \in \bar{\Omega}$, the subset of effective stationary points has members $\bar{x}_i$ with the following property:

$$\|x - \bar{x}_i\|_2 \leq r_e, \quad 1 \leq i \leq s$$

Thus, the order of the global approximation $\hat{f}(x)$ depends on the density of the stationary points, and the global smoothness stems from the smoothness of the weighting function w, which is ruling smoothly the way that the local approximations from one stationary point to another is estimated.

To preserve the definition of the global approximation $\hat{f}(x)$, $\forall x \in \bar{\Omega}$, every point in the domain should have in range r_e at least one stationary point. A convenient way to confirm that, is using the range r_e over each stationary point $\bar{x}_i$. Then N subdomains $\omega_i \subset B(\bar{x}_i, r_i)$ can be defined, such that

$$\omega_i = \{x \mid x \in B(\bar{x}_i, r_i), \|x - \bar{x}_i\|_2 \leq r_e \leq r_i\}$$

Consequently, the problem deteriorates in populating the total set of stationary points $\bar{x} = \{\bar{x}_i\}_{i=1}^N$ with elements such that

$$\bigcup_{i=1}^{N} \omega_i = \bar{\Omega}$$

Interpreted as, there should be no holes in the union of the subsets, i.e. every point of the domain $\bar{\Omega}$ should be effected by at least one local approximation.

Assuming all the above, the evaluation $\forall x \in \bar{\Omega}$ in a global smooth sense, is defined as following

$$\hat{f}(x) = \sum_{j=1}^{s} \chi_j(x) \sum_{i=0}^{n} \phi_i(x - \bar{x}_j) a_{i|\bar{x}_j}, \quad s \leq N \tag{2.34}$$

where s is the number of the effective stationary points, and

$$\chi_j(x) = \frac{w(\|x - \bar{x}_j\|_2)}{\sum_{k=1}^{s} w(\|x - \bar{x}_k\|_2)}$$

The quantity $\chi_j(x)$ represents the coefficients of the convex linear combination, of the estimations given from s in number effective stationary points $\bar{x}_j$, in a simple first moments method.

2.6 Moving Least Squares method (MLS)

The WLS method of the section 2.5, depends on the setup of the stationary points and their influence ball. Such a method can be efficient and effective, when limited precision in the global approximation sense is sufficient. Recall that the best approximation holds only for the stationary points, the rest of the field is approximated by the approximations on the stationary points. The idea in the Moving Least Squares (MLS) is to define as many stationary points as the evaluation points, i.e.

$$\bar{x} \equiv x$$

Consequently, one gets the best approximation $P_S(f)$ under the weighted inner product $(\cdot,\cdot)_w$, continuously over entire the domain. The smoothness of the global approximation, after employing the moving local least squares, is inherrited from the smoothness of the weighting function w, fact which is commented in Appendix B.

Besides the improved approximation results in MLS, which comes with a computational cost for a large number of stationary points, the computations per stationary point can be managed slightly diffrently, compared to the WLS. In WLS we had to solve for all the coefficients a_i of the normal equations (2.20) and store them, in order to have the most of the available information (i.e. the first derivatives around the stationary point, when a polynomial basis of degree $n \geq 0$ is used). That available information was very valuable to have a good approximation in the evaluation of the rest of the points $x \in \bar{\Omega}$. In the case of the MLS, there is no need to keep that additional information, if one is interested to define only the value $f(x)$ exactly on the stationary point $f(\bar{x})$. Therefore, one can solve and store, only the coefficient a_0 that evaluates explicitly the value $f(\bar{x})$, i.e. employing the (2.18) we have

$$f(\bar{x})|_{\bar{x}} = \sum_{i=0}^{n} \phi_i(\bar{x} - \bar{x}) a_{i\,|\bar{x}} = a_{0\,|\bar{x}} \tag{2.35}$$

In case of higher degree approximations, a good approximation of the gradients, e.g. $\frac{\partial f(x)}{\partial x}|_{\bar{x}}$, $\frac{\partial f(x)}{\partial y}|_{\bar{x}}$, can be obtained from the coefficients corresponding to the linear monomials, but only on the stationary point position $\bar{x}$. More precise evaluations in the vicinity of the stationary point, should be evaluated by the standard derivative of $f(x)$

$$\frac{\partial f(x)}{\partial x} = \sum_{i=0}^{n} \frac{\partial \phi_i(x - \bar{x})}{\partial x} a_i(x) + \phi_i(x - \bar{x}) \frac{\partial a_i(x)}{\partial x} \tag{2.36}$$

In the case of zero degree polynomials, we can receive an order of smoothness from the weighting function.

$$\frac{\partial f}{\partial x} = \frac{\partial a_0(x)}{\partial x} = \sum_{i=0}^{s} \left[\left(\frac{\partial}{\partial x} \frac{w_i}{\sum_{j=0}^{s} w_j} \right) \hat{f}_i \right] \tag{2.37}$$

where

$$\left(\frac{\partial}{\partial x} \frac{w_i}{\sum_{j=0}^{s} w_j} \right) = \frac{\frac{\partial w_i}{\partial x}}{\sum_{j=0}^{s} w_j} - \frac{w_i}{\left(\sum_{j=0}^{s} w_j\right)^2} \sum_{k=0}^{s} \frac{\partial w_k}{\partial x}$$

and s is the number of effective sample points.

The variation order of the shape functions (2.31), with x, i.e. $\frac{d^\alpha q(x)}{dx^\alpha}$, depends on the variation order of the weighting function w. Mirzaei [2015] provides the proves and the error estimates of the derivatives in Sobolev spaces, with the corresponding error bounds under conditions on the involved parameters. The involved parameters, are mainly the scaled support of the weighting function and the degree of tha polynomial basis.

2.7 Analysis of the MLS computations and data structures

In this section, a thorough description of the MLS computations is presented, as a guideline for all the involved data structures, sizes, compute complexities and patterns. In the next subsections, two simple versions of the local least squares can be found, i.e. the zero and 1^{st} degree in an explicit formula form. These can prove useful for the developement of more efficient GPGPU algorithm designs.

In order to describe all the involved data structures, their size and the operations, the index notation will be employed, and the corresponding index sets are shown in Table 2.1.

Index sets	
$\mathcal{N} = \{i \in \mathbb{N} \mid 1 \leq i \leq d\}$	spatial dimensions index set, where d is the number of the spatial (domain) dimensions
$\mathcal{M} = \{i \in \mathbb{N} \mid 1 \leq i \leq M\}$	the sample points index set, where M is the number of the sample points
$\mathcal{E} = \{i \in \mathbb{N} \mid 1 \leq i \leq N\}$	the evaluation points index set, where N is the number of the sample points
$\mathcal{B} = \{i \in \mathbb{N} \mid 1 \leq i \leq n\}$	the monomials index set, where n is the number of monomials for polynomials of degree $n-1$
$\mathcal{F} = \{i \in \mathbb{N} \mid 1 \leq i \leq Q\}$	the approximated fields index set, where Q is the number of the fields

Table 2.1: Table of index sets used in computations' notation.

In a standard way of using the index notation, there are two types of subscripts:

1. The *free indices* that occur once in a single term, and
2. the *summed indices*, which appear twice in a single term.

The summed indices are being used for contractions and we will use it here only for inner product. As a deviation to the standard notation, and to represent the weighted inner product as defined in section 2.3, the following will note a contraction as well

$$(u, v)_w = u_i v_i w_i$$

Our very first need in data structures, is to store the given problem's data and the output data, see Table 2.2.

I/O data structures		
Notation	Indices	Description
$\hat{x}_{ij}$ or $\hat{x}_j$	$i \in \mathcal{N}$, $j \in \mathcal{M}$	a matrix of the sample points spatial components
$\hat{f}_{ij}$	$i \in \mathcal{F}$, $j \in \mathcal{M}$	a matrix of the approximated field values per sample point (the field is assumed to be multicomponent, or conversely multiple fields)
$\bar{x}_{ij}$ or $\bar{x}_j$	$i \in \mathcal{N}$, $j \in \mathcal{E}$	a matrix with the evaluation points location
$\bar{f}_{ij}$	$i \in \mathcal{F}$, $j \in \mathcal{E}$	a matrix with the evaluations per point and per approximated field.

Table 2.2: I/O data structures

As a convention, we will adapt the short notation $\hat{x}_i$, and $\bar{x}_i$ from the Table 2.2, implying that the spatial components are incorporated.

Hereafter, the structure of the MLS computations will refer to a single evaluation point, i.e. one single local approximation as defined in section 2.4.

The vector with the distances of the sample points from the evaluation point, expressed by the $\|.\|_2$

$$d_i(\bar{x}) = \begin{bmatrix} d(\hat{x}_1 - \bar{x}) \\ d(\hat{x}_2 - \bar{x}) \\ d(\hat{x}_3 - \bar{x}) \\ \dots \\ d(\hat{x}_M - \bar{x}) \end{bmatrix}, i \in \mathcal{M} \tag{2.38}$$

The vector with the weight evaluations for all the sample point

$$w_i(\bar{x}) = w(d_i(\bar{x})), \quad i \in \mathcal{M} \tag{2.39}$$

The vector with the polynomial basis functions, i.e. $\{\phi_i\}_i = 1^{n-1}$ in section 2.3, for a single sample point

$$b_i(\hat{x}) = \begin{bmatrix} b_1(\hat{x} - \bar{x}) \\ b_2(\hat{x} - \bar{x}) \\ b_3(\hat{x} - \bar{x}) \\ \dots \\ b_n(\hat{x} - \bar{x}) \end{bmatrix}, i \in \mathcal{B} \tag{2.40}$$

The monomials b_i, are evaluated at the sample points from a shifted coordinate system placed on the current evaluation point, i.e. $\hat{x}_i - \bar{x}$. For shorter notation, it will be written as $b_i(\hat{x}_j)$, implying that we refer to a single current evaluation point which can be omitted in this text. Such monomials should form a complete space of polynomials up to the desired degree. For example in the 1D case, a polynomial of 4^{th} degree, can use the following basis

$$b = [1 \quad x \quad x^2 \quad x^3 \quad x^4]$$

The 2D case for a polynomial of 3^{rd} degree, can use the basis

$$b = \begin{bmatrix} 1 & x & x^2 & x^3 & y & xy & x^2y & y^2 & xy^2 & y^3 \end{bmatrix}$$

The 3D case for a polynomial of 2^{nd} degree, can have the basis

$$b = \begin{bmatrix} 1 & x & x^2 & y & xy & y^2 & z & zx & zy & z^2 \end{bmatrix}$$

A Vandermonde matrix with the monomials for all the sample points of a single evaluation point

$$V_{ij} = b_j(\hat{x}_i) = \begin{bmatrix} 1 & b_2(\hat{x}_1) & b_3(\hat{x}_1) & ... & b_n(\hat{x}_1) \\ 1 & b_2(\hat{x}_2) & b_3(\hat{x}_2) & ... & b_n(\hat{x}_2) \\ 1 & b_2(\hat{x}_3) & b_3(\hat{x}_3) & ... & b_n(\hat{x}_3) \\ 1 & ... & ... & ... & ... \\ 1 & b_2(\hat{x}_M) & b_3(\hat{x}_M) & ... & b_n(\hat{x}_M) \end{bmatrix} \quad (2.41)$$
$$i \in \mathcal{M}, j \in \mathcal{B}$$

The weighted Vandermonde matrix

$$V^w_{ij} = V_{\underline{i}j} w_{\underline{i}} = \begin{bmatrix} w_1 & w_1 * b_2(\hat{x}_1) & w_1 * b_3(\hat{x}_1) & ... & w_1 * b_n(\hat{x}_1) \\ w_2 & w_2 * b_2(\hat{x}_2) & w_2 * b_3(\hat{x}_2) & ... & w_2 * b_n(\hat{x}_2) \\ w_3 & w_3 * b_2(\hat{x}_3) & w_3 * b_3(\hat{x}_3) & ... & w_3 * b_n(\hat{x}_3) \\ ... & ... & ... & ... & ... \\ w_M & w_M * b_2(\hat{x}_M) & w_M * b_3(\hat{x}_M) & ... & w_M * b_n(\hat{x}_M) \end{bmatrix} \quad (2.42)$$
$$i \in \mathcal{M}, j \in \mathcal{B}$$

Then the system matrix for a single evaluation point, is given as the following Gramian

$$A_{ij} = V_{ki} V^w_{kj} = \begin{bmatrix} w_{kk} & w_k b_2(\hat{x}_k) & w_k b_3(\hat{x}_k) & ... & w_k b_n(\hat{x}_k) \\ w_k b_2(\hat{x}_k) & w_k b_2(\hat{x}_k) b_2(\hat{x}_k) & w_k b_3(\hat{x}_k) b_2(\hat{x}_k) & ... & w_k b_n(\hat{x}_k) b_2(\hat{x}_k) \\ w_k b_3(\hat{x}_k) & w_k b_2(\hat{x}_k) b_3(\hat{x}_k) & w_k b_3(\hat{x}_k) b_3(\hat{x}_k) & ... & w_k b_n(\hat{x}_k) b_3(\hat{x}_k) \\ ... & ... & ... & ... & ... \\ w_k b_n(\hat{x}_k) & w_k b_2(\hat{x}_k) b_n(\hat{x}_k) & w_k b_3(\hat{x}_k) b_n(\hat{x}_k) & ... & w_k b_n(\hat{x}_k) b_n(\hat{x}_k) \end{bmatrix} \quad (2.43)$$
$$i \in \mathcal{B}, j \in \mathcal{B}, k \in \mathcal{M}$$

The system RHS vectors for a single evaluation point and for all the approximated fields

$$F_{ij} = V^w_{ki} \hat{f}_{jk} = \begin{bmatrix} w_k \hat{f}_{1k} & w_k \hat{f}_{2k} & w_k \hat{f}_{3k} & ... & w_k \hat{f}_{Qk} \\ w_k \hat{f}_{1k} b_2(\hat{x}_k) & w_k \hat{f}_{2k} b_2(\hat{x}_k) & w_k \hat{f}_{3k} b_2(\hat{x}_k) & ... & w_k \hat{f}_{Qk} b_2(\hat{x}_k) \\ w_k \hat{f}_{1k} b_3(\hat{x}_k) & w_k \hat{f}_{2k} b_3(\hat{x}_k) & w_k \hat{f}_{3k} b_3(\hat{x}_k) & ... & w_k \hat{f}_{Qk} b_3(\hat{x}_k) \\ ... & ... & ... & ... & ... \\ w_k \hat{f}_{1k} b_n(\hat{x}_k) & w_k \hat{f}_{2k} b_n(\hat{x}_k) & w_k \hat{f}_{3k} b_n(\hat{x}_k) & ... & w_k \hat{f}_{Qk} b_n(\hat{x}_k) \end{bmatrix} \quad (2.44)$$
$$i \in \mathcal{B}, j \in \mathcal{F}, k \in \mathcal{M}$$

We obtain the solution of the Q (the number of approximated fields) systems in number, as following

$$c_{ij} = A^{-1}_{ik} F_{kj} \quad (2.45)$$
$$i \in \mathcal{B}, j \in \mathcal{F}, k \in \mathcal{B}$$

2.7.1 Two straightforward special cases on the polynomial degree

The case of 0-degree
For the zero degree polynomial case, i.e. $n = 1$, the solution vector per approximated field, is given straightforward as following

$$c_j = \frac{w_k \hat{f}_{jk}}{w_{kk}} = \begin{bmatrix} \frac{w_k \hat{f}_{1k}}{w_{kk}} & \frac{w_k \hat{f}_{2k}}{w_{kk}} & \frac{w_k \hat{f}_{3k}}{w_{kk}} & \dots & \frac{w_k \hat{f}_{Qk}}{w_{kk}} \end{bmatrix} \tag{2.46}$$

The case of 1D and 1^{st}-degree
For the first degree polynomial case, i.e. $n = 2$, the solution vector per approximated field, is evaluated as following

$$\begin{aligned} c_{1j} &= \frac{F_{1j} - P_1 F_{2j}}{A_{11} - P_1 A_{21}} \\ c_{2j} &= \frac{F_{2j} - A_{21} c_{1j}}{A_{22}} \end{aligned} \tag{2.47}$$

where we take

$$\begin{aligned} A_{11} &= w_{kk} \\ A_{12} &= A_{21} = w_k b_2(\hat{x}_k) \\ A_{22} &= w_k b_2(\hat{x}_k) b_2(\hat{x}_k) \\ F_{1j} &= w_k \hat{f}_{jk} \\ F_{2j} &= w_k \hat{f}_{jk} b_2(\hat{x}_k) \\ P_1 &= \frac{A_{12}}{A_{22}} = \frac{w_k b_2(\hat{x}_k)}{w_k b_2(\hat{x}_k) b_2(\hat{x}_k)} \end{aligned} \tag{2.48}$$

as explained in section 2.6 and showed in relation (2.35), for the value of the field $\bar{f}$ we need only the coefficient c_{1j}. With this front substitution form of Gauss elimination one can calculate directly the first term.

The same coefficients with a back substitution, where one can have directly the derivative coefficient.

$$\begin{aligned} c_{1j} &= \frac{F_{1j} - A_{12} c_{2j}}{A_{11}} \\ c_{2j} &= \frac{F_{2j} - P_1 F_{1j}}{A_{22} - P_1 A_{12}} \end{aligned} \tag{2.49}$$

where we take

$$\begin{aligned} A_{11} &= w_{kk} \\ A_{12} &= A_{21} = w_k b_2(\hat{x}_k) \\ A_{22} &= w_k b_2(\hat{x}_k) b_2(\hat{x}_k) \\ F_{1j} &= w_k \hat{f}_{jk} \\ F_{2j} &= w_k \hat{f}_{jk} b_2(\hat{x}_k) \\ P_1 &= \frac{A_{21}}{A_{11}} = \frac{w_k b_2(\hat{x}_k)}{w_{kk}} \end{aligned} \tag{2.50}$$

The case of 2D and 1^{st}-degree
For the first degree polynomial case, i.e. $n = 3$, the solution vector per approximated field, is evaluated as following

$$\begin{aligned} c_{1j} &= \frac{F_{1j} - c_{2j}A_{12} - c_{3j}A_{13}}{A_{11}} \\ c_{2j} &= \frac{F_{2j} - P_{11}F_{1j} - c_{3j}(A_{23} - P_{11}A_{13})}{A_{22} - P_{11}A_{12}} \\ c_{3j} &= \frac{F_{3j} - P_{12}F_{1j} - P_{21}(F_{2j} - P_{11}F_{1j})}{A_{33} - P_{12}A_{13} - P_{21}(A_{23} - P_{11}A_{13})} \end{aligned} \tag{2.51}$$

where

$$\begin{aligned} A_{11} &= w_{kk} \\ A_{12} &= A_{21} = w_k b_2(\hat{x}_k) \\ A_{13} &= A_{31} = w_k b_3(\hat{x}_k) \\ A_{22} &= w_k b_2(\hat{x}_k) b_2(\hat{x}_k) \\ A_{23} &= A_{32} = w_k b_2(\hat{x}_k) b_3(\hat{x}_k) \\ A_{33} &= w_k b_3(\hat{x}_k) b_3(\hat{x}_k) \\ F_{1j} &= w_k \hat{f}_{jk} \\ F_{2j} &= w_k \hat{f}_{jk} b_2(\hat{x}_k) \\ F_{3j} &= w_k \hat{f}_{jk} b_3(\hat{x}_k) \\ P_{11} &= \frac{A_{21}}{A_{11}} \\ P_{12} &= \frac{A_{31}}{A_{11}} \\ P_{21} &= \frac{A_{32} - P_{12}A_{12}}{A_{22} - P_{11}A_{12}} \end{aligned} \tag{2.52}$$

as explained in section 2.6 and showed in relation (2.35), for the value of the field $\bar{f}$ we need only the coefficient c_{1j}. With this front substitution form of Gauss elimination one can calculate directly the first term.

□

Chapter 3

The parallelization of the MLS method and implementation in CUDA

3.1 Introduction

In Chapter 2 the MLS method is analyzed as an abstract mathematical tool to approximate functions. In mesh deformation applications, the MLS method is instantiated to the known displaced boundary nodes (0 displacement data are included), as the given sample points-data, and the unknown displacement of the interior nodes, as the evaluation points-results. The most frequent case is that where all the boundary nodes will be prescribed with some displacements (zero displaced fixed nodes are included), and the new positions of the interior nodes are required.

In this chapter the parallelization of the MLS method is analyzed and CUDA algorithms are proposed for the parallel execution of the computations. As stated in Chapter 2, the MLS is a standalone local approximation method over a single evaluation point, i.e. it evaluates the node displacement with an individual local least squares fit. Hence, a parallel algorithm can be designed where the computations of each evaluation point are done independently and without the need to exchange information between the computing units. This allows a very easy coarse grain parallelism of the MLS computations.

Furthermore, in section 3.4, a fine grain parallelization is analyzed, which concerns the computations of a single evaluation point. In section 3.3 an analysis of the computing time is shown for a fully serialized MLS procedure. The results from that section in combination with the analysis in section 2.7, assist to focus the parallelization efforts on the coefficients evaluation task, exclusively. Thus, in this chapter, the case of the 0-degree local least squares is presented and analyzed with an efficient algorithm design (i.e. the reduction algorithm). Subsequently, the parallelization of the higher degree MLS method is analyzed, utilizing the results from section 2.7. Finally, in section 3.5 the CUDA algorithm designs are presented, along with the necessary parameters of the CUDA execution model.

3.2 Coarse grain parallelization of the MLS method

In MLS method, the perfect data independency among the evaluation nodes, permits a theoretically efficient parallel algorithm for which the total execution time converges to the spent time for a single node evaluation, as the computing resources grow to infinity. Moreover, there is no need for infinite resources, when a number of computing processors p greater or equal than the number of the evaluation mesh nodes N, would be sufficient. The theoretical minimum of the execution time (i.e. of one single node evaluation) is known as the span time or critical length $T_\infty = T_N$. At this point we implicitly assumed that the MLS method launched for a single node consists of fully dependent processes which are serialized (i.e. a critical length). Subsequently, the span time T_N will be investigated to parallelize as well.

In most practical cases, the work will be done from a number of processors $p \leq N$ with iterations. Then the total execution time will be greater than the span time, i.e.

$$T_p \geq T_N \tag{3.1}$$

which is coined as the span law.
Assuming that the total execution time with no parallelization on a single processor is T_1, then the total execution time with p processors is constrained by

$$T_p \geq \frac{T_1}{p} \tag{3.2}$$

which is known as the work law.

Furthermore, Brent's theorem guarantees the following bounds

$$T_p \leq T_N + \frac{T_1 - T_N}{p} \tag{3.3}$$

or combined with the work law

$$\frac{T_1}{p} \leq T_p \leq T_N + \frac{T_1 - T_N}{p} \tag{3.4}$$

Then according to the above, we can define the following useful notions

- The speedup

  $$S_p = \frac{T_1}{T_p} \tag{3.5}$$

  where substituting the upper bound of Brent's theorem we get the Amdahl's law

  $$S_p = \frac{1}{\frac{T_N}{T_1} + \frac{T_1 - T_N}{T_1 p}} \tag{3.6}$$

  In the denominator the first term gives a constant corresponding to the serial part of the process and limiting always the overall speedup (large serial part (T_N) limits drastically the speedup). The second term expresses the gained speedup from the parallelized part of the process (small parallelized part of the process ($T_1 - T_N$) will result small overall speedup even for a large number of processors p).

- The parallelism
$$\frac{T_1}{T_\infty} \tag{3.7}$$
which is the maximum possible speedup as $p \to \infty$.
- The efficiency
$$\frac{S_p}{p} \equiv \frac{T_1}{T_p p} \tag{3.8}$$
which is the ratio of the execution times with and without parallelism. For $\frac{S_p}{p} = 1$ the algorithm is perfect linear on the input size (simple model), for $\frac{S_p}{p} > 1$ the speedup is super linear due to memory hierarchy effects. Both cases are considered efficient.

Explaining the MLS parallelization, assume that there are p processors available to evaluate N in total interior mesh nodes and d independent components of the displacement. The designed parallel algorithm should permit cases where $p \leq N$ by letting each processor to iterate on N/p mesh nodes. Then the total execution time T_p will be bounded in the limits given in eq. (3.4), with T_N the evaluation time for one mesh node. This algorithm is very easy to implement on a Multithreaded-Multicore CPU (or any parallel scheme with multi-cpu and cluster platforms) for all the polynomial degrees of the method.

3.3 Fine grain analysis of the MLS execution time

In this section the execution time for a serialized code of the MLS method is presented and analyzed. The scope is to determine the parts of the process that is worthwhile to invest some efforts to parallelize and achieve a close to optimal speedup. These parts are the one that consume most of the execution time and there is sufficient data independency to design a parallel algorithm. It is important to maximize the parallelizable part of the process, as only then the available rich compute resources can give a good speedup. For more see the Amdahl's law in 3.2.

To measure the execution time of the MLS process for various polynomial degrees, a single problem is used with the same parameters. The benchmark problem has a number of evaluation nodes $N = 7110$, and a number of the sample points $M = 437$. The serial code is generally very well optimized and compiled with the optimization flag -O3 and the mavx instruction sets enabled. The CPU that executed the computations, was the quad-core Intel Core i7-3740QM, executing with one thread.

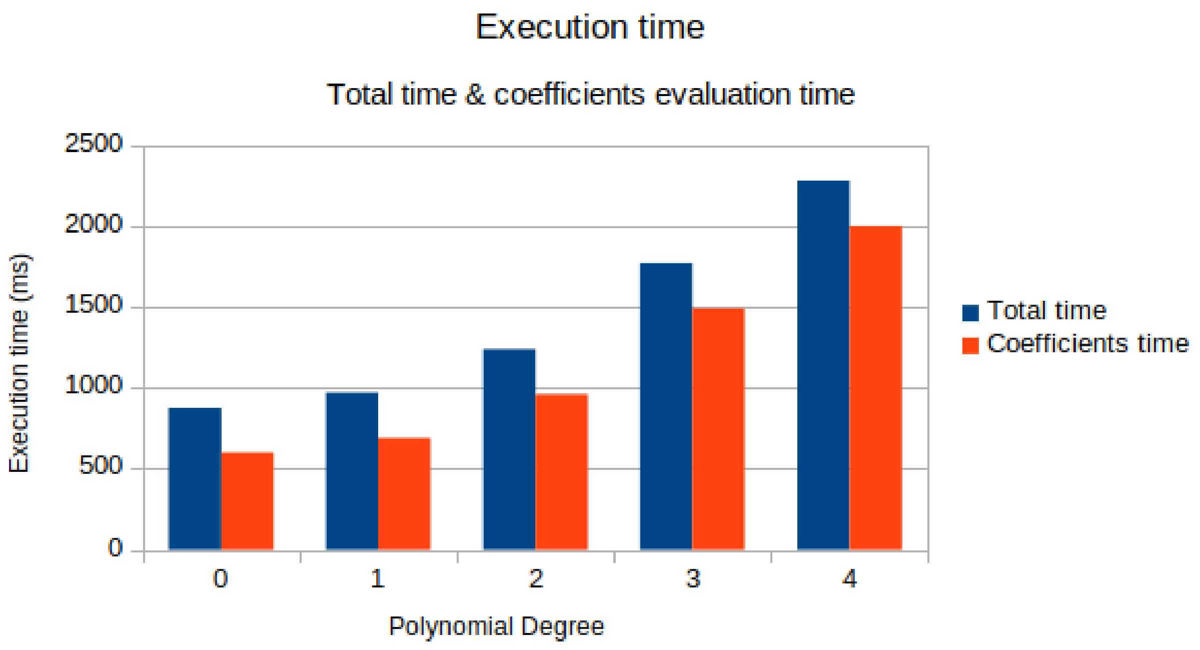
The global execution times for the mentioned problem and the hardware-software setup, for polynomial degrees $0 - 4$ are as following:

$$600ms, 700ms, 900ms, 1360ms, 2000ms$$

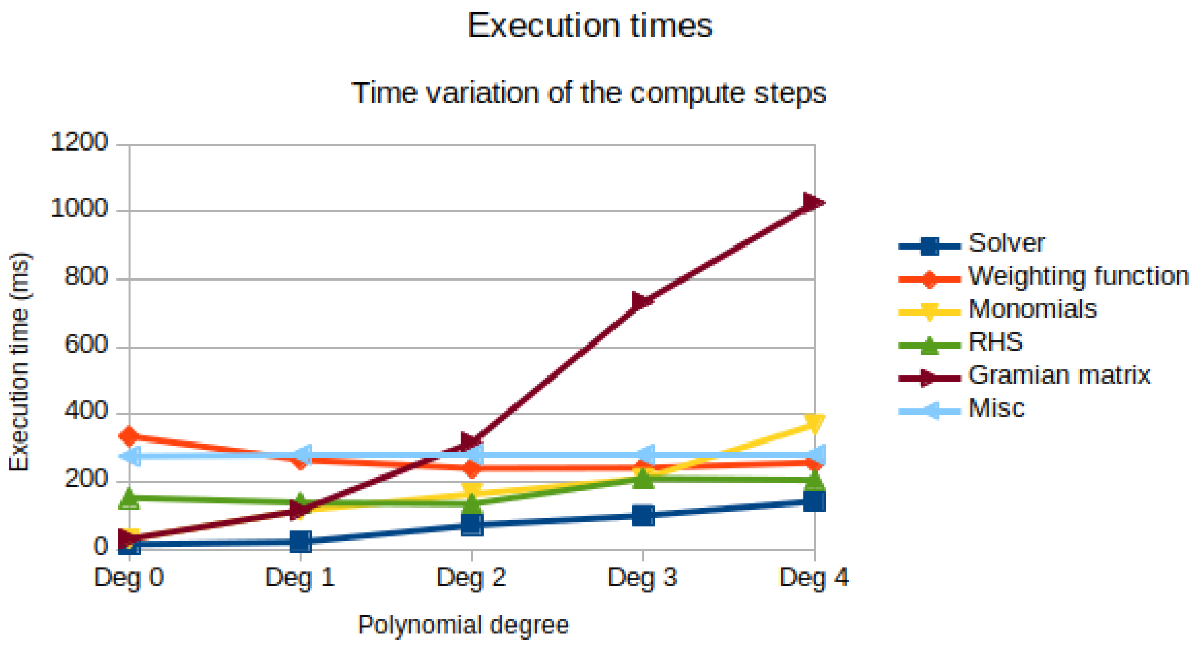
In more detail, in figure 3.1 the execution time for various polynomial degrees is presented. In figure 3.1a the execution time of the coefficients evaluation process is compared with the total execution time of the MLS process. The dominance of the process is aparent for all the polynomial degrees in a slightly increased proportion as the degree elevates. In figure 3.1b and 3.2a, the various parts of the coefficient evaluation process are presented, with their execution time, and figure 3.2b illustrates the time proportion of each part in the coefficients evaluation process time.

Note that the solver of the linear system keeps a very low proportion of the execution time, thus it will be not of any interest to parallelize. Moreover, explicit formulas can be derived for the coefficient a_0, after solving the gauss elimination analytically, which can assist the parallel algorithm design overall, see 2.7.1.

The evaluation cost of the Gramian matrix $\mathbf{A}$ grows to a considerable level starting from the 2^{nd} degree and dominating over the rest parts after the 3^{rd} degree. The evaluation cost of the weighting function, the monomials and the RHS is almost constant and considerable. The latter evaluated quantities and the Gramian matrix are all related and dependent per sample point (see section 2.7), thus all of them should be evaluated with a low communication cost in the developed parallel algorithms.

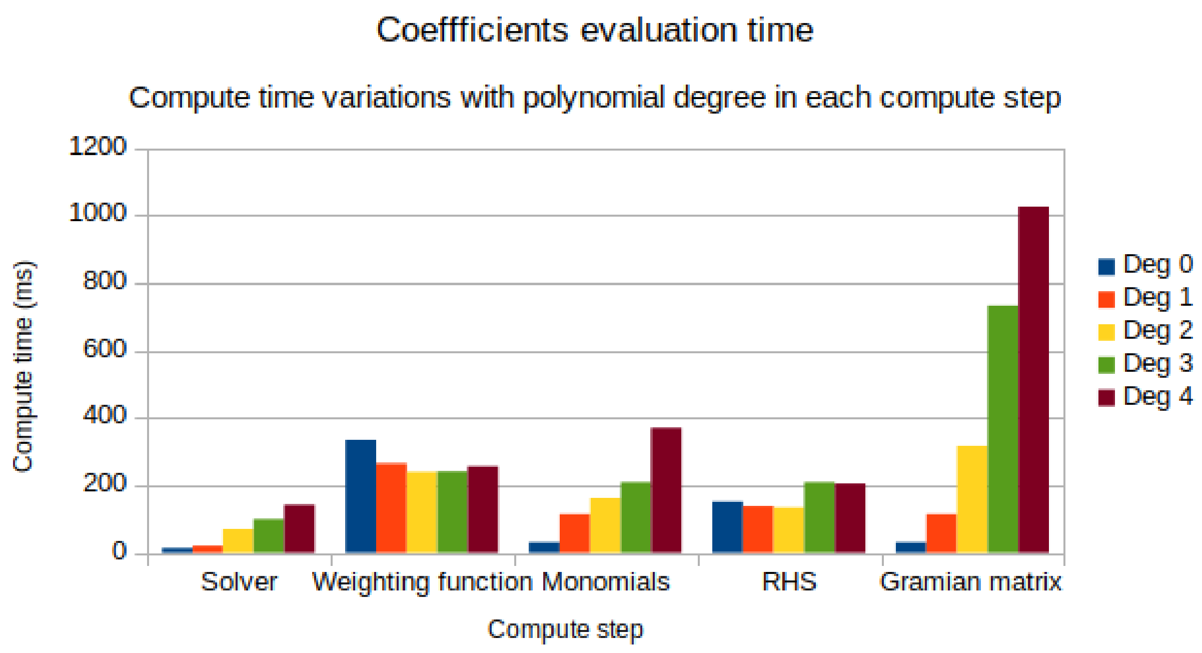


(a) Global execution times.

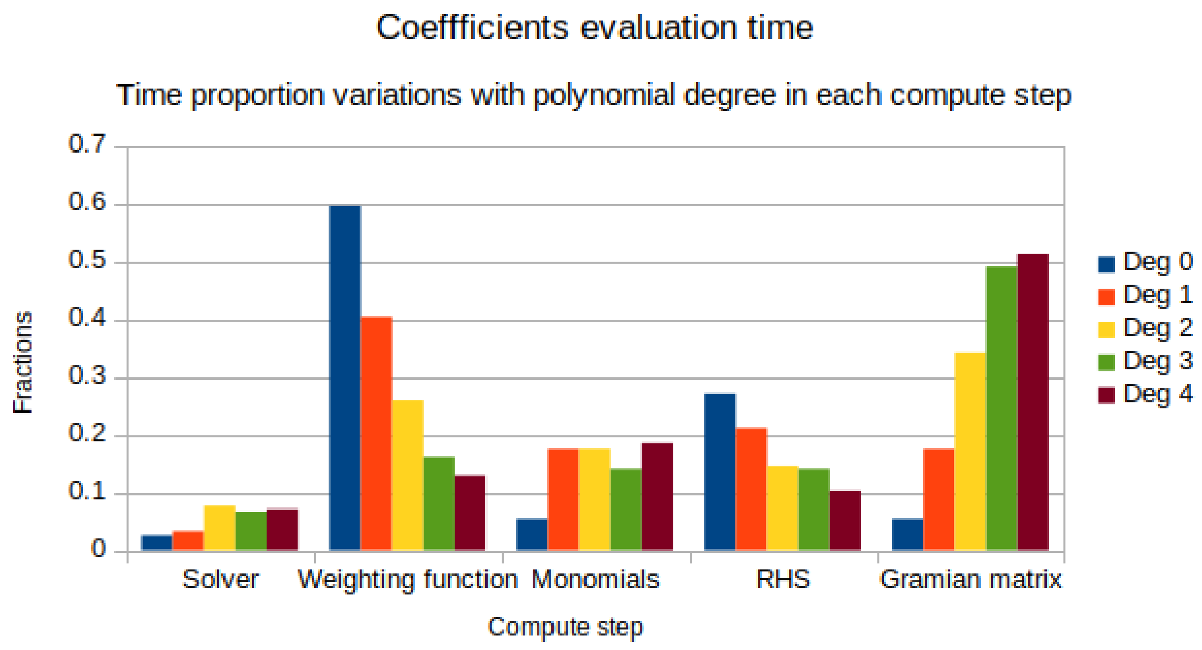


(b) Execution times per compute step.

Figure 3.1: Execution times.



(a) Computing time (ms).



(b) Time fractions.

Figure 3.2: Time variations with polynomial degree, per compute step.

3.4 Fine grain parallelization of the MLS method

In section 3.2, the main parallelization of the method has been shown, i.e. several adapted nodes are evaluated in parallel, with the assumption that all the processes within the evaluation of a single mesh node are dependent and serial. Hence, a single local least squares process time constitutes the so called span time T_∞ of the entire process, which is the convergence limit of the entire process time T_p as the number of the processors p increases

$$\lim_{p\to\infty} T_p = T_\infty = T_N$$

where T_N is the equivalent span time when we are dealing with a finite number of adapted mesh nodes N (i.e. interior mesh nodes typically).

In this section the computations within the local least squares are analyzed and efficient parallelization algorithms are proposed, in order to reduce the span time T_N and consequently the entire process time.

First, the simple 0-degree local least squares is analyzed, where the computations are based on one monomial and the complexity is lower. Following is the complexity analysis for the general case of n in number monomials, where it is recognized that several parts should be considered to parallelized separately, but with very low communication cost, as many quantities can be evaluated once and shared.

3.4.1 The parallelization of the 0-degree local least squares

It is simple to start with the 0-degree local approximation as is described in eq. (2.32) of section 2.4, and is repeated below for a stationary-evaluation point x

$$f(x) = a_0(x) = \frac{\sum_{i=0}^{M} w_i(x) f_i}{\sum_{i=0}^{M} w_i(x)}$$

where M is the number of effective (within the weighting function's span) boundary nodes with prescribed displacements (generally termed as sample points).

Observing the weighted average above, the sums on the numerator and the denominator could reduce the span time, if the size M of the set of sample points per evaluation point is significant, such that the set can be scattered in subsets to a p number of processors and then gather the results back for the final summation. In CUDA, there is an equivalent efficient algorithm design, known as the reduction algorithm, where we take advantage of the memory hierarchy and fast access memory (i.e. the thread block shared memory). A simple example of the reduction algorithm in real life, is the procedure followed in sports competition finals, where in each round it remains half of the participants until the final winner, with the assumption that all the games per round are happening at the same time and the next round starts immediately after the end of all the games in the round, see figure 3.3 for a diagram representation (Kirk and Hwu [2012]).

The reduction algorithm is designed to process the data in tiles if $p < M/2$, where the number of tiles is $M/(2p)$, and each tile is processed in $\log_2(2p)$ iterations. The total number of additions is the arithmetic sequence $\sum_{i=1} \frac{2p}{2^i} = 2p-1$, i.e. $\mathcal{O}(2p)$ and implies the efficiency

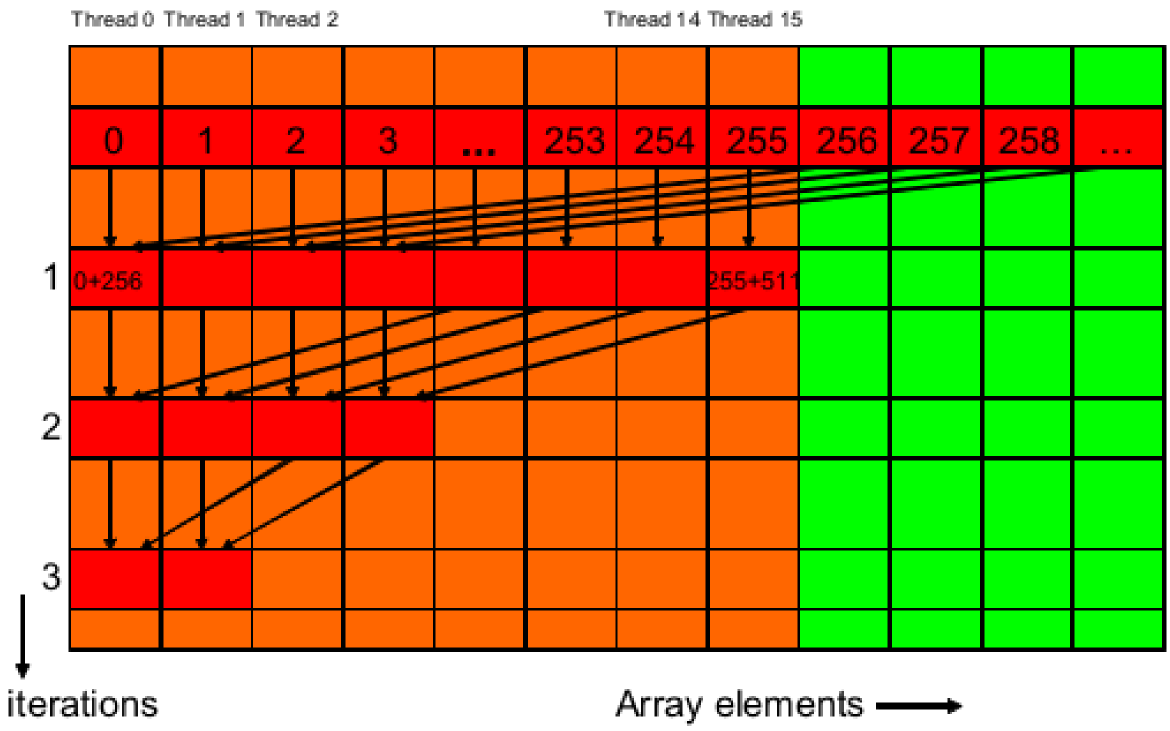


Figure 3.3: The reduction algorithm in diagram representation. (Kirk and Hwu [2012])

of the algorithm $\left(\frac{T_1}{\log_2(2p)2p} \geq 1\right)$. In the first iteration a pair of values is assigned to every processor for the sum opperation, hence p processors can process up to $2p$ elements. The results, p in number, are returned to the shared memory, and the second iteration is ready to start, with $p/2$ processors being assigned a new pair of values from the previously returned results. The procedure iterates with a reduced number of data and active processors by half in each step. The algorithm terminates when at the last step a single processor will sum the last pair of data, which is the result of $2p$ summed elements. The summation of the results returned from the $M/(2p)$ tiles, can be executed on the host processor, or by launching a new reduction algorithm over the $M/(2p)$ number of data.

If one reduction algorithm can be launched for entire the set of effective sample points per evaluation point (a constant number of effective sample points is assumed), and $NM/2$ number of processors is available, then the span time of the entire MLS process can be reduced to a smaller fraction, i.e.

$$T_\infty = T_{NM/2} = \mathcal{O}(\log_2 M)$$

This design is a work efficient design to reduce the span time, but not resources efficient, as after the first round of the reduction algorithm in the computations of each evaluation point, a large amount of the processors are unemployed. Consequently, a very careful design should be implemented to restrict the reservation of unemployed processors through entire the process.

For a smaller number of available processors per evaluation point, $\bar{p} \leq M/2$, the Brent's bounds are applied again for a single point evaluation (a single local least squares fit), where the quantities are written with an overbar to avoid confusion with the entire process quantities

$$\frac{\bar{T}_1}{\bar{p}} \leq \bar{T}_{\bar{p}} \leq \bar{T}_{M/2} + \frac{\bar{T}_1 - \bar{T}_{M/2}}{\bar{p}} \tag{3.9}$$

where $\bar{T}_1$ is the former entire process time span (T_N) with the processes in the local least squares serialized, $\bar{T}_{M/2}$ is the work efficient time span but not resources efficient ($T_{NM/2}$), and $\bar{T}_{\bar{p}}$ is the time span of the entire process.

In the very realistic case that our resources are limited and the total number of available processors is $M/2 \leq p \cdot \bar{p} \leq NM/2$, then an algorithm design where each point is evaluated with $\bar{p} = M/2$ processors, has the following time bounds

$$\begin{aligned} \frac{T_1}{pM/2} \leq T_{pM/2} \leq T_{NM/2} + \frac{T_1 - T_{NM/2}}{pM/2} \\ \Leftrightarrow \frac{T_1}{pM/2} \leq T_{pM/2} \leq \mathcal{O}(\log_2 M) + \frac{T_1 - \mathcal{O}(\log_2 M)}{pM/2} \end{aligned} \tag{3.10}$$

Such that

$$\begin{aligned} \lim_{p\to\infty} \frac{T_1}{pM/2} \leq T_{pM/2} \leq T_{NM/2} + \frac{T_1 - T_{NM/2}}{pM/2} \\ \Leftrightarrow \frac{T_1}{NM/2} \leq T_{NM/2} \leq T_{NM/2} + \frac{T_1 - T_{NM/2}}{NM/2} \end{aligned} \tag{3.11}$$

The right side states that the execution time is limited by the serial part of a single local least squares process and the parallelization of the total evaluations. The left side is stating that the execution time will be larger than the serialized execution time devided by the number of processors parallelizing the evaluation of points. The speedup can be expressed with eq. (3.5).

If our resources are even more limited, i.e. $p \cdot \bar{p} < M/2$, or the design of the algorithm is limiting the resources per local least squares computations $\bar{p} < M/2$ (i.e. the span time is between the work efficient and the fully serialized, given by $\bar{T}_{\bar{p}}$ of (3.9)), Then, using the results from eq. (3.9) we obtained

$$\begin{aligned} \frac{T_1}{p\bar{p}} \leq T_{p\bar{p}} \leq \bar{T}_{\bar{p}} + \frac{T_1 - \bar{T}_{\bar{p}}}{p\bar{p}} \\ \Leftrightarrow \frac{T_1}{p\bar{p}} \leq T_{p\bar{p}} \leq T_{NM/2} + \frac{T_N - T_{NM/2}}{\bar{p}} + \frac{T_1 - T_{NM/2} - \frac{T_N - T_{NM/2}}{\bar{p}}}{p\bar{p}} \\ \Leftrightarrow \frac{T_1}{p\bar{p}} \leq T_{p\bar{p}} \leq T_{NM/2} + \left(\frac{T_1 - T_{NM/2}}{p\bar{p}} + \frac{T_N - T_{NM/2}}{\frac{\bar{p}^2 p}{\bar{p}p - 1}} \right) \end{aligned} \tag{3.12}$$

which is the eq. (3.10) for an arbitrary $\bar{p}$ increased by the partial parallelization of the local least squares process. The parenthesis is separating the span time $T_{NM/2}$ and the two parallelized processes with and appropriately reduced time.

Considering the above upper bound, the lower speedup bound is

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{T_{NM/2} + \frac{T_1 - T_{NM/2}}{p\bar{p}} + \frac{T_N - T_{NM/2}}{\frac{\bar{p}^2 p}{\bar{p}p - 1}}} \tag{3.13}$$

an estimation can be done by substituting the time T with the computational complexity, e.g. $T_1 = N \cdot T_N$, $T_N = \mathcal{O}(M)$, $T_{NM/2} = \mathcal{O}(\log_2 M)$

$$\begin{aligned} S_p &= \frac{N \cdot \mathcal{O}(M)}{\mathcal{O}(\log_2 M) + \frac{N\cdot\mathcal{O}(M) - \mathcal{O}(\log_2 M)}{p\bar{p}} + \frac{\mathcal{O}(M) - \mathcal{O}(\log_2 M)}{\frac{\bar{p}^2 p}{\bar{p}p-1}}} \\ &= \frac{1}{\frac{\mathcal{O}(\log_2 M)}{N\cdot\mathcal{O}(M)} + \frac{N\cdot\mathcal{O}(M) - \mathcal{O}(\log_2 M)}{p\bar{p}N\cdot\mathcal{O}(M)} + \frac{\mathcal{O}(M) - \mathcal{O}(\log_2 M)}{N\cdot\mathcal{O}(M)\frac{\bar{p}^2 p}{\bar{p}p-1}}} \end{aligned} \tag{3.14}$$

where the term $\frac{\mathcal{O}(\log_2 M)}{N\cdot\mathcal{O}(M)}$ vanishes very fast even for a low number of sample points (M). Thus, the speedup is limited only by the available resources, as following

$$\begin{aligned} S_p &= \frac{Np\bar{p}^2}{N\bar{p} + \bar{p}p - 1} \\ &= \frac{p\bar{p}}{1 + \frac{p}{N} - \frac{1}{N\bar{p}}} \end{aligned} \tag{3.15}$$

Moreover, for large number of evaluation points N it does not matter how the resources will be managed, between minimizing the span time or parallelizing the evaluation of nodes. Finally, it proves that the algorithm design is efficient and scalable.

3.4.2 The parallelization of a higher degree local least squares

In the higher degree cases of the local least squares, the span time T_N in eq. (3.4) involves higher computational complexity which cannot be managed as one entity. The analysis of the involved complexity is shown below, and further CUDA specific algorithm designs are proposed in section 3.5. The results of the current parallelization of the span time $T_{\bar{p}}$, can be used as in the previous section, in eq. (3.12) to estimate the execution time bounds.

As shown in section 2.7, first we need the evaluation of the n monomials for M sample points, which constructs a Vandermonde matrix with $\mathcal{O}(n \cdot M)$ computational complexity, further this computational cost is doubled in order to compute the weighted Vandemonde matrix V^w. Then to construct the Gramian matrix $\mathbf{A}$, we need the a matrix-matrix multiplication of the two Vandermonde matrices that requires a cost of $\mathcal{O}(n^2 \cdot M)$. The cost for the right hand side is $\mathcal{O}(n \cdot Q \cdot M)$, where Q is the number of displacement components. Summarizing the total cost to construct the normal equations of a single local least squares is

$$\mathcal{O}(2n \cdot M + n^2 \cdot M + n \cdot Q \cdot M) \tag{3.16}$$

Taking under consideration that the Gramian matrix is symmetric, the second term can be half of the computational cost, but the general order of the computational complexity is still the same.

Parallelizing the Vandermonde matrix computations

Observing the necessary computations and the patterns that can be found in eq. (2.42) and (2.43), there are quantities that can be computed in parallel per sample point and per monomial, i.e. the first term in the computational complexity $\mathcal{O}(2n \cdot M)$ can be reduced, where the computations per sample point are all data independent, and the same can be considered for the monomials to activate a massive parallel computation of the Vandermonde matrices. This design suggests assigning one computing unit per element in the Vandemonde matrices. One can observe that within the matrix, quantities are repeated, thus one approach could be to compute them once and then store them in a shared memory for subsequent access and construction of the Vandermonde matrices. Another approach which appears to be more efficient for CUDA (lower communication cost), is to ask every computing unit to evaluate for itself (and store in registers) in a serial manner the corresponding weight value, the corresponding monomial value, and at the end the required element in the Vandermonde matrix that was assigned as a task, which can be eventually stored in a shared memory. For a CUDA algorithm design, this requires the definition of a computational grid with appropriate size to fit all the monomial terms and the sample points and map one on one. Though, to continue with the subsequent computations of eq. (2.44), and (2.45) we desire the fastest possible access to the data for several computing units which will undertake the rest of the work in cooperation. That arises a constraint, as the fast shared memory is limited and is partitioned to make it available for several simultaneous groups of computing units (thread blocks in CUDA). Moreover, the size of a thread block has a certain limit size (max. 1024 threads for the current CUDA architectures), and the total number of allowable registers are very limited and partitioned for several simultaneous thread blocks. The shared resources of the shared memory and the registers, will permit an execution, but less thread blocks will occupy the multiprocessor (SM). This can be an issue for some algorithms, when the CUDA scheduler, while managing the thread blocks in the multiprocessor, will not find enough thread blocks variant in the requested instructions, to fill the full pipeline of the multiprocessor and overlap instructions of fetching and storing data or executing operations in the ALUs, resulting lower performance. All the above reasons, are limiting the size of a thread block and it can not be generally assumed that we can work on a computational grid which maps one on one to the Vandemonde matrix elements.

In section 3.5, two algorithm designs are proposed, both designs preserve all the monomials in the thread block but keep a subset of the sample points with size M_i. The size M_i is suggested to be a power of two and greater than 32, i.e. 2^k, $k \geq 5$, as the CUDA architecture is processing the instructions from groups of threads with size 32 under the model SIMT (single instruction multiple threads), i.e. the thread blocks are sliced in pieces of 32 threads when they are passed to the multiprocessor with one common instruction. The latter slicing, is done in the dimension x of the thread block, so in that dimension the sample points should be mapped, letting the dimension y for the monomials with more flexibility in size, as their size is problem defined and essential to keep the full number in the thread block. Retaining the full number of the monomials in the thread block, permits the evaluation of the subsequent Gramian matrix and the RHS matrix per sample point, without expensive communication costs to construct these matrices; recall that the algorithm is designed to store all the Vandermonde matrix elements in the shared memory of the thread block.

The available subset of the sample points per thread block, permits the partial completion of the normal equations of the local least squares. The final normal equations can be completed either by iterations within one kernel launch over M/M_i number of sample point subsets, or by subsequent summation from few subsets on the host. The former case is the first design of section 3.5.1, and the latter case is the second design of section 3.5.2, which is launched with the subsets of sample points in the same computational grid of one CUDA kernel launch, but in different block threads so that the final summation is done on the cpu when the kernel terminates.

Parallelizing the RHS matrix computations

The RHS matrix of eq. (2.45) is actually a matrix with small number of columns in the application of mesh adaptation, as it signifies the unknown functions to approximate, i.e. for 2D meshes it is 2, and for 3D meshes it is 3. From that perspective, the problem is exactly the same with what is analyzed for the 0-degree local least squares, i.e. the computation of the numerator of a weighted sum. That suggests the same efficient algorithm, which is the reduction algorithm applied 2 or 3 times according to the number of dimensions of the mesh.

The reduction algorithm will be executed in the same kernel that was launched to compute the Vandermonde matrices, with the available block size, as the necessary data of the Vandermonde matrices V and V^w, are already located in the fast shared memory and it is desirable to avoid additional communication costs to relaunch a new kernel with a different computational grid size. Unfortunatelly, that will result a large amount of threads in the block being unemployed, especially for high degrees of polynomials, as we need only few of them. Nevertheless, the resulted performance is still better than launching a new kernel, with some trials of alternative designs.

To execute the reduction algorithm for one displacement component we need $M_i/2$ number of threads, i.e. for a 2D mesh we need in total M_i number of threads in the thread block, and for a 3D mesh we need $3M_i$ threads. One of the dimensions of the thread block, as explained above, is already aligned with the size M_i and suggested to be $2^k,\ k \geq 5$ generally. The other dimension of the thread block is the one fitting the number of the monomials, which dimension keeps the redundant additional threads.

It is worth to mention, that the two reduction algorithms for the two displacement components for a 2D mesh, can be executed at the same time with one reduction algorithm for the first half of threads and the second on the second half of threads. This is not destroying the consistency of a warp of threads for the SIMT model, as the executed instructions are identical. For the 3D meshes, there will be two twins reduction algorithms and one additional subsequently resting even more threads in the block, unfortunatelly.

Parallelizing the Gramian matrix computations

For the computations of the Gramian matrix, the same constraint of the thread block size holds, i.e. the block size is defined to fit the needs of the Vandermonde evaluations, as it has been found that keeping this evaluation as well in the same CUDA kernel, gives the best performance results by avoiding extra data transferring costs, or re-evaluation of quantities.

The best performance design that has been found to compute the Gramian matrix, as presented in eq. (2.44), is through M_i iterations in each thread block, i.e. the elements of the matrix are mapped one to one to the threads in the block, where each thread accumulates the result from the available subset of M_i sample points in M_i iterations. From the first point of view, it does not seem really efficient, though the problem of summing the results of the M_i sample points, constitutes a difficult task itself without iterations. An alternative solution could be using atomic operations, but it is not faster or different than the iterations. Setting reduction algorithms for smaller subsets of the sample points M_i demand additional shared memory which is not enough for high polynomial degrees. And finally, launching a new kernel with approriate thread block size and an efficient matrix-matrix multiplication does not give better performance.

3.5 CUDA algorithm designs of the MLS method

Many details on the CUDA architecture and facts that should be considered while designing a CUDA algorithm, are already given in section 3.4.2 while proposing parallelization schemes. Here, the most important facts will be mentioned.

The processing flow, as shown in Figure 3.4 and in deeper detail, consists of

- the host instructor (a cpu) which will coordinate the process flow,
- a bidirectional communication of the GPU memory and the system memory, i.e. data can be transmitted both directions at the same time without the intervention of the cpu,
- the gpu has access to the system memory only on demand by default, some improvements on the memory access are the host pinned memory where memory pages are locked on the system memory for higher speed access from the gpu, and the unified memory which keeps a virtual memory address space avoiding the page locking,
- the gpu receives an instruction from the host cpu to run a programme, which programme is called kernel, in a single instruction multiple threads (SIMT) model, i.e. one single instruction is executed on several threads which are currying their own data, opposed to the single instruction multiple data (SIMD) model where a single instruction of one thread is implying an operation on a vector of values,
- the gpu can initiate it's own kernels in a recursive way,
- the gpu supports task parallelism by independent process streams, i.e. several kernels can be running concurrently,
- the running kernels are distributing work packages in blocks (thread blocks) over one or more multiprocessors (SM) see Figure 3.5,
- within each multiprocessor the parallel process is scheduled internally, where the process is executed in two levels, the higher level of the thread blocks, and the subsequent level of warps, both levels are scheduled optimally in order to maximize the infill of the multiprocessor's pipeline.

The last process part is the fundamental one that impacts the performance, i.e. if the reside thread blocks in the multiprocessor are few and/or with poor variety of instructions to fill the processing pipeline, the performance gets poor as the instructions cannot be overlapped. The array on the right side of Figure 3.5, comprises the pipeline of a single multiprocessor, where several computing units exist, as well as units for special operations, data communication units, etc.

One of the reasons to have few thread blocks residing in the multiprocessor (SM occupancy level), is overloading the registers in the kernel algorithm design (very limited resource), and/or overloading the shared memory (limited resource also).

Another cause of poor performance is the latencies due to excessive global memory accesses which is slow to access, or local shared memory accesses with conflicts (bank conflict issue). Moreover, branches within the warps will result latencies, as the multiprocessor will execute both ways of the branches and in each way some or all of the threads will be inactive.

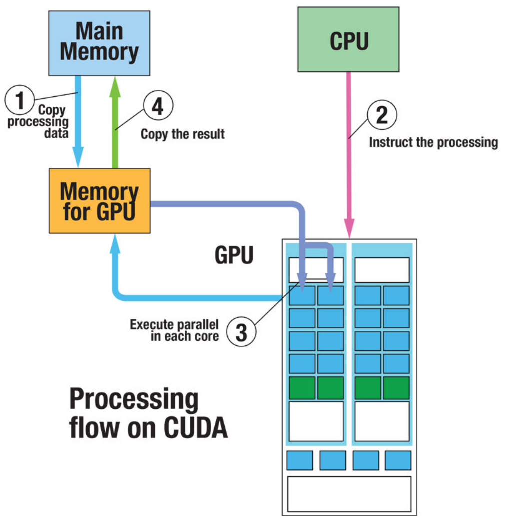


Figure 3.4: The global CUDA process flow.

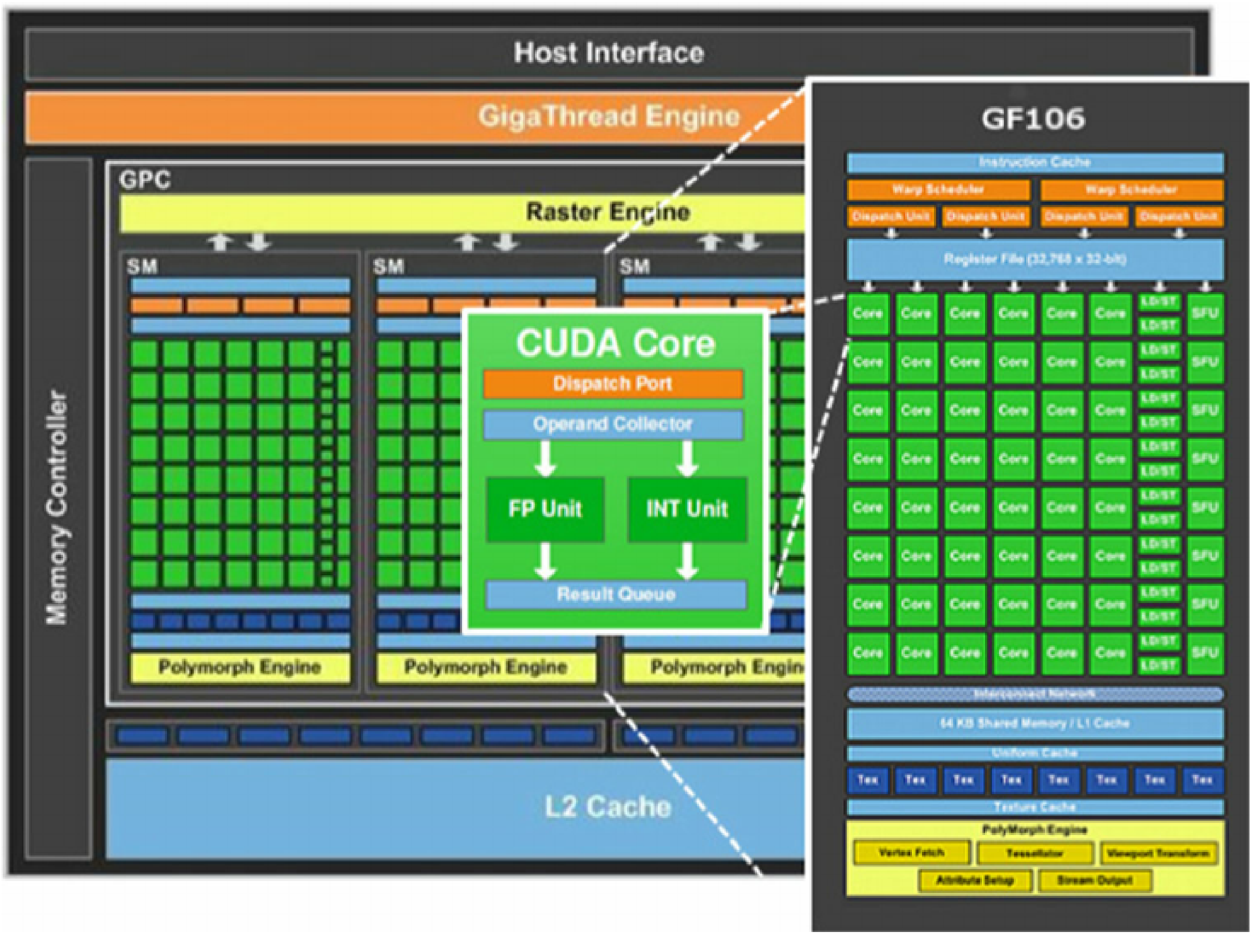


Figure 3.5: CUDA architecture diagram.

There are many problems and algorithm implementations where data has to be shared or communicated among the compute units, for such cases the memory hierarchy of the processor architecture should be taken in advantage. Explaining, regarding the Figure 3.6, the CUDA architecture memory hierachy provides a common (slow) memory space for all the kernels running on the GPU, i.e. the global memory, additionally there is an L2 cache memory (not shown in figure) for frequently used data and/or constant memory data for faster access. On each SM there is a limited amount of very fast shared memory which is being partitioned over the number of residing thread blocks. The threads of each block can access only their own partition of the shared memory, which is the space where they can share data and communicate. If threads in different blocks need to communicate, the only common space is the slow global memory. Finally, each thread that resides in a multiprocessor, has its own memory space by occupation some of the registers (extremely fast memory next to the computing units), which are shared among all the residing threads from all the blocks in the multiprocessor.

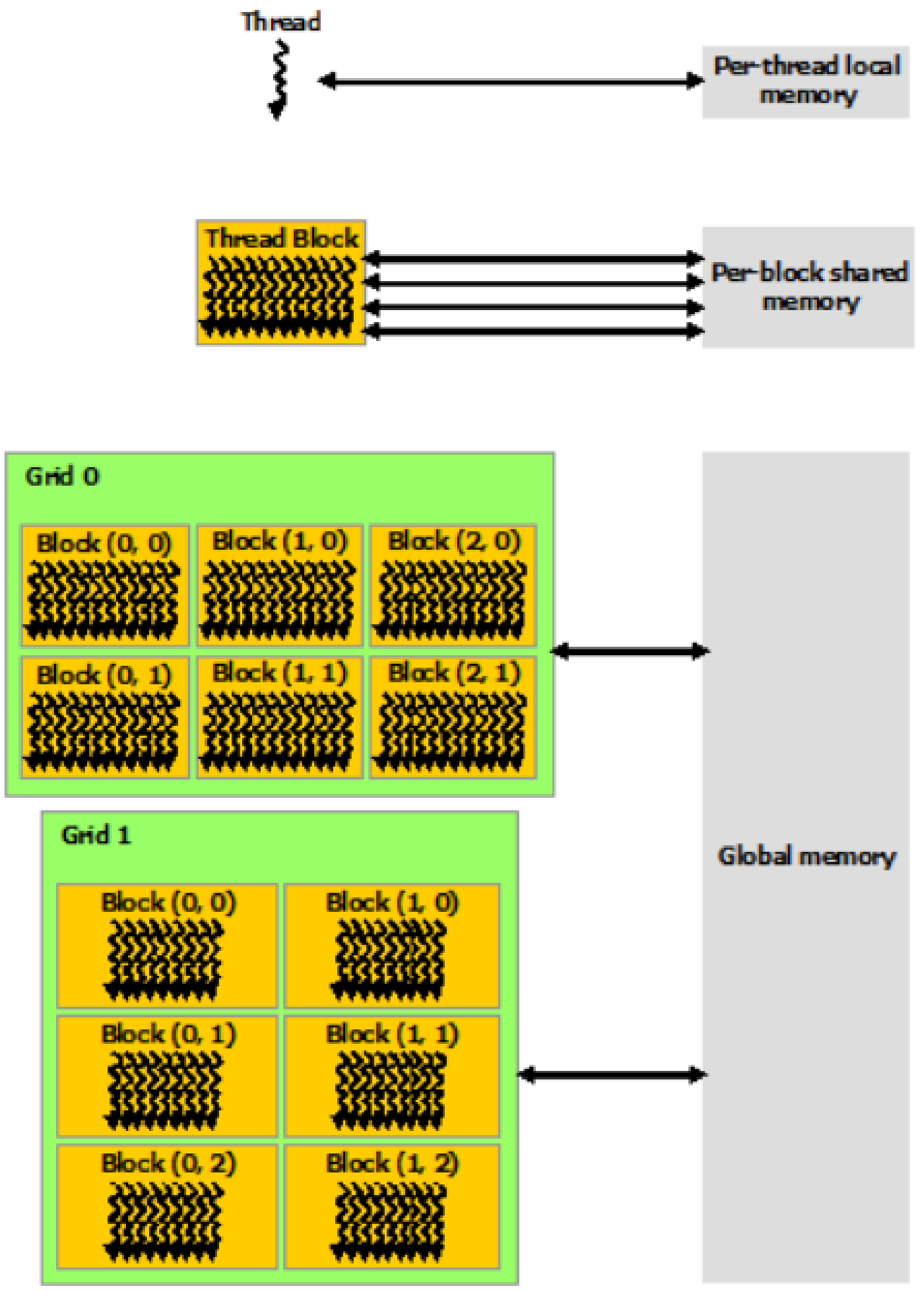


Figure 3.6: CUDA memory hierarchy.

3.5.1 CUDA algorithm design 1, A design with iterations over subsets of the sample points

This algorithm is designed to process in parallel all the evaluation points or subsets of them, which subset size can be set by a parameter of the algorithm. See table 3.1 for all the kernel launching parameters. Every kernel launch is processing all the sample points within each thread block by iterations, i.e. the size of the thread block M_i (in dimension x of the thread block) that will determine the hosted number of sample points is set by a parameter of the algorithm, highly recommended to be a power of 2, and the resulted iterations in the kernel will be M/M_i in number.

the block contains the data for all the monomials in order to be able to calculate the partial bbT which corresponds to the current sample points,

minimized data transferring by calculating bbT and bf in the same kernel launch,

several stationary points launched in a single kernel which can be controlled as a parameter

CUDA computational grid	
Grid size type	Size assigned
gridDim.x	(Parametric, optimal 2^n) Number of evaluated stationary points per kernel.
gridDim.y	1
blockDim.x	(Parametric, optimal 2^n) Size M_i of the subset of sample points per block
blockDim.y	(Problem constrained) Number of monomials
Shared memory	
$2n \cdot M_i + n^2$	
Compute streams	
1	

Table 3.1: CUDA kernel computational grid sizes and parameters, design 1

3.5.2 CUDA algorithm design 2, A design where the subsets of sample points are scattered to multiple thread blocks

This algorithm design is launching iteratively few kernels for subsets of the evaluation points (or all the evaluation points if set by a parameter), which amount of kernels is a parameter and can be adapted. A lower number of the parallel launched evaluation points can reduce the requirements of global memory, or to control the performance for GPUs with a large number of SMs. For each subset of parallel evaluated points, the kernel launches are adapting to the sample points number of the problem and the sample points set to be scattered within a thread block and several blocks in the computational grid. Thus, launching kernels with the parameters in table 3.2, may result iterations over smaller subsets of sample points if desired. Moreover, two more kernel launches may happen, if the scattering of the sample points with the chosen parameters are letting residual sample points.

As suggested in section 3.4.2, the number of the sample points in a thread block should be a number power of 2. Furthermore, this design as explained in the aforementioned section, computes the summations of the Gramian matrices and the RHS matrices partially with the available sample points in the thread block, and the last summations from different thread blocks are executed subsequently on the host. This design, requires a certain global memory amount and traffic, where the amount can be controlled as mentioned above with the size and number of sample point subsets, and the traffic seems to be not affecting the performance, as part of the computations are transferred to the host. Additionally, the part of the computations transferred to the host, is quite serialized for the Gramian matrix, as analyzed in section 3.4.2.

CUDA computational grid	
Grid size type	Size assigned
gridDim.x	(Parametric, optimal 2^n) Number of sample point subsets, M/M_i.
gridDim.y	(Parametric, optimal 2^n) Number of evaluated stationary points per kernel
blockDim.x	(Parametric, optimal 2^n) Size M_i of the subset of sample points per block
blockDim.y	(Problem constrained) Number of monomials
Shared memory	
$2n \cdot M_i + n^2$	
Compute streams	
1	

Table 3.2: CUDA kernel computational grid sizes and parameters, design 2

3.5.3 CUDA algorithm design 3, A design with matrix-matrix multiplications on the cuBLAS library

This version of algorithm is the simplest possible to implement, by implementing directly the matrix-matrix multiplications of eq. (2.44) and (2.45), to derive the Gramian matrix and the RHS matrix. The appropriate cuBLAS library function for the matrix-matrix multiplications, is the *cublasGgemm*.

To launch the cuBLAS function there is no need to define manually a CUDA computational grid of threads. The function is launched only by passing the pointers to the input matrices data, located in the device global memory. Furthermore, as parameters to the launched function it should be passed, the sizes of the matrices, and the parameters α and β of the following executed operation

$$C = \alpha op(A) op(B) + \beta C$$

where we need $\alpha = 1$ and $\beta = 0$. For more details the NVIDIA CUDA documentation is the most complete source.

The cuBLAS function is launched once to derive the Gramian matrix, and twice to compute the RHS for the x and y displacements components, separately, per evaluation point (i.e. an interior mesh node). To enable the parallel execution of the 3 function launches per single evaluation point, and generally for the concurrent evaluation of many evaluation points, multiple computing streams are used which are able to run independent CUDA kernels concurrently.

To launch the above computations the Vandermonde matrices should be first evaluated for a sufficent number of evaluation points that will be evaluated concurrently. Thus, a custom kernel is doing the preprocessing work, under instructions of parallel computations found in section 3.4.2. The parameters of the launched custom kernel are found in table 3.3.

The performance of this design is not sufficient, and there is no control or insight of the computations and the computational grid. A number of concurrent streams, larger than 25, gives no additional speedup for a gpu with compute capabilities 3.0, Quadro K1000M, with 1 SM and 192 CUDA cores.

CUDA computational grid	
Grid size type	Size assigned
gridDim.x	2.
gridDim.y	(Parametric, optimal 2^n) Number of evaluated stationary points per kernel
blockDim.x	(Parametric) Number of computing streams (32)
blockDim.y	(Problem constrained) Number of monomials
Shared memory	
Automatic in cuBLAS	
Compute streams	
32	

Table 3.3: CUDA kernel computational grid sizes and parameters, design 3, a preprocessor kernel to compute the Vandermonde matrices for the matrix-matrix multiplications with cuBLAS.

Chapter 4

Performance results of the MLS method on CUDA GPGPUs

4.1 2D test case 1

The coarse grain parallelization level of the method is very easy to implement for any type of available parallel processors, i.e. by scattering the completely independent data to several processors and following the SPMD model. Therefore, for this parallelization level, results from CPU multi-core execution are included in the performance comparison charts. Results from heteregenous processing, by sharing the evaluted points between the multi-core CPU and the GPGPU, are presented as well.

All the compared cases are summarized as following

- CPU single thread, reference time as a serialized process*
- CPU 8 threads/ 4 cores
- CUDA algorithm design 1 (single/double precision)
- CUDA algorithm design 2 (single/double precision)
- CUDA algorithm design 3 (single/double precision)
- 50% work for a CPU with 7 threads and 50% work for the CUDA algorithm design 1

*the process in this case is partially parallelized with SIMD instructions by automatic optimization from the compiler (-mavx flag enabled), augmented also by ILP, as the CPU is a pipelined superscalar processor.

The hardware used for this comparison is:

- CPU Intel Core i7-3740QM Quad core
- GPGPU Quadro K1000M, 1 SM, 192 CUDA cores, Kepler architecture, Compute capability 3.0

The problem solved for the comparison is the one defined in section 3.3.

Below, graph representations can be found comparing the performance of each parallelization scheme as defined above. The performance of each scheme is measured for various polynomial degrees (0-6), in single and double arithmetic precision. The importance of the polynomial degree is presented in section C.3.2, where higher mesh quality solutions can yield and higher robustness is provided over the method's parameters. Furthermore, for higher degree polynomial solutions, i.e. from degree 5 and above, the single precision arithmetics are of insufficient precision to give solutions, thus higher degrees (5+) should be combined with slower double precision arithmetics on the GPU.

In figure 4.1, the global speed of the solution is illustrated for the various analysed cases. Cases of higher polynomial degree were the bar is missing, is when the algorithm executed in excessively large time.

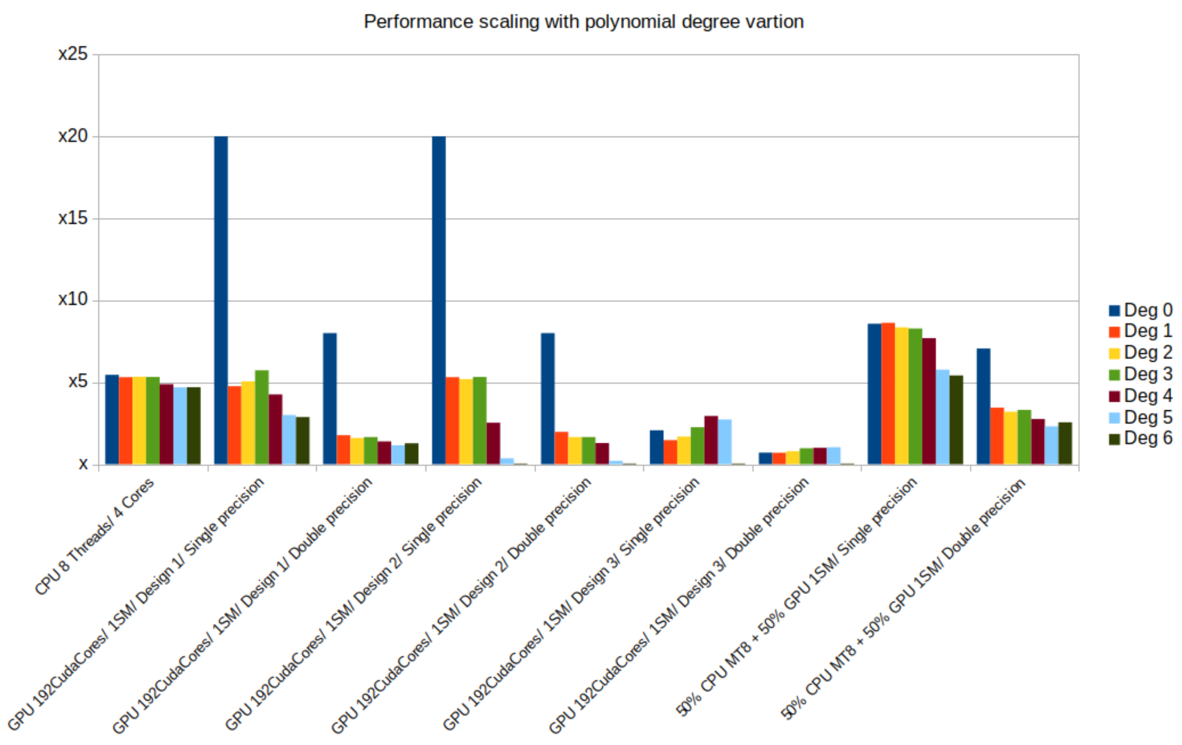


Figure 4.1: Comparison graph of the performance speedup.

In figure 4.2 and 4.2, the execution time of each case is shown in (ms). Again cases assigned with a very large execution time, are considered as non acceptable solutions.

The very high performance of the 0-degree algorithm was expected, as it is designed with the efficient reduction algorithms, see 3.4.1. The x20 speedup that we can see, it should be higher if the SIMD instructions are disabled on the CPU. Moreover, running the same algorithm on a powerful GPU (with 10+ SM) this speedup should scale linearly to very large values.

For the rest of the polynomial degrees, the results could be characterized good (considering the weak GPU and the strong CPU) for a polynomial degree up to 4. Though the proposed algorithm designs are not efficient, which becomes apparent when the problem scales in size (higher polynomial degree) the speedup drops. The latter fact, exhibits the fact that the second CUDA algorithm design is less efficient, as the speedup drops faster and higher polynomial degrees are not able to accomplish in an acceptable time. Though, both of the custom CUDA algorithm designs, are parametrized on the size of the data packages being processed, a deeper investigation may give better results.

The difference between the single and double precision solutions is the expected one, less than half of the perfomance for the slower double precision arithmetics, as exactly specified from NVIDIA.

Regarding the algorithm design 3 (implemented with the cuBLAS), although the implementation is very easy and appealing, it performs poorly which suggests alternative designs. A good try could be launching recursive kernels from a kernel (during this thesis no available hardware was found with higher CUDA compute capabilities to allow this).

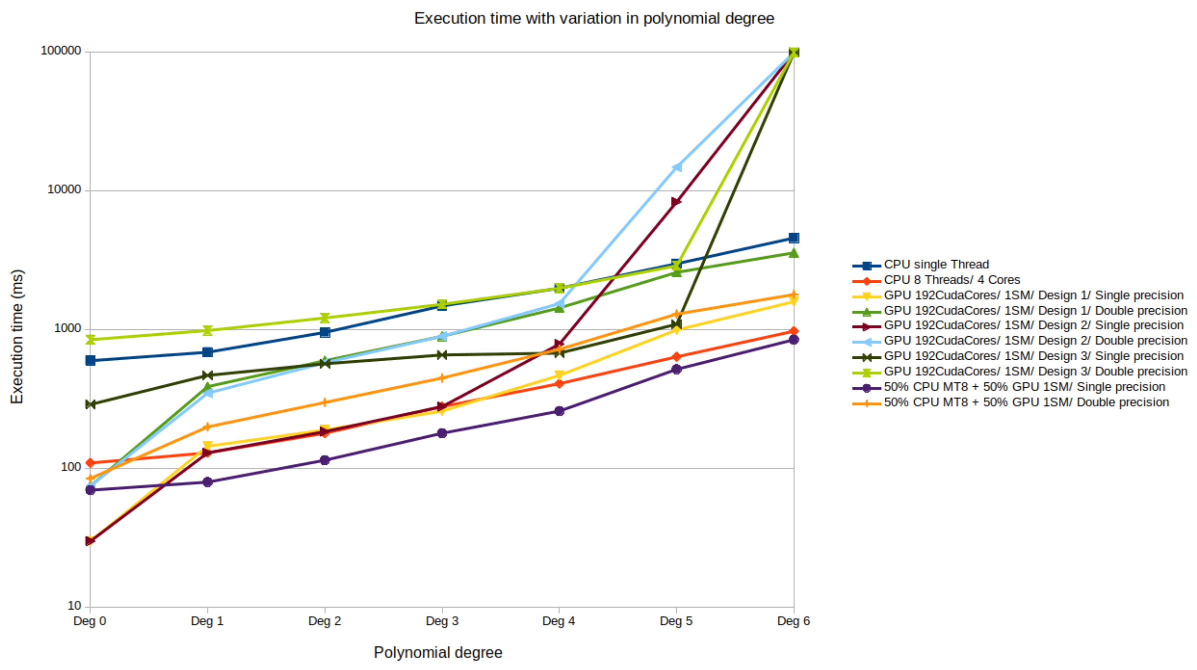


Figure 4.2: The execution time evolution of various parallel schemes, with the polynomial degree.

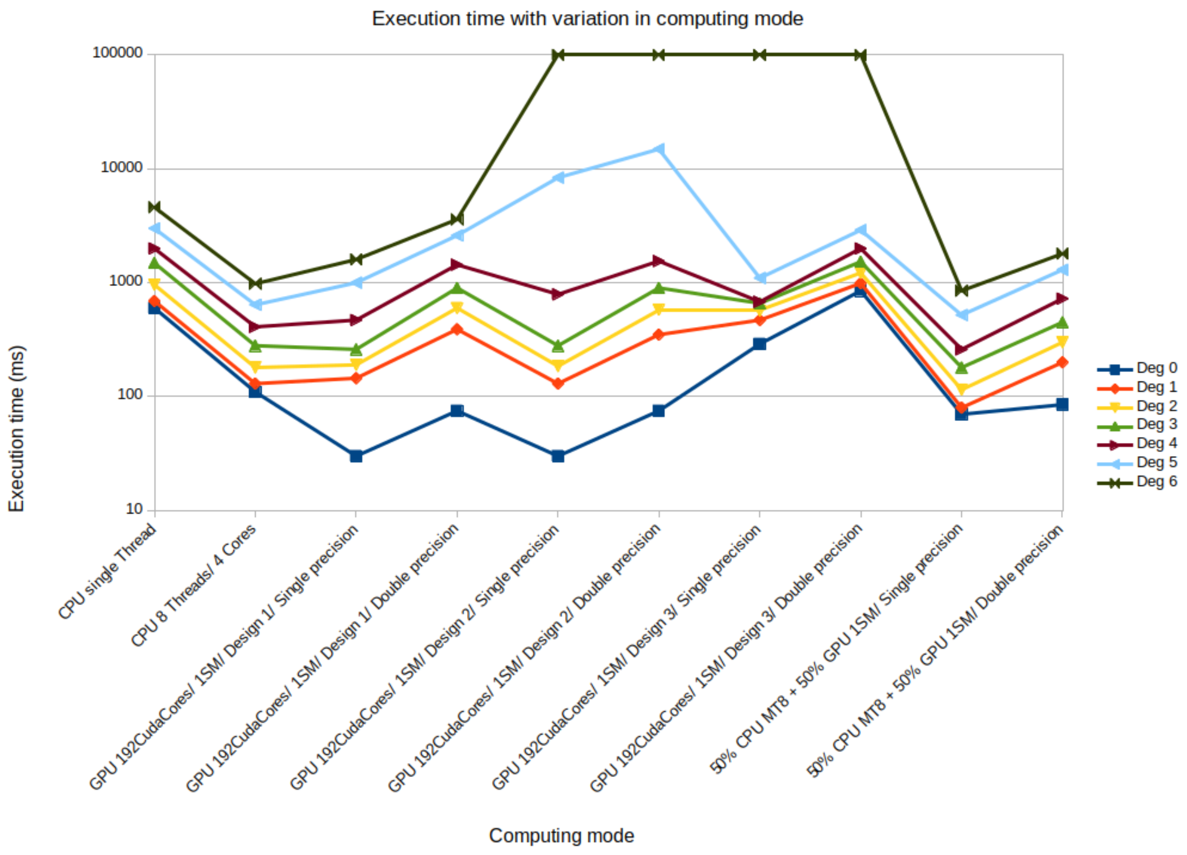


Figure 4.3: The execution time per polynomial degree for various parallel schemes.

Chapter 5

Conclusions

There are plenty problems in computational mechanics which are solved on GPGPUs and the mesh deformation stage exists in the process. Executing the mesh deformation on the same high throughput compute devices would reduce the solving time further. The MLS method exposes a perfect data independent coarse level parallelism, and a less parallelizable fine grain parallelism that needs careful algorithm design.

Despite the fact that the importance of the higher polynomial degree MLS, for largely deformed meshes with acceptable mesh quality, is acknowledged, there are two important issues for the GPGPUs which degrade the performance. The fine grain parallelism becomes a harder challenge when the MLS polynomial degree increases. There the required data communication becomes more intensive, resulting lower performance. Balanced solutions should be considered, where recomputations or iterations may prove more efficient, compared to data sharing and communicating. Second, for very high polynomial degrees (e.g. degree 5 and above), the single precision arithmetics are insufficient to give valid solutions. Hence, higher degree MLS executions should combined with the slower double precision arithmetics on the GPGPU, dropping the performance even more. Nevertheless, the produced speedups presented in this work, are expected to be much higher with powerful GPGPUs. In contrary, the parallelization of the MLS method is an easy task for the simple 0-degree case. The obtained results from the parallelization of the 0-degree MLS method in real applications, are as efficient as were designed and theoretically expected.

The developed algorithms, generally expose good compute performance with very high speedups for low polynomial degree MLS interpolations, e.g. up to x20 in the case of the zero degree MLS method (known also as the Shepard's method, or the Inverse Distance Weighting method). The speedups for higher polynomial degrees are moderate, i.e. x5, but the utilized hardware comprised of a very strong CPU and a GPGPU with poor compute resources.

Appendix A

Elements from the approximation theory

A.1 Approximation of functions

The approximation theory is concerned with how functions f can be approximated with simple functions $\hat{f}$, and with estimating the upper bounds of the introduced errors ϵ, or even evaluating the error in some cases. Furthermore, making approximations with a computer restricts one to use simple functions from the class of polynomials, which are easy to handle and evaluate numerically. The implicated accuracy does not have to be higher than the underlying computer's floating point arithmetic.

The space of polynomials $\mathbb{P}_n$ is of degree less than or equal to n and it is spanned by the basis $span\{1, x, ..., x^n\}$ of dimension $\dim \mathbb{P} = n + 1$.

In approximation of continuous functions f with polynomials, i.e. $\hat{f} = p_n \in \mathbb{P}_n$, there is a fundamental theorem that guarantees the existence of an arbitrary good approximation uniformly along the domain, which means there is a max error bound uniform for all the points, as following

Weierstrass Theorem
Let $f \in \mathcal{C}[a, b]$. Then for any $\epsilon > 0$ there exists a polynomial $p_n \in \mathbb{P}_n[a, b]$ for which

$$\|f - p_n\|_\infty \leq \epsilon \tag{A.1}$$

There are many proofs of the theorem above, with an interesting proof construction coming from *S. Bernstein*, which proves the existence of the uniform approximation polynomials and provides them in a simple form, see Davis [1975]. The same theorem holds for functions $f(x)$ defined on real domains with *N-dimensions*, $x \in \Omega, \Omega \subset \bar{\mathbb{R}}^N$.

The same results of approximation can hold even when the approximation is forced to interpolate some values on distinct points. The following theorem guarantees the uniform approximation.

Walsh Theorem
Let $\bar{\Omega} \subset \mathbb{R}^N$ a compact set, $\omega_m = \{x_i | x_i \in \bar{\Omega}, i = 1..m\}$ a set of m distinct points, $f \in \mathcal{C}(\bar{\Omega})$, and $p_n \in \mathbb{P}_n(\bar{\Omega})$. Assume that $f(x)$ is defined on $\bar{\Omega}$ and is uniformly approximable by polynomials.
Then it is uniformly approximable by polynomials p_n that satisfy the constraints

$$p(x_i) = f(x_i), \quad i = 1..m \tag{A.2}$$

Stone trying to prove the *Weierstrass Theorem*, showed that using algebra of functions from the same function space of the function f, one can converge arbitrarily close to the function f as soon as some properties hold.

Stone-Weierstrass Theorem
Let $\bar{\Omega} \subset \mathbb{R}^N$ a compact set, $f \in \mathcal{C}(\bar{\Omega})$, $\hat{f} \in \mathcal{S}(\bar{\Omega})$, where $\mathcal{S}$ is a subspace of $\mathcal{C}$, $\mathcal{C}$ is equipped with a metric $\rho(\mathcal{C})$, and the following properties hold

- $\mathcal{S}$ contains all the constant functions
- $\forall \hat{f}_1, \hat{f}_2 \in \mathcal{S}, a \in \mathbb{R} \Rightarrow \hat{f}_1 + \hat{f}_2 \in \mathcal{S}$, $a\hat{f}_1 \in \mathcal{S}$, $\hat{f}_1\hat{f}_2 \in \mathcal{S}$
- $\forall x_1, x_2 \in \Omega | x_1 \neq x_2$, there exists $\hat{f} \in \mathcal{S}$ such that $\hat{f}(x_1) \neq \hat{f}(x_1)$

Then $\mathcal{S}(\bar{\Omega})$ is dense in $\mathcal{C}(\bar{\Omega})$, i.e. $\forall f \in \mathcal{C}(\bar{\Omega})$ there is a sequence $\{\hat{f}_n\} \subset \mathcal{S}$ such that

$$\left\| f - \hat{f}_n \right\|_{\rho(\mathcal{C})} \to 0 \quad as \quad n \to \infty \tag{A.3}$$

The result is more general and the first theorem is just a case where the used subspace of functions $\mathcal{S}$ is of polynomials $\mathbb{P}$, and the chosen metric is the supremum $\|.\|_\infty$.

$\square$

A.2 Best approximation

Note that the *Stone-Weierstrass* theorem, for all $f \in \mathcal{C}(\bar{D})$ with $\bar{D} \subset \mathbb{R}^d$, guarantees the existance of an appropriate solution space $\mathcal{S} \subset \mathcal{C}$ which contains a desired approximation for an arbitrary small error.

The notion of the *best approximation*, is related to a specifically given function $f \in \mathcal{C}(\bar{D})$, a specific solution space $\mathcal{S} \subset \mathcal{C}(\bar{D})$, and a specific metric $\rho(\mathcal{C}(\bar{D}))$. In such a specific framework, the *best approximation*, is the solution with the minimum possible error, i.e. $\hat{f}_b \in \mathcal{S}$, such that

$$\left\| f - \hat{f}_b \right\|_{\rho(\mathcal{C})} \leq \left\| f - \hat{f} \right\|_{\rho(\mathcal{C})}, \forall \hat{f} \in \mathcal{S} \tag{A.4}$$

With a metric equipping the function space $\mathcal{C}$, one can measure how close the functions f and $\hat{f}$ are, which is the base to define optimality criteria to minimize that measurement.

Evidently, the task of finding the *best approximation* is a minimization problem. The objective function to this minimization problem is the measuring function of the error, as following

$$E(\hat{f}) = \left\| f - \hat{f} \right\|_{\rho(\mathcal{C})}, \hat{f} \in \mathcal{S} \tag{A.5}$$

Some important questions regarding the *best approximation* are, if it always exists and it is unique, or under which circumstances it is unique, the ability to estimate it, and if it has any asympotic properties as one expands the solution space.

A.2.1 Existence of a best approximation

As stated in the previous section, the best approximation is a minimization problem of the function $E(\hat{f}) : \mathcal{S} \to \mathbb{R}$, i.e. a minimization of a norm function among the elements in the solution space $\mathcal{S}$, see Eq. A.5.

To justify the existence of a solution of a minimized function, we need the property of convexity for the solution and the minimized function itself.

Assume $\mathcal{C}$ is a complex linear space $\mathcal{S} \subset \mathcal{C}$. Then $\mathcal{S}$ is convex if for any $\hat{f}_1, \hat{f}_2 \in \mathcal{S}$

$$\begin{gathered} \lambda \hat{f}_1 + (1 - \lambda)\hat{f}_2 \in \mathcal{S} \\ \forall \lambda \in (0, 1) \end{gathered} \tag{A.6}$$

Generally, the linear combination

$$\sum_{i=1}^{n} \lambda_i \hat{f}_i$$

is called a convex combination of a set $\{\hat{f}_i\}$

$$\textit{when} \quad \lambda_i \geq 0 \quad \textit{and} \quad \sum_{i=1}^{n} \lambda_i = 1$$

Assume $\mathcal{S}$ is a convex subset of a linear space $\mathcal{C}$, then a function $E(\hat{f}) : \mathcal{S} \to \mathbb{R}$ is said to be convex if

$$E\left(\lambda \hat{f}_1 + (1-\lambda)\hat{f}_2\right) \leq \lambda E(\hat{f}_1) + (1-\lambda)E(\hat{f}_2) \qquad \forall \hat{f}_1, \hat{f}_2 \in \mathcal{S}, \quad \forall \lambda \in [0,1] \tag{A.7}$$

The norms as functions are generally convex and continuous. Furthermore, if we are dealing with unbounded solution spaces $\mathcal{S}$ then the norm functions are also coercive. In practice the solution spaces are finite-dimensional so we have the following results.

Assume $\mathcal{C}$ a normed space, and $\mathcal{S}_n \subset \mathcal{C}$ is a closed and convex subset with n finite. Then given an $f \in \mathcal{C}$, there is an element $\hat{f}_b \in \mathcal{S}_n$ such that

$$\left\| f - \hat{f}_b \right\|_{\rho(\mathcal{C})} = \inf_{\hat{f} \in \mathcal{S}_n} \left\| f - \hat{f} \right\|_{\rho(\mathcal{C})} \tag{A.8}$$

Briefly, the convexity and the boundedness of a function space $\mathcal{S}_n$, can guarantee the existence of the *best approximation*. The space of polynomials $\mathbb{P}_n$, as defined in the first section of this chapter, is a finite dimensional, convex space.

For problems where a function synthesis (or reconstruction) is required out of a set of points with provided values k in number, we have the following corollary for the best uniform approximation.

Let $\mathcal{C}[a,b]$ be a closed space equiped with the norm $\|.\|_\infty$, $\mathcal{S} = \mathbb{P}_n$ with $n \geq 0$, and $f \in \mathcal{C}$. Furthermore, a set of points $\mathcal{P} = \{x_0, x_1, ..., x_k\}$ is given with $k \geq n$. Then there is a solution $\hat{f}_b \in \mathbb{P}_n$ to the minimization problem

$$\min_{\hat{f} \in \mathbb{P}} \left\| f - \hat{f} \right\|_\infty \tag{A.9}$$

$$\textit{find } \{a_i\}_{i=0}^n \textit{ such that} \quad \min \max_{0 \leq i \leq k} |f(x_i) - \sum_{j=0}^n a_j x_i^j|$$

The problem of an overdetermined system of linear equations belongs here as well.

Given p in number sets of $\{a_{ij}, f_j\}_{i=0}^n$, $0 \leq j \leq p$, and $p > n$. The following problem has a solution

$$\textit{find } \{x_i\}_{i=0}^n \textit{ such that} \quad \min \max_{0 \leq i \leq p} |y_i - \sum_{j=0}^n a_{ij} x_j|$$

A.2.2 Uniqueness of the best approximation

For the uniqueness, the minimized function needs the property of the strict convexity. Following the definition of the convexity of a function in the previous section, if additionally $\hat{f}_1 \neq \hat{f}_2$ when $\lambda \in (0,1)$, then the function $E(\hat{f})$ is strictly convex.

The uniqueness of the *best approximation* is based on the following two theorems

Assume $\mathcal{C}$ is equiped with a norm $\|.\|_q$, and $\mathcal{S} \subset \mathcal{C}$ is a convex subset. Given that the function $\|.\|_q^p$ is strictly convex for some $p \geq 1$, then the best approximation $\hat{f}_b \in \mathcal{S}$ is unique for the minimization problem

$$\min_{\hat{f} \in \mathcal{S}} \left\| f - \hat{f} \right\|_q^p \tag{A.10}$$

A space is strictly normed when $\|u + v\| = \|u\| + \|v\|$, and if for $u \neq 0$ and a non-negative scalar λ it is $v = \lambda u$.

Assume $\mathcal{C}$ is a strictly normed space, $\mathcal{S} \subset \mathcal{C}$ is a non-empty convex subset, and $f \in \mathcal{C}$. Then the best approximation $\hat{f}_b \in \mathcal{S}$ is unique.

Two very important examples of such metric spaces, are the inner product space and the Lebesque spaces $\mathcal{L}^p(\Omega)$ equiped with a norm $\|.\|_p$, $1 \leq p < \infty$. The two example function spaces, are both strictly convex and strictly normed, which fact implies the existence of a unique solution in the best approximation problem.

The spaces $L^p(\Omega), \quad 1 \leq p < \infty$ are Banach spaces and they are concrete realizations of the abstract completion of $\mathcal{C}(\bar{\Omega})$ under the norm

$$\|f\|_p = \left[\int_\Omega |f(x)|^p dx \right]^{1/p}$$

The best approximation minimization problem on the $\mathcal{L}^p(\Omega)$ spaces, is defined with the objective function

$$E(\hat{f}) = \left\| f - \hat{f} \right\|_{\mathcal{L}^p(\Omega)}^p$$

The latter norm function, is strictly convex on the space $\mathcal{L}^p(\Omega)$, therefore it has a unique solution.

The space $L^\infty(\Omega)$ is also a Banach space, but it is much larger than the space $\mathcal{C}(\bar{\Omega})$ with the ∞-norm

$$\|f\|_\infty = ess \sup_{x \in \Omega} |f(x)|$$

The norm $\|.\|_{L^\infty}$ is not strictly convex, yet there are classical results stating the existance of a best uniform approximation for important function classes, e.g. Chebyshev equi-oscillation theorem, which is

Let $\mathcal{C}[a,b]$ be a closed space, $\mathcal{S} = \mathbb{P}_n$ with $n \geq 0$, and $f \in \mathcal{C}$. Then there is a unique solution $\hat{f}_b \in \mathbb{P}_n$ to the minimization problem

$$\min_{\hat{f}\in\mathbb{P}_n} E(\hat{f}) = \min_{\hat{f}\in\mathbb{P}_n} \left\| f - \hat{f} \right\|_\infty \tag{A.11}$$

For a set of n+2 points, $a \leq x_0 < x_1 < ... < x_{n+1} \leq b$, the uniqueness implies that

$$f(x_j) - \hat{f}_b(x_j) = (-1)^j \min_{\hat{f}\in\mathbb{P}_n} E(\hat{f}) \quad j = 0, 1, 2, ..., n+1 \tag{A.12}$$

A.2.3 The best approximation in inner product spaces

The inner product spaces, are the linear spaces where the inner product can operate on any two elements. The inner product is a generalization of the canonical dot product in $\mathbb{R}^3$, i.e.

$$(x, y) = \sum_{i=1}^{3} x_i y_i, \quad \forall x, y \in \mathbb{R}^3$$

i.e. the inner product estimates the covariance of two elements in the space.

A complete inner product space is called Hilbert space. Or in other words, an inner product space is a Hilbert space, if it is a Banach space under the inner product norm. A useful example of a Hilbert space is the $L^2(\Omega)$ space, which is an inner product space with the cannonical inner product (the sum of the component products, where the components correspond to the domain points)

$$(f_1, f_2) = \int_\Omega f_1(x) f_2(x) dx$$

This inner product yields the $L^2(\Omega)$ norm (as defined in the previous section)

$$\|f\|_2 = (f, f)^{1/2} = \left[\int_\Omega |f(x)|^2 dx \right]^{1/2}$$

In the previous section, the unique solution of the best approximation problem on strictly normed and strictly convex spaces has been stated. Furthermore, the inner product space has been brought as an example of such spaces.

Formally, setting a framework where the Hilbert space $\mathcal{H}$, is a real valued metric space, equiped with an inner product induced norm, we can write the following Lemma for the uniqueness of a best approximation.

Let $\mathcal{S}$ be a convex subset of a real inner product space $\mathcal{C}$. For any $f \in \mathcal{C}$, $\hat{f}_b \in \mathcal{S}$ is its best approximation in $\mathcal{S}$ if and only if it satisfies

$$(f - \hat{f}_b, \hat{f} - \hat{f}_b) \leq 0, \quad \forall \hat{f} \in \mathcal{S} \tag{A.13}$$

Using the proposition of the existence of the best approximation from the previous section we have the theorem.

Let $\mathcal{S}$ be a non-empty, closed, convex subset of a Hilbert space $\mathcal{C} = \mathcal{H}$. For any $f \in \mathcal{C}$, there is a unique solution $\hat{f}_b \in \mathcal{S}$ from the following minimization

$$\min_{\hat{f} \in \mathcal{S}} \left\| f - \hat{f} \right\| \tag{A.14}$$

We call the *best approximation* $\hat{f}_b$ the projection of f onto the closed convex set $\mathcal{S}$, and write $\hat{f}_b = P_{\mathcal{S}}(f)$. The operator $P_{\mathcal{S}} : \mathcal{C} \to \mathcal{S}$ is called the projection operator of $\mathcal{C}$ onto $\mathcal{S}$. In general $P_{\mathcal{S}}$ is a nonlinear operator, particularly when $\mathcal{S}$ is a closed subspace, the projection operator is linear.

The projector $P_{\mathcal{S}}$ has the following two properties

- it is monotone

$$(P_K(f_1) - P_K(f_2),\ f_1 - f_2) \geq 0 \quad \forall f_1, f_2 \in \mathcal{C} = \mathcal{H}$$

- it is non expansive

$$\|P_K(f_1) - P_K(f_2)\| \leq \|f_1 - f_2\| \quad \forall f_1, f_2 \in \mathcal{C} = \mathcal{H}$$

The last uniqueness theorem of the best approximation, is also true for convex and closed finite dimensional subsets of inner product spaces.

□

A.3 The orthogonal projection operator

The theorem on the best approximation uniqueness, in the last section, holds also for complete subspaces of inner product spaces, which are Hilbert subspaces, and the best approximation $\hat{f}_b \in \mathcal{S}$ is characterized by the property

$$(f - \hat{f}_b, \hat{f}) = 0 \quad \forall \hat{f} \in \mathcal{S}$$

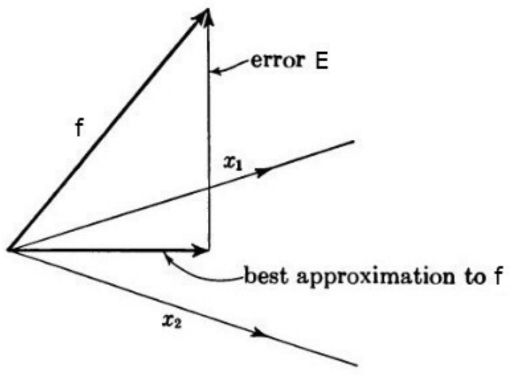


Figure A.1: Simplified geometrical illustration of the best approximation in inner product spaces.

A geometric interpretetion of this property, under the assumption that the solution space is orthonormal of dimension two, can be illustrated in Fig. A.1. There the solution space $\mathcal{S}$ is represented as a plane and each point of the plane is a candidate solution. The approximated function f is in a more rich space out of the plane, keeping a distance, the error $f - \hat{f}_b$, which is orthogonal to the subspace $\mathcal{S}$. The projection mapping P_S is then called an orthogonal projection operator. The properties of the orthogonal projection operator are summarized in the next theorem.

Assume $\mathcal{S}$ is a complete subspace of an inner product space $\mathcal{C}$. Then the orthogonal projection operator $P_S : \mathcal{C} \to \mathcal{C}$ is linear and self-adjoint i.e.

$$(P_S(f_1),\ f_2) = (f_1, P_S(f_2)) \quad \forall f_1, f_2 \in \mathcal{C}$$

In addition (Pythagorean theorem)

$$\|f\|^2 = \|P_S(f)\|^2 + \|f - P_S(f)\|^2 \quad \forall f \in \mathcal{C}$$

and as a consequence

$$\|P_S\| = 1$$

An important special situation arises when the solution space $\mathcal{S}_n \subset \mathcal{C}$ is defined with an orthonormal basis $span\{\phi_0, ..., \phi_n\}$. Then the element $P_{Sn}(f) = \hat{f}_b \in S_n$ is the solution of the minimization problem

$$\min_{\hat{f} \in S_n} \left\| f - \hat{f} \right\|$$

$$\Leftrightarrow E(f; a_i) = \left\| f - \sum_{i=0}^{n} a_i \phi_i \right\|^2, a_0, ..., a_n \in \mathbb{R}$$

From which we obtain the identity

$$E(f;a_i) = \|f\|^2 - \sum_{i=0}^{n} |(f,\phi_i)|^2 + \sum_{i=1}^{n} |a_i - (f,\phi_i)|^2$$

Clearly the minimum of $E(f;a_i)$ is achieved by letting $a_i = (f,\phi_i), i = 0,...,n$. Thus the orthogonal projection of f into $\mathcal{S}_n$ is given by

$$P_{Sn}(f) = \sum_{i=0}^{n} (f,\phi_i)\phi_i$$

since

$$\|f - P_{Sn}(f)\| = \min_{\hat{f}\in\mathcal{S}_n} \left\| f - \hat{f} \right\| \to 0 \quad as\ n \to \infty$$

we have the expansion

$$f = \lim_{n\to\infty} \sum_{i=0}^{n} (f,\phi_i)\phi_i = \sum_{i=0}^{\infty} (f,\phi_i)\phi_i$$

where the limit is understood in the sense of the norm $\|.\|$. The quantity $P_{Sn}(f)$ is also called the least squares approximation of f by the elements of $\mathcal{S}_n$.

A.4 Orthogonal polynomials

A very important application of the orthogonal projectors is the least squares approximation of functions by polynomials.

Let $\mathcal{C} = L^2(-1,1)$, and $\mathcal{S}_n = \mathbb{P}_n(-1,1)$ the space of polynomials of degree less than or equal to n. Then an orthonormal polynomial basis for $\mathcal{C}$ is known, $\{\phi_n \equiv L_n\}_{n\geq 0}$, the normalized Legendere polynomials.

$$L_n(x) = \sqrt{\frac{2n+1}{2}} \frac{1}{2^n n!} \frac{d^n}{dx^n}[(x^2-1)^n], \quad n \geq 0 \tag{A.15}$$

For any $f \in \mathcal{C}$ its best approximation from $\mathbb{P}_n(-1,1)$ is given by

$$P_{Sn}(f) = \sum_{i=0}^{n} (f,L_i)_{L^2(-1,1)} L_i(x)$$

More generally, let $w(\Omega)$ is a positive weight function and integrable on $\Omega = [-1,1]$. We use the interval $[-1,1]$; all other finite intervals $[a,b]$ can be converted to $[-1,1]$ by a simple linear change of variables. Then we can define the L^2 weighted function space

$$L^2_w(\Omega) = \{f \mid f\ measurable,\ \int_\Omega |f(x)|^2 w(x)dx < \infty\}$$

is a Hilbert space with the inner product

$$(f_1,f_2)_{0,w} = \int_\Omega f_1(x)f_2(x)w(x)dx, \quad \forall f_1,f_2 \in L^2_w(\Omega)$$

and the norm

$$\|f\|_{0,w} = \sqrt{(f,f)_{0,w}}$$

Then again, two functions are said to be orthogonal if the inner product vanishes $(f_1, f_2)_{0,w} = 0$.

The Gram-Schmidt procedure with the monomials $\{1, x, x^2, ...\}$ can be applied to construct a system of orthogonal polynomials $p_n(x)_{n=0}^{\infty}$, such that the degree of p_n is n. For any $f \in L^2_w(-1,1)$, the best approximating polynomial of degree less than or equal to N is

$$P_N(f) = \sum_{n=0}^{N} \xi_n p_n(x), \quad \xi_n = \frac{(f,p_n)_{0,w}}{\|p_n\|^2_{0,w}}, \quad 0 \le n \le N \tag{A.16}$$

The best approximation $P_N u$ is charcacterized by the property that it is the orthogonal projection of u onto the polynomial space $\mathbb{P}_N(-1,1)$ with respect to the inner product $(.,.)_{0,w}$.

A generalization of the Legendre orthogonal polynomials and the first kind Chebyshev polynomials, are the Jacobi polynomials related to the weight function $w^{(\alpha,\beta)}(x) = (1-x)^{\alpha}(1+x)^{\beta}, -1 \le \alpha, \beta \le 1$. for $\alpha = \beta = 0$ it yields the Legendre polynomials, for $\alpha = \beta = -1/2$ it yields the first kind Chebyshev polynomials.

□

Appendix B

Weighting functions for local weighted least squares approximations

B.1 Weighting functions in general rational form

As analyzed in section 2.4, the weighting function has a fundamental role in the local least squares approximation; primarily allowing the existence of the approximation, and further controlling the character of the result, e.g. the degree of locality, which locality is closely related to the order of the approximation (i.e. how good it is), the smoothness, the interpolation, etc.

Below we write again the characteristics that the weighting function should possess in order to be functional:

1. It should be a positive and continuous function
2. Compact support, i.e. $w(x-\bar{x}) = 0$, $x \notin B(\bar{x}, r)$, or in a weaker case $w \to 0$, as $x \to \infty$
3. Rapidly decaying
4. Integrable

Some plots of the used weighting functions in this work, are illustrated in Figure B.1.

The general form of the weighting functions in local least squares, is the following rational

$$w(x) = \frac{Q(x)}{P(x)} \tag{B.1}$$

The numerator has usually a smooth form of an analytic function e^{-sx^k}, or a polynomial form as the Wendland C^2 with high smoothness, as following

$$Q(x) = \left(1 - \frac{x}{r}\right)^4 \left(4\frac{x}{r} + 1\right) \tag{B.2}$$

After numerical experimentation in 1D and 2D problems, see in the next section the 1D parametric analysis, the role of the numerator has been found negligible when a denominator

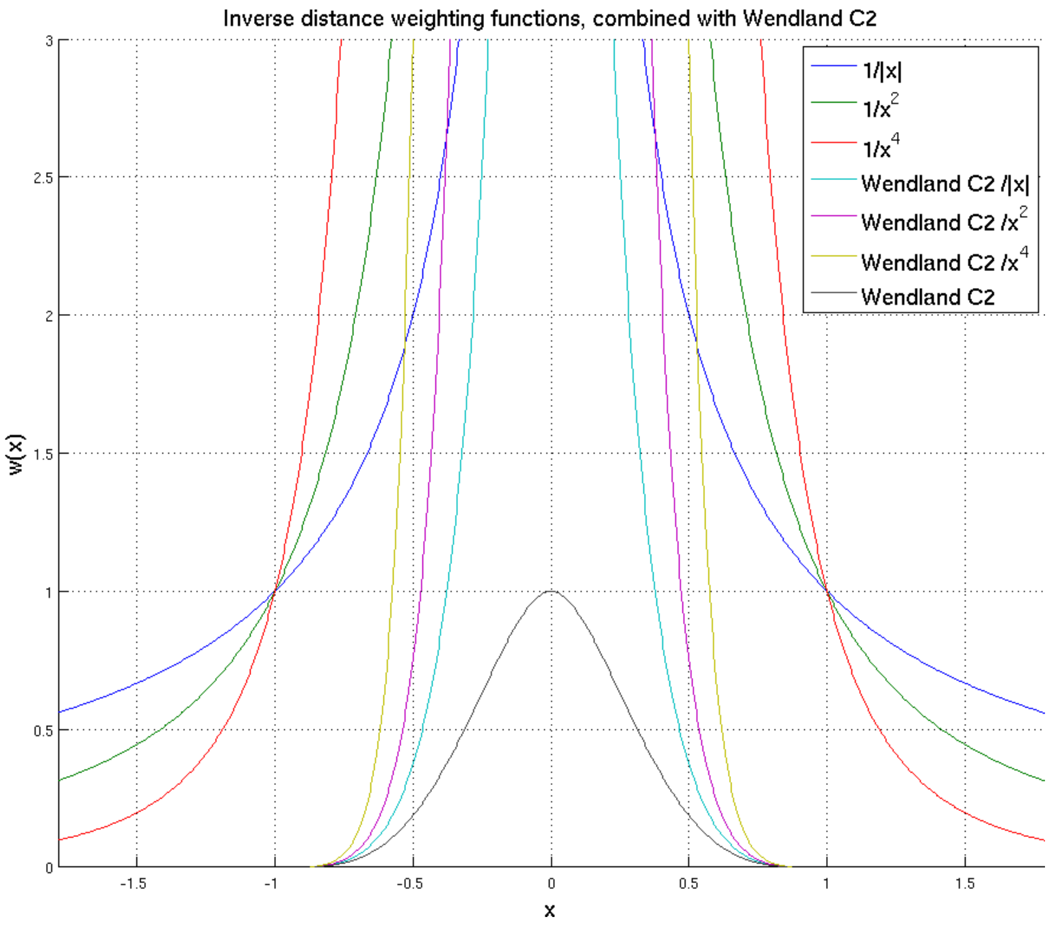


Figure B.1: Weighting functions used for interpolation, except the last one which is the C^2 Wendland, a non-interpolant generally. The first three are simple inverse distance functions, in comparison with the next three that are combined with the smoothening Wendland C^2 function.

exists in the following form. The denominator has a very important role in the interpolation ability of the local approximation, in the way that it has been explained in section 2.4. Thus, it is usually a distance function, most commonly a simple monomial of the distance x^a, which transform the weighting function into a singular one on the center point, by resulting very large values and very large decay speeds around the center. To dismiss the problem of the undefined value on the center, an infinitesimal value ϵ is introduced. Specifically, we will focus to the following class of weighting functions

$$w(x) = \frac{Q(x)}{\left(\frac{x}{r}\right)^a + \epsilon} \tag{B.3}$$

where r is the scaling parameter of the function's support, more precisely it is the desired size of the half-support, and $a \in \mathbb{R}^+$ is an interpolant parameter, i.e. certain values of the parameter, according to the difficulty of the problem, will constitute the local approximation an interpolant. For $a = 0$, one gets classical radial kernels, which are non interpolant for largely variable values of closely spaced sample points, mainly because they lack of sufficient decaying speed. In Figure B.1, weighting functions for various values of the interpolant parameter a are shown.

The infinitesimal value ϵ in the denominator should be assigned as small as possible, adopted to the machine numerical accuracy. The value $\epsilon = 1e - 12$ is sufficient for operations with single or double precision floating numbers.

B.2 A parametric analysis on the parameters of the rational weighting functions

In this section we solve a 1D problem with the MLS method, and variate the involved parameters of the weighting function, to demonstrate and analyse the effect on the quality of the global approximation. The same results can be extended to 2D and 3D problems, assisting a better selection of the parameter values. This 1D benchmark problem is set to contain small and large variations on the values of closely located sample points, moreover it possesses a tough region (on the right side of the domain) which is difficult to interpolate smoothly. First, for the 0-degree polynomial, we measure the interpolation quality of the solutions for various admissible parameter values. Next, we setup a parametric analysis with several combinations of the parameter values, on various polynomial degrees. In the latter case, we plot the resutls of the parametric analysis in 3D surfaces for a better and direct overview of the resutls.

The global approximation is desired to be interpolant, thus the quality metric of the approximation is the total relative error on the sample points

$$\|e_{rel}\|_2 = \left[\sum_{i=0}^{M} \left(\frac{\hat{f}(\hat{x}) - f(\hat{x})}{f(\hat{x})} \right)^2 \right]^{0.5} \tag{B.4}$$

The MLS solutions are utilizing the inverse distance weighting functions, with the Wendland C^2 function on the nominator for smoothener

$$w(x) = \frac{\left(1 - \frac{x}{r}\right)^4 \left(4\frac{x}{r} + 1\right)}{\left(\frac{x}{r}\right)^a + \epsilon} \tag{B.5}$$

It is remarkable that the solutions without the Wendland smoothener, i.e. the nominator is set to the constant value 1, are quite similar yielding similar error estimates.

Mainly, we examine the effect of the parameters a and r, as defined above, for various degrees of the polynomials. Before starting with the main findings, we demonstrate the sensitivity of the solution to the singularity correction factor ϵ, on the 0-degree approximation, with interpolation parameter $a = 4$, and $r = 0.05$. We consider the following set of values

$$\{\epsilon\} = \{1e-6,\ 1e-8,\ 1e-10,\ 1e-12,\ 1e-16\}$$

and we get the corresponding relative errors

$$\{\|e_{rel}\|_2\} = \{0.2545,\ 0.1514,\ 0.2545,\ 0.1514,\ 0.0042,\ 9.568e-5,\ 7.9723e-5\}$$

The plots of the solutions can be found in Figure B.2. We remark that generally the sensitivity is small, except the difficult region on the right side of the domain. Though, values above $1e-12$ are performing sufficiently good.

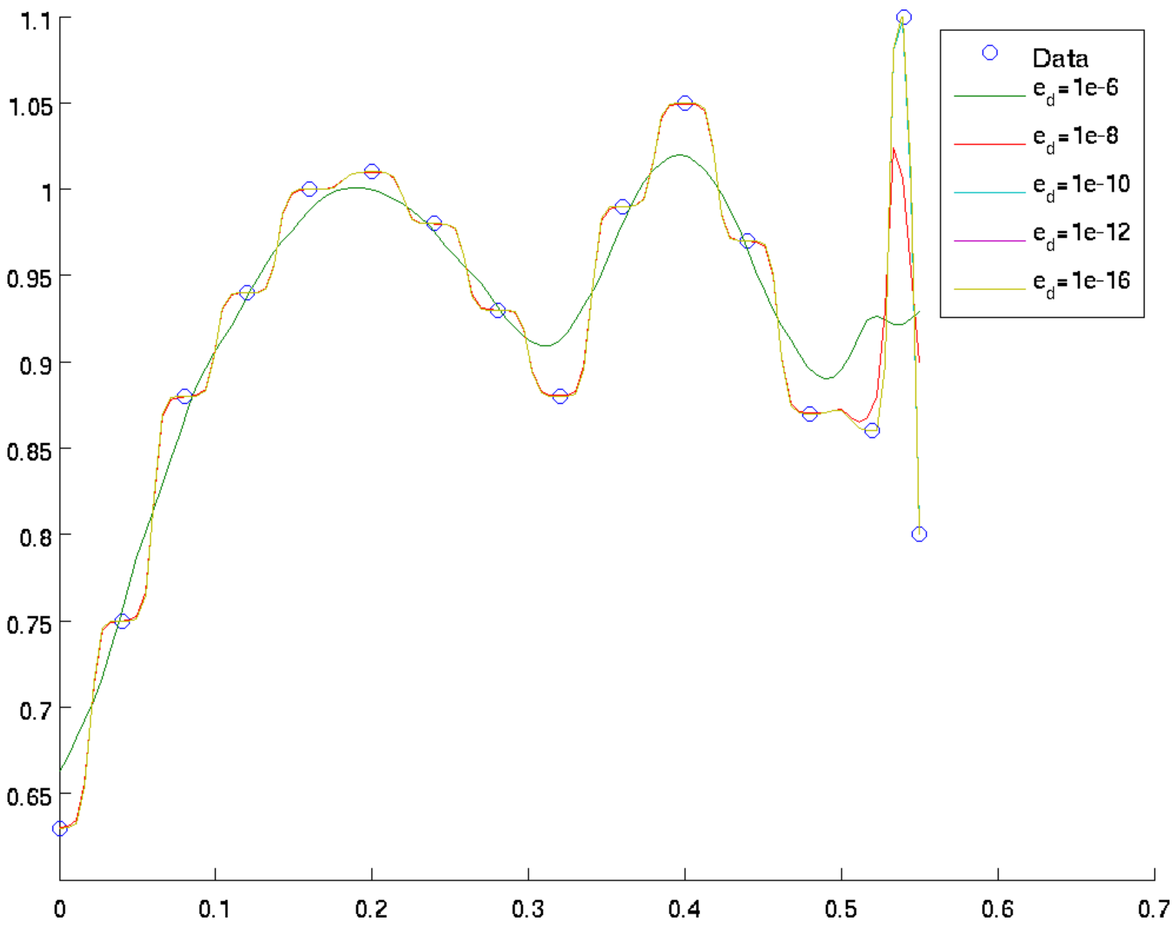


Figure B.2: Solution plots for variant singularity correction values ϵ. Polynomial degree $n = 0$. Weghting function parameters $a = 4$ and $r = 0.05$. Total relative interpolation error for each solution: $\|e_{rel}\|_2 = \{0.2545, 0.1514, 0.2545, 0.1514, 0.0042, 9.5680e-05, 7.9723e-05\}$

The effect of the interpolation parameter (a)
First, we investigate the interpolation parameter a with the set of values

$$\{a\} = \{1,\ 2,\ 4,\ 6,\ 8\}$$

The degree of the polynomial is always $n = 0$, and the support parameter $r = 1$. The analysis gives the corresponding relative errors

$$\{\|e_{rel}\|_2\} = \{0.0645,\ 0.0055,\ 9.5680e-05,\ 0.1455,\ 0.2511\}$$

The plots can be found in Figure B.3. We note that the interpolation parameter a has a global effect on each sample point approximation, no matter of how difficult are the data. The parameter value $a = 4$, seems to be an optimal one, achieving sufficiently good results. Testing the same interpolation parameter values, with support parameter $r = 0.05$, the results are improved as shown in Figure B.4 and assessed by the relative error metrics

$$\{\|e_{rel}\|_2\} = \{4.1028e-02,\ 5.0504e-03,\ 7.9290e-05,\ 1.3483e-06,\ 1.5700e-07\}$$

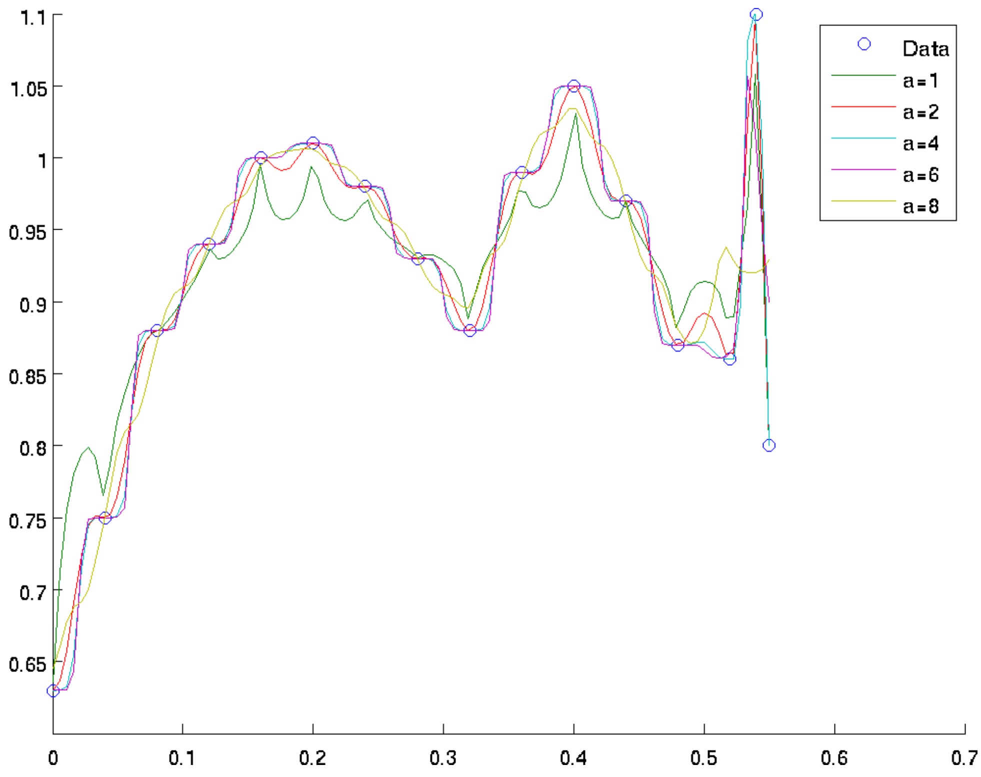


Figure B.3: Solution plots for variant interpolation parameter values a. Polynomial degree $n = 0$. Weghting function parameter $r = 1$. Total relative interpolation error for each solution: $\|e_{rel}\|_2 = \{0.0645, 0.0055, 9.5680e-05, 0.1455, 0.2511\}$

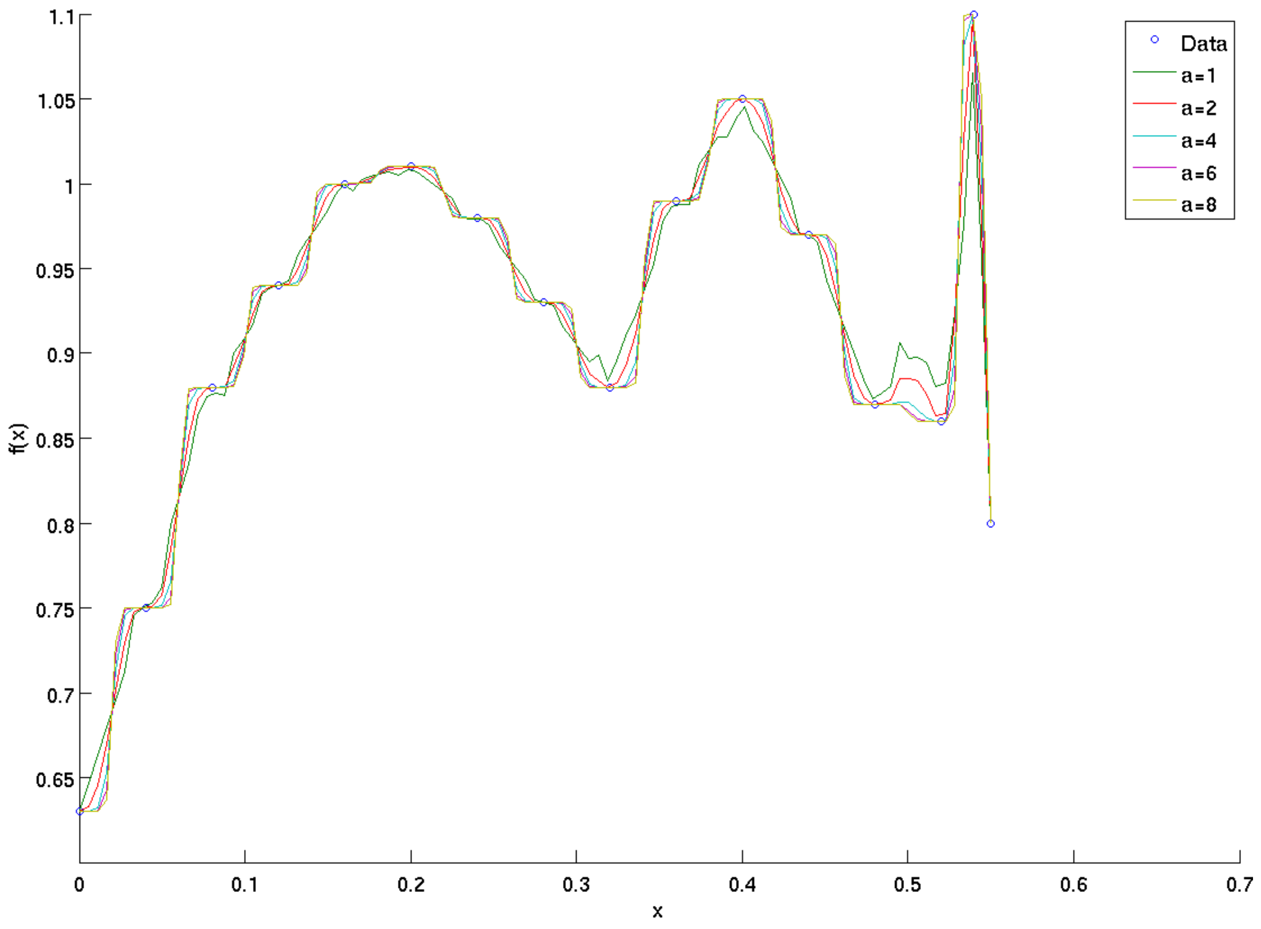


Figure B.4: Solution plots for variant interpolation parameter values a. Polynomial degree $n = 0$. Weghting function parameter $r = 0.05$. Total relative interpolation error for each solution: $\|e_{rel}\|_2 = \{4.1028e-02,\ 5.0504e-03,\ 7.9290e-05,\ 1.3483e-06,\ 1.5700e-07\}$

The effect of the support parameter (r)
Next, we investigate the support parameter r, which parameter should be always related to the spacing interval of the sample points, and it should not be interpreted as a general value for all the problems. In this problem the spacing interval is constant everywhere $d\hat{x} = 0.04$, except the right end of the domain where it is min $d\hat{x} = 0.01$. For this problem we choose the following set of values, representing explicitly the size of the half-support

$$\{r\} = \{0.02,\ 0.05,\ 0.1,\ 0.5,\ 1,\ 5,\ 50,\ 500\}$$

The polynomial degree is $n = 0$, and the interpolation parameter $a = 4$. Then we have the corresponding relative errors

$$\{\|e_{rel}\|_2\} = \{2e-05,\ 7e-05,\ 7e-05,\ 7e-05,\ 8e-05,\ 9e-05,\ 2e-02,\ 2e-01,\ 5e-01\}$$

The plots can be found in Figure B.5. All the solutions for $r \leq 1$ are sufficiently accurate. Note that there is an interaction between the interpolation parameter (a) and the span parameter (r), as shown previously between the Figures B.3 and B.4, i.e. different values of the parameter (a) will introduce a higher sensitivity on the (r) parameter and poor results. Such cases can be seen in the following more general parametric analysis.

It is better to keep proper interpolation parameter values, providing slow or zero variance with the values of (r), which can lead to simpler and general rules of support value (r) selection. Moreover, large sensitivity in the parameter (r) will lead to selection of small values, where one has to be careful not to exceed a minimum threshold that can constitute the system of equations ill-conditioned, e.g. if the number of effective sample points within the support is not at least the same as the degree of the polynomial.

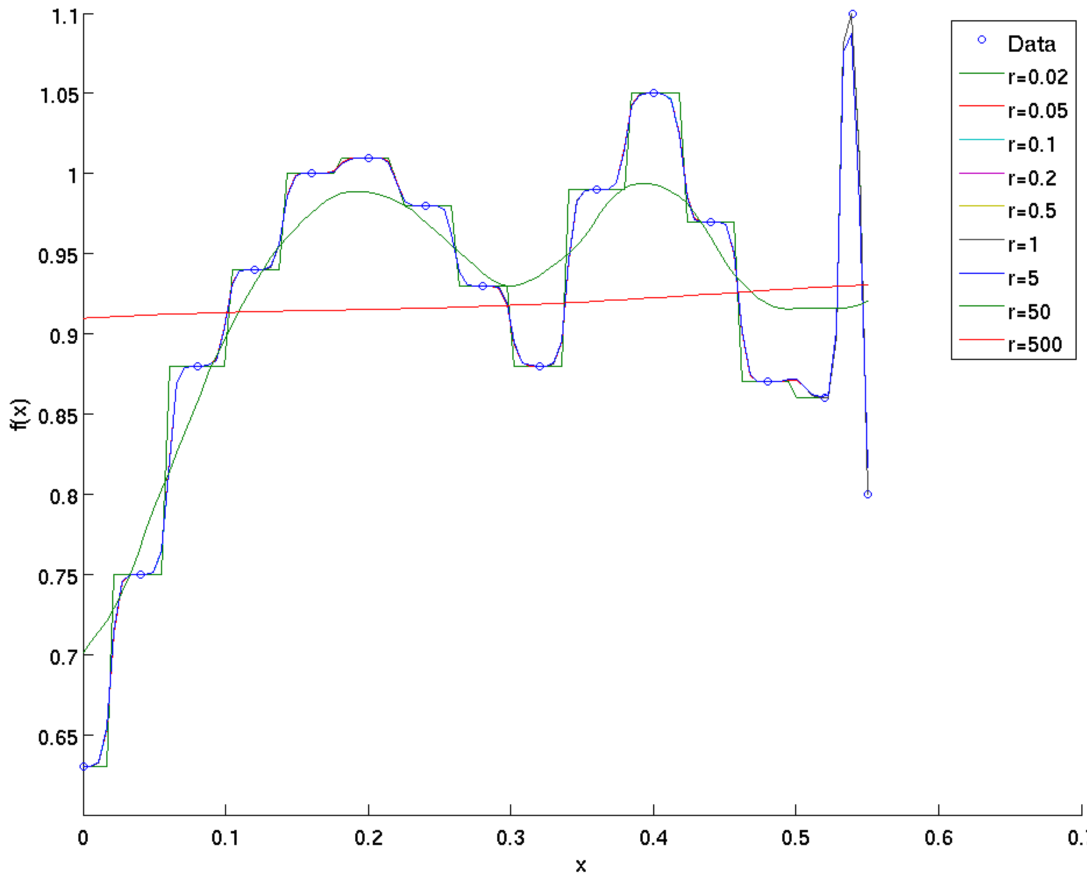


Figure B.5: Solution plots for variant span parameter values r. Polynomial degree $n = 0$. Weghting function parameter $a = 4$. Total relative interpolation error for each solution: $\|e_{rel}\|_2 = \{2.0300e - 05,\ 7.9290e - 05,\ 7.9609e - 05,\ 7.9727e - 05,\ 8.0148e - 05,\ 9.5680e - 05,\ 2.3941e - 02,\ 2.8402e - 01,\ 5.9267e - 01\}$

A parametric analysis for various polynomial degrees

For greater generality, we variate the weighting function parameters (a) and (r) for the polynomial degrees from 0 to 4. As above, we compute the total relative error of the interpolation for the various cases, and then plot 3D surfaces where in the z-axis are the error values, in x-y we choose two parameters of the (a), (r) and (n) (the degree of polynomial). All the results can be summirized in 3 surfaces plots.

First, we choose in x-y plane the interpolation parameter (a) and the support parameter (r). The interolation errors are plotted in 4 surfaces, one per polynomial degree, see Figure B.6. We observe that for all the polynomial degrees, the optimal range of the parameter (a) is between the values 2 and 4, constituting the solutions insensitive to the support parameter(r). Generally, all the polynomial degrees follow the same trends, and the differences in the interpolation error are becoming more sensible for large support values (r). As expected, low values of the interpolation paremeter (a), or no interpolation for $a = 0$, do not bring good interpolation results for any size of the support. Futhermore, for small enough support parameter (r), i.e. in the order of the interval distance between the sample points, the total interpolation error becomes sufficiently low and the solutions are almost insensitive to the interpolation parameter $a \geq 2$.

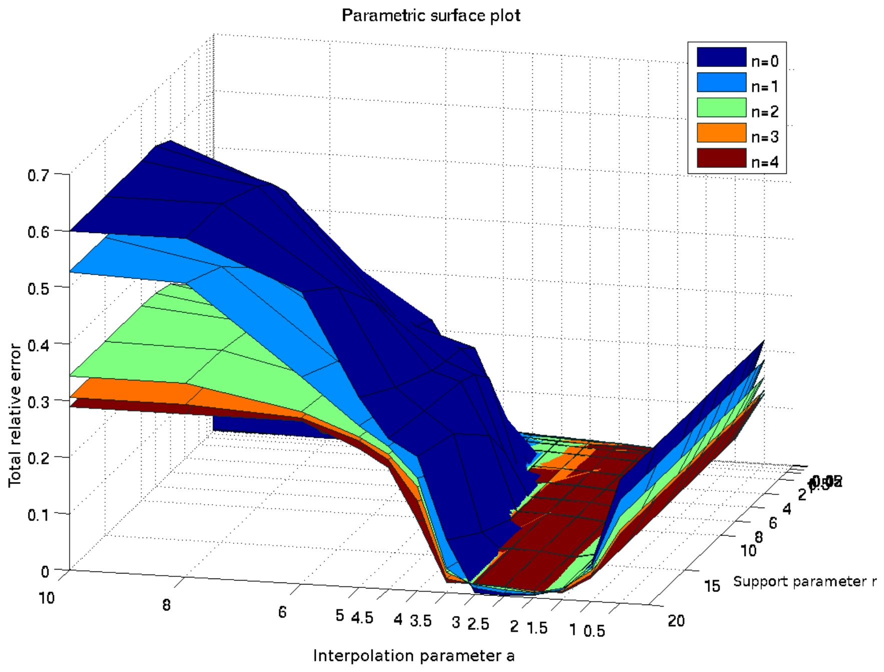


Figure B.6: The total relative error of interpolation, with variation of the interpolation parameter (a) and the support parameter (r). Each surface is representing the error values in z-axis, for a specific polynomial degree. Shown are the polynomial degrees from 0 to 4.

Next, we choose in x-y plane the support parameter (r) and the polynomial degree (n). The interolation errors are plotted in 13 surfaces, one per interpolation parameter value

$$\{a\} = \{0.5,\ 1,\ 1.5,\ 2,\ 2.5,\ 3,\ 3.5,\ 4,\ 4.5,\ 5,\ 6,\ 8,\ 10\}$$

The results are shown in Figure B.7, where we see that interpolation values around $a = 3$, give sufficiently good results for all the tested polynomials and support parameter values. Values of $a \geq 4$ are quite unstable in relation with the support parameter (r) and should be avoided. Values of $a \leq 2$ give a stable high interpolation error.

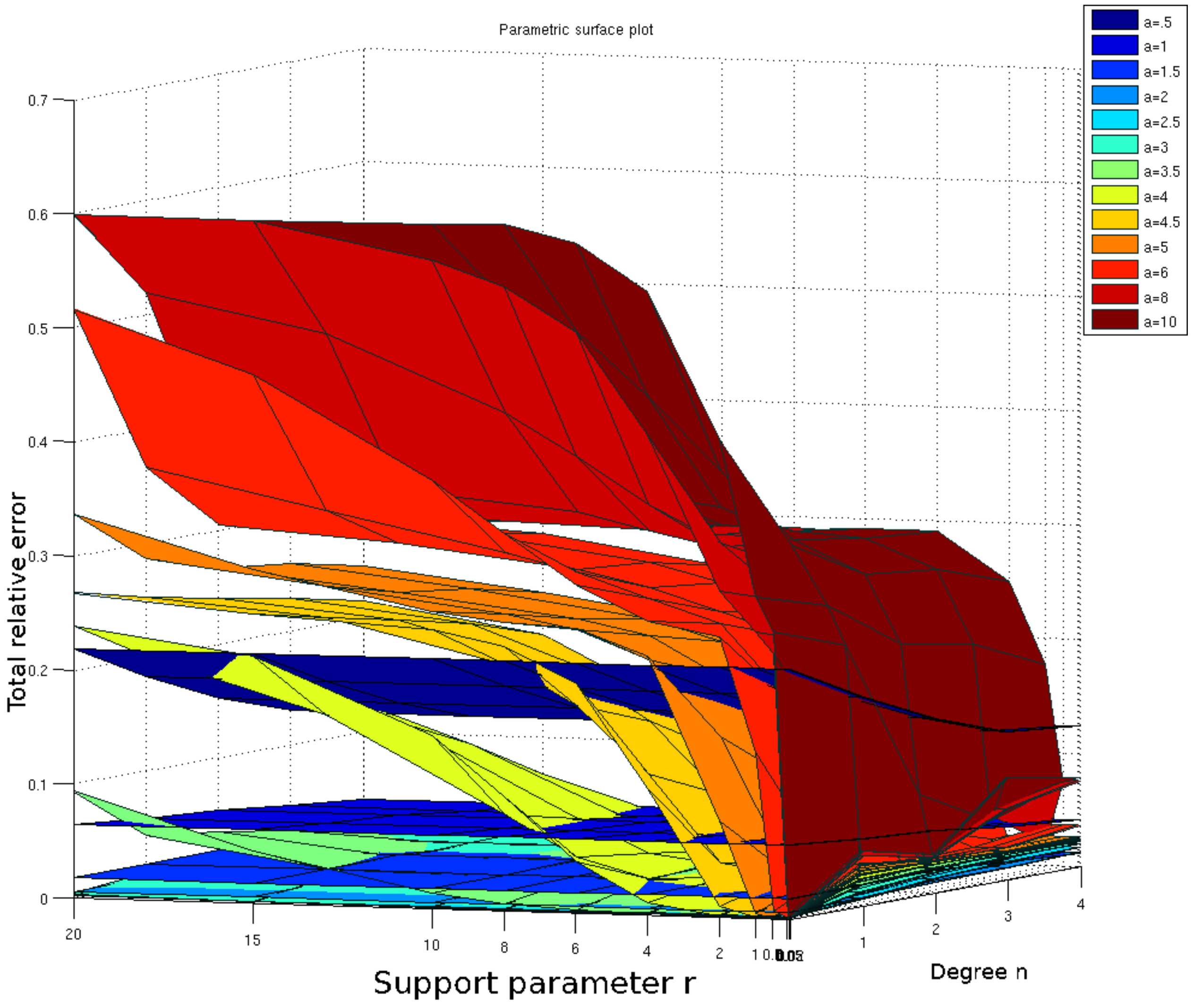


Figure B.7: The total relative error of interpolation, with variation of the the support parameter (r) and the polynomial degree (n). Each surface is representing the error values in z-axis, for a specific interpolation parameter (a). Shown are the interpolation values $\{0.5,\ 1,\ 1.5,\ 2,\ 2.5,\ 3,\ 3.5,\ 4,\ 4.5,\ 5,\ 6,\ 8,\ 10\}$.

At last, we choose in x-y plane the interpolation parameter (a) and the polynomial degree (n). The interolation errors are plotted in 12 surfaces, one per support parameter value

$$\{r\} = \{0.02,\ 0.05,\ 0.1,\ 0.5,\ 1,\ 2,\ 4,\ 6,\ 8,\ 10,\ 15,\ 20\}$$

Note that the degree of the polynomial affects the condition of the system of equations in combination with the support parameter. For instance the 4^{th} and 3^{rd} degree interpolation, provides well behaved solutions for $r \geq 0.2$, the 2^{nd} degree for $r \geq 0.1$, and the 1^{st} degree $r \geq 0.04$ (i.e. the maximum sample points interval $d\hat{x}$ in this benchmark problem). The results are shown in Figure B.8, where we see that the minimum possible parameter value (r), provides the lowest interpollation error among all the cases. Finally, we observe again that the interpolation parameter (a) in range $2 \leq a \leq 4$, gives an extra interpolation boost and invariance to the degree n and the support r.

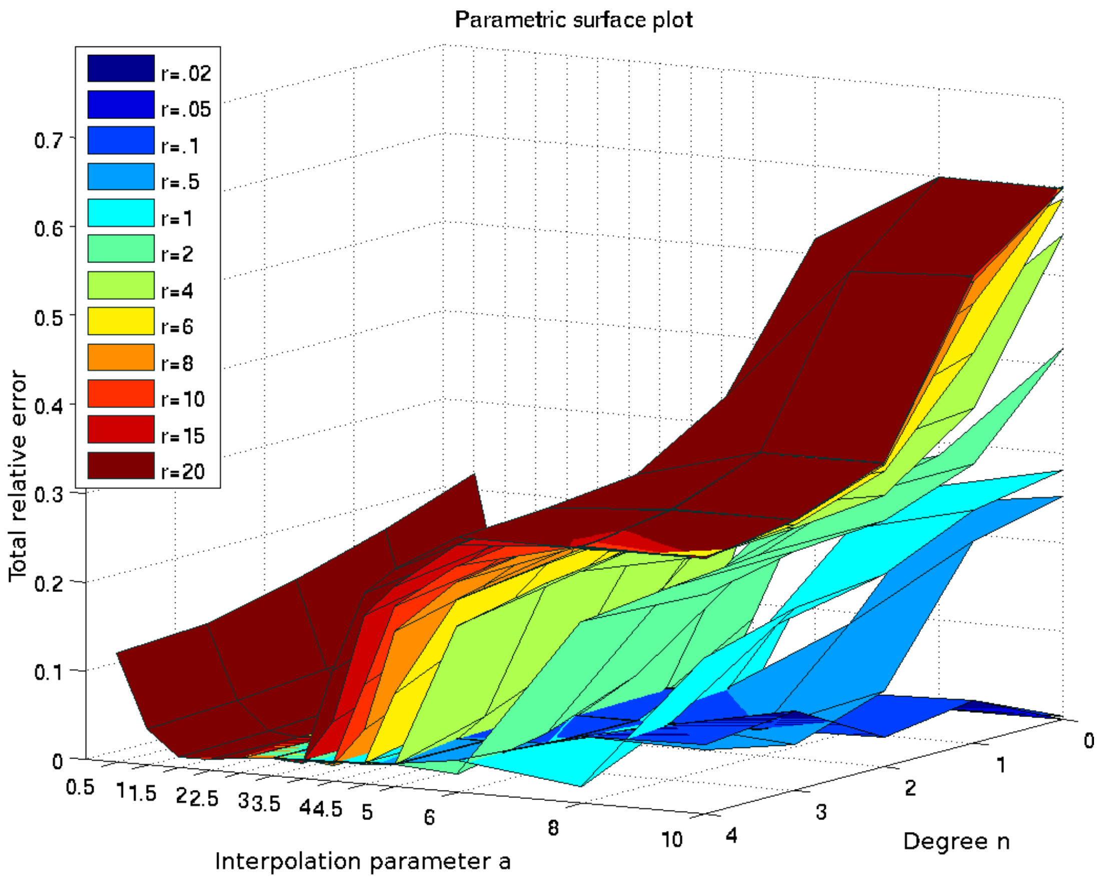


Figure B.8: The total relative error of interpolation, with variation of the interpolation parameter (a) and the polynomial degree (n). Each surface is representing the error values in z-axis, for a specific support parameter (r). Shown are the the support values $\{0.02,\ 0.05,\ 0.1,\ 0.5,\ 1,\ 2,\ 4,\ 6,\ 8,\ 10,\ 15,\ 20\}$.

Appendix C

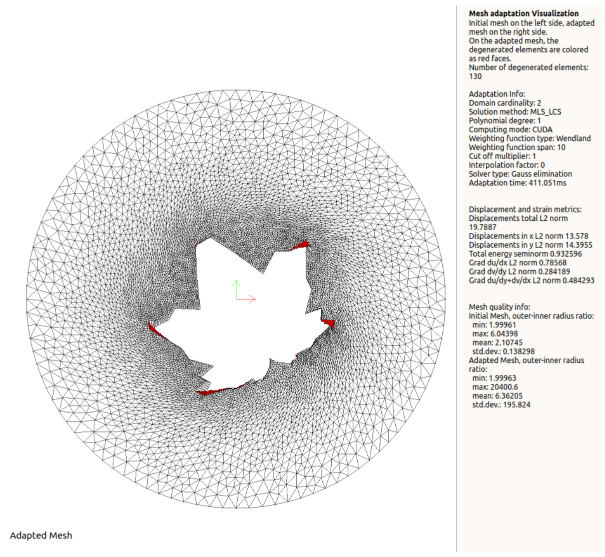
The performance of the MLS method in mesh deformation problems

C.1 Introduction

The idea of this method is to determine the displacements of the interior mesh nodes for which just a limited number of displacements are known on the boundaries. If a certain model function describing the system of nodes is presumed, then the parameters of this function are searched which explain the system best in terms of minimisation of the quadratic residuals. Utilizing the MLS method this is done locally, launching a local least squares approximation for each interior node, one by one. Then, the solution for the constant parameter a_0, determines explicitly the movement of each interior node, see sections 2.6 and 2.4. According to the developed theory in chapter 1 on the MLS method, all the nodes with prescribed displacements (including the fixed nodes with 0 displacement) constitute the sample points $\hat{x}$ of the problem. The rest of the nodes for which the new position is unknown, they are the stationary points $\bar{x}$.

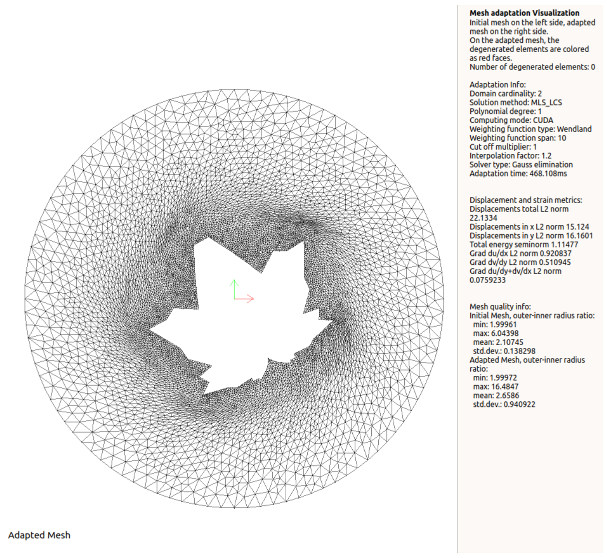
The problem of adapting a mesh to displacements on the boundary, can be described as well as reconstructed displacement fields with given sample data on the boundaries. In this approach, the fields are independent per displacement component (u_x, u_y, u_z), such that the MLS method is applied on each interior node independently as many times as the number of the domain dimensions, e.g. 2 times for a 2D mesh and 3 times for a 3D mesh. If for any reason we have no displacements on the boundaries in any of the directions, the method is useless to apply in that specific domain component, as there is no correlation between the displacement fields.

The displacements of the boundaries are prescribed explicitly as it is essential for the solved problem, though the displacements on the interior nodes very close to the boundary are determined with the MLS method, which should follow smoothly close to the prescribed displacements on the boundary, in order to preserve the quality of the mesh on the boundary. Hence, we would like the MLS approximation to yield smooth displacement fields which will interpolate the displacements on the boundaries. A special care should be attained for cases where we have large variations of displacements on closely spaced boundary nodes. As demonstrated with several notions and 1D examples in sections 2.4 and B, such large



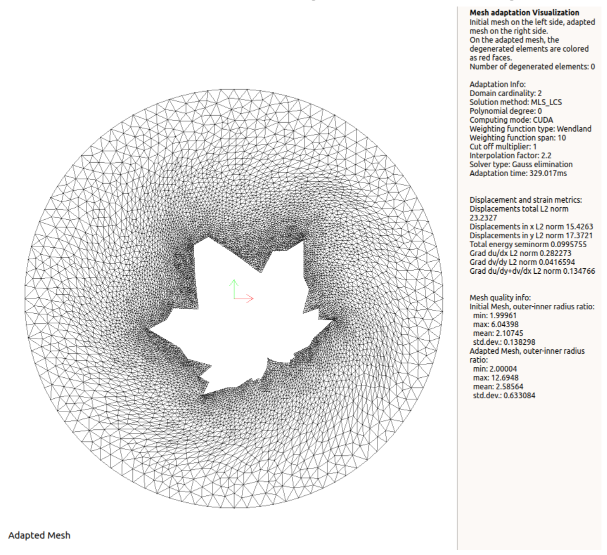
```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements:
130

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 0
Solver type: Gauss elimination
Adaptation time: 411.051ms


Displacement and strain metrics:
Displacements total L2 norm
19.7887
Displacements in x L2 norm 13.578
Displacements in y L2 norm 14.3955
Total energy seminorm 0.932596
Grad du/dx L2 norm 0.78568
Grad dv/dy L2 norm 0.284189
Grad du/dy+dv/dx L2 norm 0.484293


Mesh quality info:
Initial Mesh, outer-inner radius ratio:
  min: 1.99961
  max: 6.04398
  mean: 2.10745
  std.dev.: 0.138298
Adapted Mesh, outer-inner radius
ratio:
  min: 1.99963
  max: 20400.6
  mean: 6.36205
  std.dev.: 195.824
```

(a) Interpolation parameter $a = 0$, polynomial degree 1. Resulted 130 degenerated elements on the boundary.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 1.2
Solver type: Gauss elimination
Adaptation time: 468.108ms


Displacement and strain metrics:
Displacements total L2 norm
22.1334
Displacements in x L2 norm 15.124
Displacements in y L2 norm 16.1601
Total energy seminorm 1.11477
Grad du/dx L2 norm 0.920837
Grad dv/dy L2 norm 0.510945
Grad du/dy+dv/dx L2 norm
0.0759233


Mesh quality info:
Initial Mesh, outer-inner radius ratio:
  min: 1.99961
  max: 6.04398
  mean: 2.10745
  std.dev.: 0.138298
Adapted Mesh, outer-inner radius
ratio:
  min: 1.99972
  max: 16.4847
  mean: 2.6586
  std.dev.: 0.940922
```

(b) Interpolation parameter $a = 1.2$, polynomial degree 1. Resulted good mesh quality.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 2.2
Solver type: Gauss elimination
Adaptation time: 329.017ms


Displacement and strain metrics:
Displacements total L2 norm
23.2327
Displacements in x L2 norm 15.4263
Displacements in y L2 norm 17.3721
Total energy seminorm 0.0995755
Grad du/dx L2 norm 0.282273
Grad dv/dy L2 norm 0.0416594
Grad du/dy+dv/dx L2 norm 0.134766


Mesh quality info:
Initial Mesh, outer-inner radius ratio:
  min: 1.99961
  max: 6.04398
  mean: 2.10745
  std.dev.: 0.138298
Adapted Mesh, outer-inner radius
ratio:
  min: 2.00004
  max: 12.6948
  mean: 2.58564
  std.dev.: 0.633084
```

(c) Interpolation parameter $a = 2.2$, polynomial degree 0. Resulted good mesh quality.

Figure C.1: The figure illustrates the failing result of a mesh adaptation, caused by applying a non-interpolant MLS method and misinterpolating the boundary displacements. The example is for the 1^{st} degree MLS solutions with a) interpolation parameter $a = 0$, b) $a = 1.2$, c) 0-degree MLS $a = 2.2$. Similar results are produced with higher degrees and large variations on the boundary.

variations can introduce large interpolation errors, if the correct parameters are not chosen for the weighting function. The latter case, is shown in Figure C.1 for a 2D mesh adaptation example solved with the 1^{st} degree MLS method, and a Wenland $C2$ inverse distance weighting function. Moreover, it has been shown that decreasing the support of the weighting function can boost the local character of the MLS over an evaluation point, with the cost that the variations of the displacements are becoming much more rapid decaying. The rapid decaying displacements around the boundaries can be a problem of destroying the quality of the mesh in that close region, letting the further points non-displaced.

The 0-degree MLS method with proper inverse distance weighting functions, also well known as the Shepard's method (see Shepard [1968]) that was developed for surface reconstruction problems, has been proved extremely efficient to solve the problem of mesh adaption to boundary displacements, overcoming the above mentioned difficulties. Though, the magnitude of the boundary displacements in difficult cases (as the benchmark problem in the following sections) is still limited. Moreover, the method is by some orders computationally cheaper as we are dealing only with the constant polynomial parameter. Higher polynomial degree MLS adaptations are giving improved interpolated results, in the same way of implying an inverse distance weighting function.

In this chapter a performance analysis of the method is developed, to demonstrate its effectiveness in mesh adaptation problems. First, we start with the consistency analysis of the method, to confirm that it can represent some primative deformation modes, i.e. the affine transformations. It is required that, given some simple boundary displacements, which correspond to an affine transformation, the displacement patterns in the interior should be the expected ones. These results are considered important, because any other imposed complex displacements on the boundaries will be a synthesis of the fundamental, expected to behave correctly as well. This concept is very common in finite elements methods, where it is proved that the solution basis can represent correctly some primitive fields.

Next, the performance of the method on 2D mesh adaptation problems is analysed. Feasibility and quality results are presented, after a vast number of cases on the parameters of the general weighting function, see B. These analyses are oriented to reveal the role of each parameter, the feasible values, and the optimal ranges for various polynomial degrees. The feasibility is defined by adaptations without degenerated elements, and the optimality of the parameters is linked with the mesh quality metrics.

C.2 Consistency tests of the IDW method on mesh deformation applications

In this section the MLS method applied on the described mesh adaptation problem, is tested in reliability to represent some expected fundamental displacement fields in the interior for the corresponding displacements on the boundaries. The 1^st degree MLS is behaving in an excellent way representing exactly all the fundamental modes as expected. The 0-degree MLS is shown to behave sufficiently good also. For cases where the 0-degree MLS locks rigidly some regions of the mesh, the weighting function parameters can relax them yielding a sufficiently good expected behavior.

The tested fundamental displacement fields are all affine transformations, i.e. translations, rotations and stretches. For the latter type of displacements we know a priori the expected displacement and gradient fields when the test domain is a simple one. In such a case the errors in the fields can be measured straightforward.

Thus, all the consistency tests will be executed on a simple 2D square domain of dimension 9x9. We will use the L^2 norms on the involved fields (displacement or gradient) over the domain, to measure the magnitudes of the expected and approximated fields and define errors

$$\|u\| = \left[\int_{\Omega} u(x)^2 d\Omega\right]^{0.5} \tag{C.1}$$

To measure the approximated fields numerically we use the following discrete norm

$$\|u\| = \left[\sum_{i=1}^{n} u(x)^2 A_i\right]^{0.5} \tag{C.2}$$

where A_i is the area of a single element defined in the mesh.

Thus, to verify the results of every tested fundamental mode the following field measurements will be checked
The L^2 norm of the x-displacement field $\|u\|$,
The L^2 norm of the y-displacement field $\|v\|$,
The L^2 norm of the total displacements field $\|u_t\|$,
The L^2 norm of the x-displacements $grad_x$ field $\|du/dx\|$,
The L^2 norm of the y-displacements $grad_y$ field $\|dv/dy\|$,
The L^2 norm of the shear deformation field $\|du/dy + dv/dx\|$.

In the following plots, the displacement fields are based on the evaluations on the nodes with the standard MLS method results, i.e. the values of the a_0 coefficient. The gradient fields are derived from the gradient values on the nodes, which are estimated from the corresponding linear coefficients a_i of the MLS method. For the 0-degree MLS the gradient fields are produced from the gradient values on the nodes, estimated as defined in eq. (2.37).

The MLS method for both the polynomial degrees, is launched with the simple inverse distance weighting function

$$w(x) = \frac{1}{\left(\frac{x}{10}\right)^4 + 1e - 12}$$

with interpolation parameter $a = 4$ and support size parameter $r = 10$.

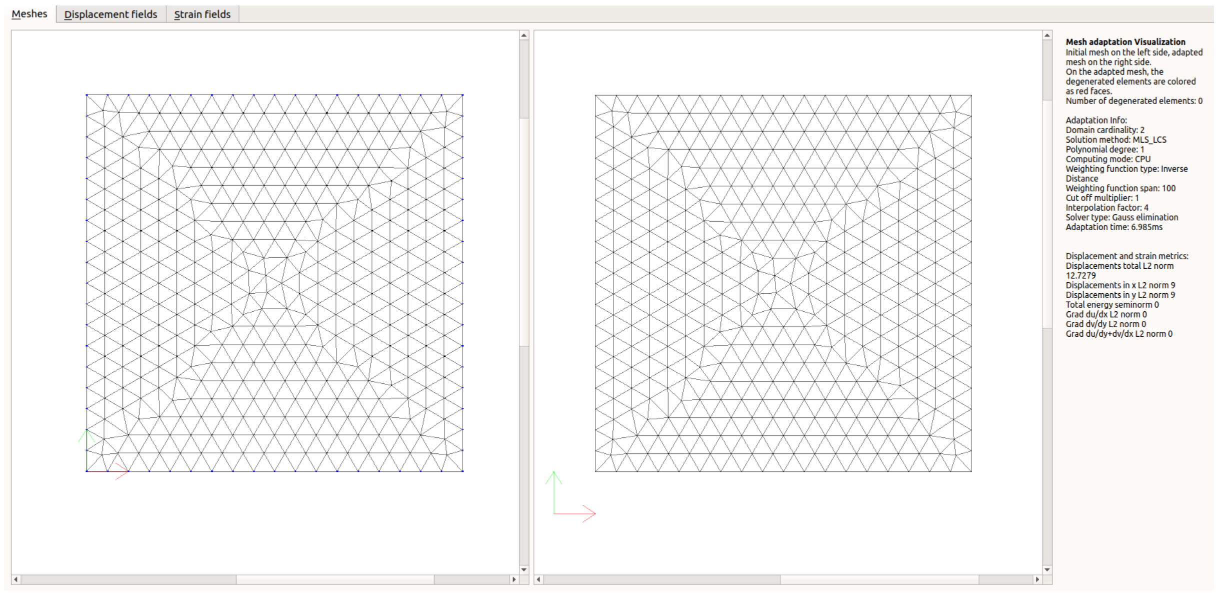
C.2.1 Consistency test on the rigid translation

First, the rigid translations is tested to verify the consistency of the displacement fields with the expected ones.

We translate the nodes of one (or more) of the boundaries, letting the rest nodes free to follow the move. The applied translation in this test is $u_x = 1$ and $u_y = 1$. We expect all the nodes to move by the same displacement with no stretches applied, see Figure C.2, and the following field measurements
$\|u\| = 9$, $\|v\| = 9$, $\|u_t\| = 12.7279$,
$\|du/dx\| = 0$, $\|dv/dy\| = 0$, $\|du/dy + dv/dx\| = 0$

The test is successful for both of the MLS launches (both polynomial degrees), as the displacement gradients are zero and all the nodes follow the expected movement. The field measurements are confirmed with zero error, as shown in the right info panel in Figures C.2 and C.3.

The displacement and gradient fields for the 0-degree MLS are plotted in the figures C.4 and C.5, indicating no defects.

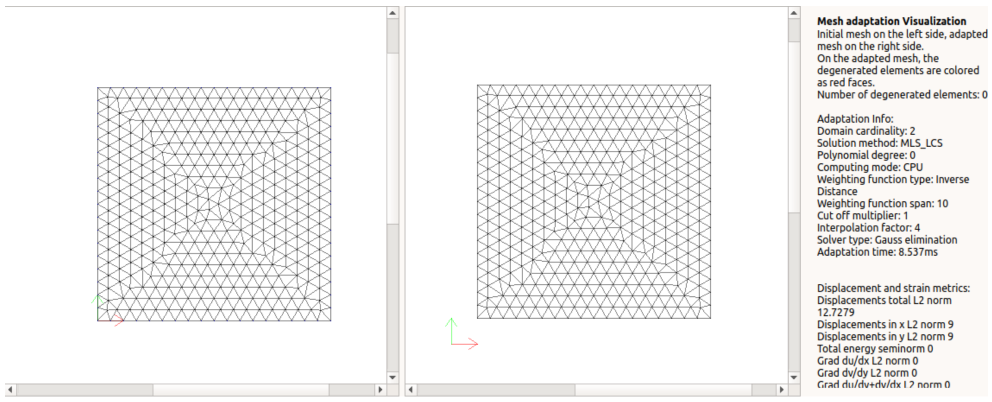


Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 100
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 6.985ms

Displacement and strain metrics:
Displacements total L2 norm 12.7279
Displacements in x L2 norm 9
Displacements in y L2 norm 9
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0

Figure C.2: The meshes before and after the rigid translation $u_x = 1$, $u_y = 1$ of the square domain with size 9x9. Adaptation with the 1^{st} degree MLS method. On the right panel, the settings of the MLS methods are shown, as long as the L^2 displacement and gradient fields measurements.

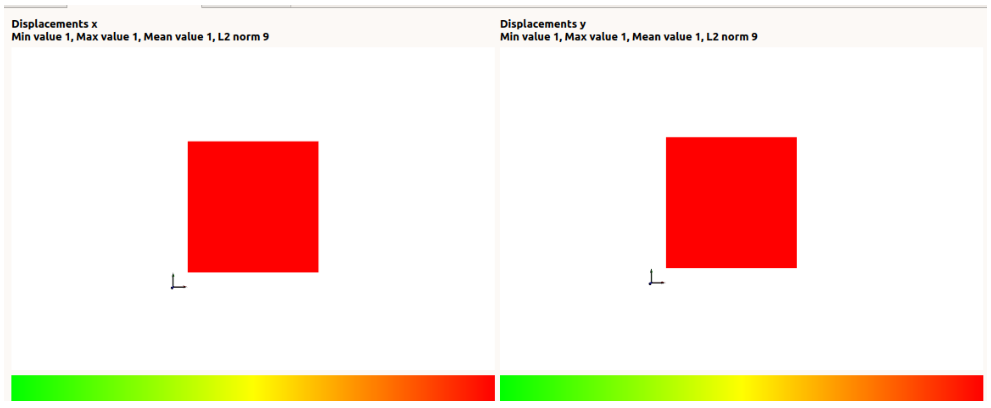


Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 8.537ms

Displacement and strain metrics:
Displacements total L2 norm 12.7279
Displacements in x L2 norm 9
Displacements in y L2 norm 9
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0

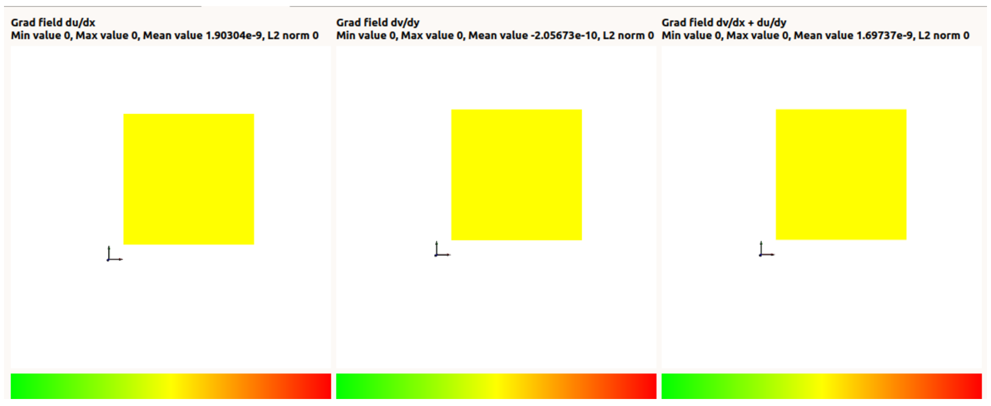
Figure C.3: The meshes before and after the rigid translation $u_x = 1$, $u_y = 1$ of the square domain with size 9x9. Adaptation with the 0-degree MLS method.



Displacements x
Min value 1, Max value 1, Mean value 1, L2 norm 9

Displacements y
Min value 1, Max value 1, Mean value 1, L2 norm 9

Figure C.4: The displacement fields after a rigid translation $u_x = 1$, $u_y = 1$ of the square domain with size 9x9. Adaptation with the 0-degree MLS method. On the top bar, the L^2 field measurements are shown.



Grad field du/dx
Min value 0, Max value 0, Mean value 1.90304e-9, L2 norm 0

Grad field dv/dy
Min value 0, Max value 0, Mean value -2.05673e-10, L2 norm 0

Grad field dv/dx + du/dy
Min value 0, Max value 0, Mean value 1.69737e-9, L2 norm 0

Figure C.5: The displacement gradient fields after a rigid translation $u_x = 1$, $u_y = 1$ of the square domain with size 9x9. Adaptation with the 0-degree MLS method. On the top bar, the L^2 field measurements are shown.

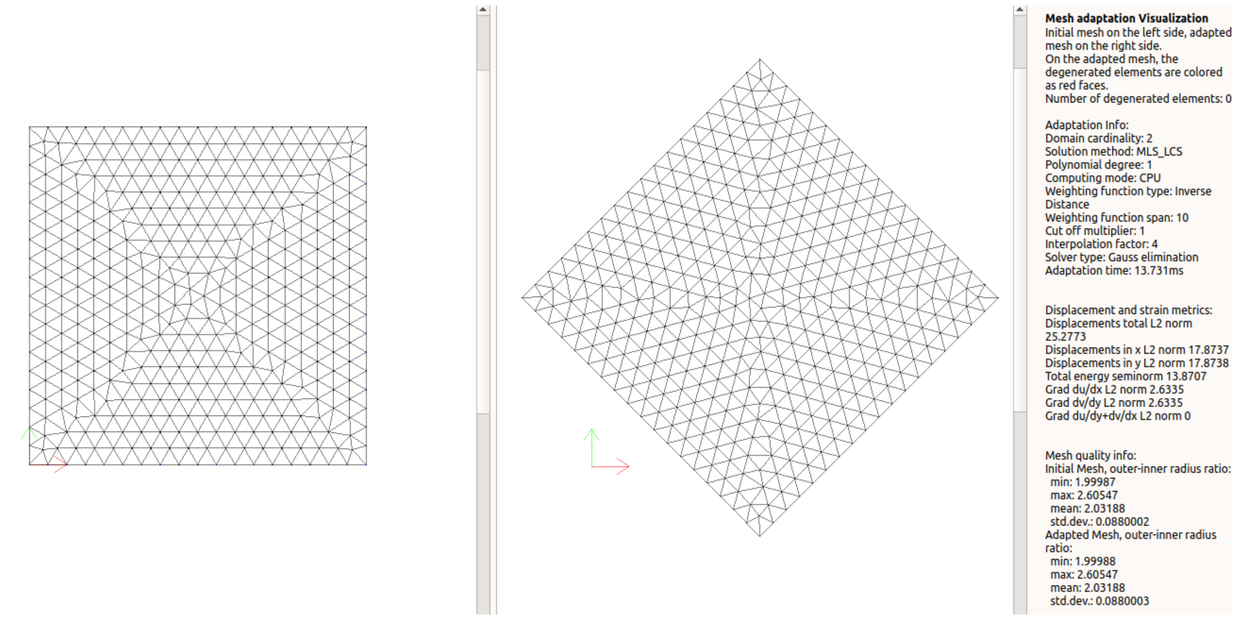
C.2.2 Consistency test on the rigid rotation

The rigid rotation is tested to verify the consistency of the displacement fields with the expected ones.

The tested mesh is rotated by applying a rotation around the center point of the square on the boundary nodes, letting the rest nodes free to follow the move. The applied rotation in this test is $\theta = 45deg$ around the node $x = 4.5, y = 4.5$. We expect all the nodes to rotate about the rotation point, with no stretches appearing, i.e. the local gradients of the displacement fields should evaluate constant as following $du/dx = 1 - \; cos\,(\theta) = 0.292$, $dv/dy = 1 - \; cos\,(\theta) = 0.292$, $du/dy + dv/dx = 0$
$\|du/dx\| = 2.63$, $\|dv/dy\| = 2.63$, $\|du/dy + dv/dx\| = 0$

The test is successful for the 1^{st} degree MLS, as the displacement gradients are constant and all the nodes follow the expected movement.

The 0-degree solution exhibits bad performance for a high interpolation parameter value $a = 4$, because of the zero gradient property on the sample points (property of the interpoland MLS method for 0-degree polynomial). The results are improved to a sufficient practical degree, where an arbitrary rotation can be achieved with the expected displacement field, if the interpolation parameter is reduced to $a = 2$. The gradient fields for the two MLS solutions are shown in the figures C.9 and C.10, indicating no defects for the 1^{st} degree and slight defects for the 0-degree on the boundaries.



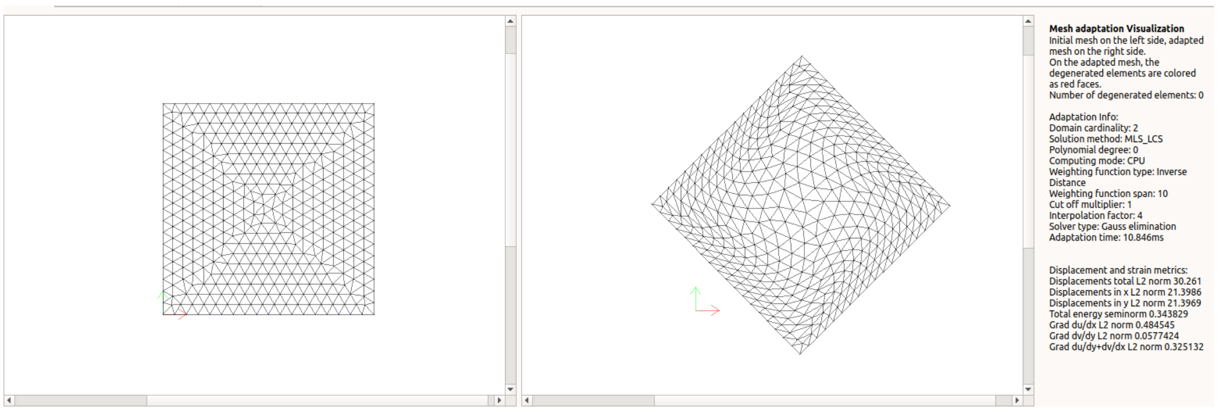
Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 13.731ms

Displacement and strain metrics:
Displacements total L2 norm 25.2773
Displacements in x L2 norm 17.8737
Displacements in y L2 norm 17.8738
Total energy seminorm 13.8707
Grad du/dx L2 norm 2.6335
Grad dv/dy L2 norm 2.6335
Grad du/dy+dv/dx L2 norm 0

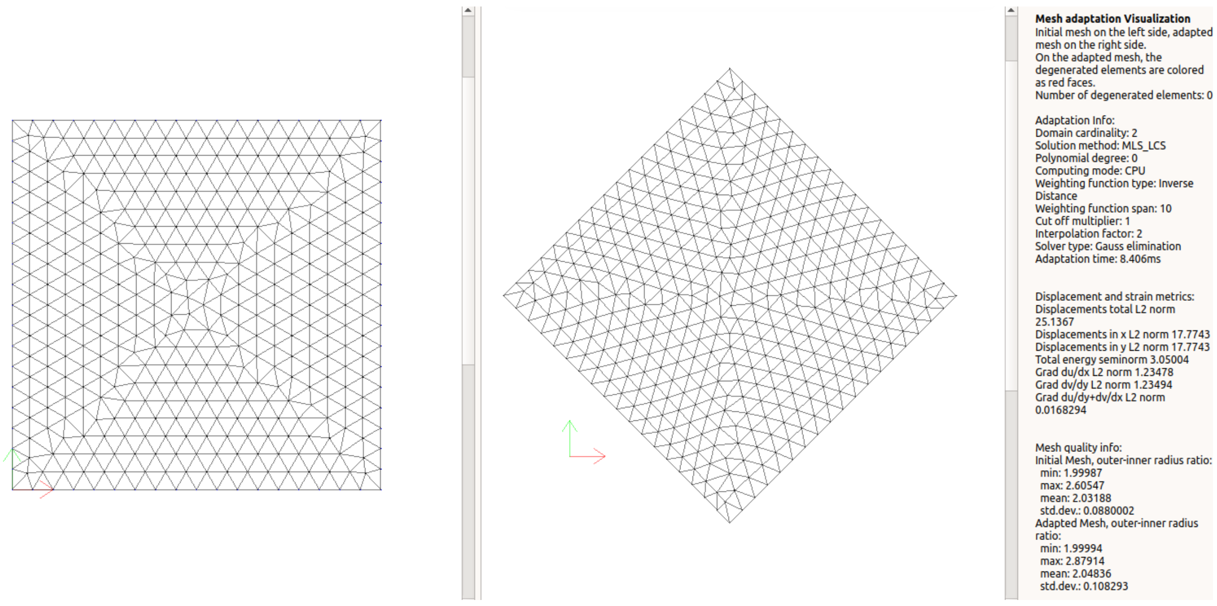
Mesh quality info:
Initial Mesh, outer-inner radius ratio:
  min: 1.99987
  max: 2.60547
  mean: 2.03188
  std.dev.: 0.0880002
Adapted Mesh, outer-inner radius ratio:
  min: 1.99988
  max: 2.60547
  mean: 2.03188
  std.dev.: 0.0880003

Figure C.6: The meshes before and after the rigid rotation of the square domain with size 9x9. Adaptation with the 1^{st} degree MLS method. On the right panel, the settings of the MLS methods are shown, as long as the L^2 displacement and gradient fields measurements.



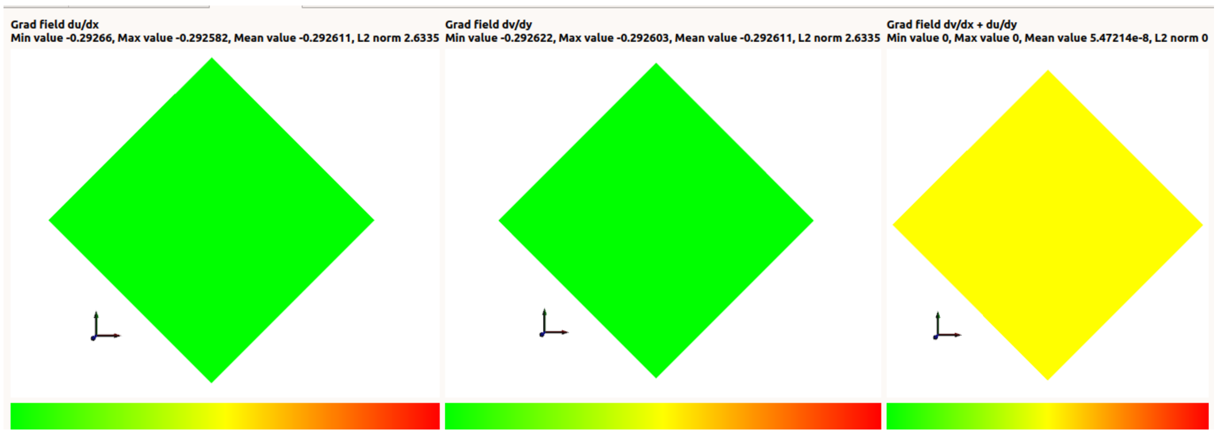
```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse
Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 10.846ms


Displacement and strain metrics:
Displacements total L2 norm 30.261
Displacements in x L2 norm 21.3986
Displacements in y L2 norm 21.3969
Total energy seminorm 0.343829
Grad du/dx L2 norm 0.484545
Grad dv/dy L2 norm 0.0577424
Grad du/dy+dv/dx L2 norm 0.325132
```

Figure C.7: Adaptation with the 0-degree MLS method.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse
Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 2
Solver type: Gauss elimination
Adaptation time: 8.406ms


Displacement and strain metrics:
Displacements total L2 norm
25.1367
Displacements in x L2 norm 17.7743
Displacements in y L2 norm 17.7743
Total energy seminorm 3.05004
Grad du/dx L2 norm 1.23478
Grad dv/dy L2 norm 1.23494
Grad du/dy+dv/dx L2 norm
0.0168294


Mesh quality info:
Initial Mesh, outer-inner radius ratio:
  min: 1.99987
  max: 2.60547
  mean: 2.03188
  std.dev.: 0.0880002
Adapted Mesh, outer-inner radius
ratio:
  min: 1.99994
  max: 2.87914
  mean: 2.04836
  std.dev.: 0.108293
```

Figure C.8: Adaptation with the 0-degree MLS method and an optimizated interpolation parameter a.



```
Grad field du/dx
Min value -0.29266, Max value -0.292582, Mean value -0.292611, L2 norm 2.6335

Grad field dv/dy
Min value -0.292622, Max value -0.292603, Mean value -0.292611, L2 norm 2.6335

Grad field dv/dx + du/dy
Min value 0, Max value 0, Mean value 5.47214e-8, L2 norm 0
```

Figure C.9: The displacement gradient fields after a rigid rotation of the square domain with size 9x9. Adaptation with the 1^{st} degree MLS method. On the top bar, the L^2 field measurements are shown.

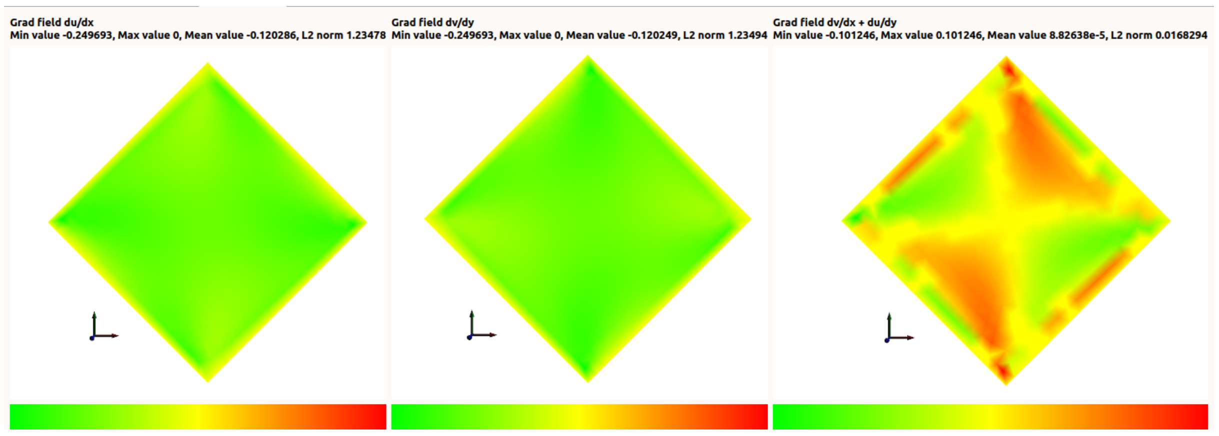


Figure C.10: The displacement gradient fields after a rigid rotation of the square domain with size 9x9. Adaptation with the 0-degree MLS method and an optimed interpolation parameter a.

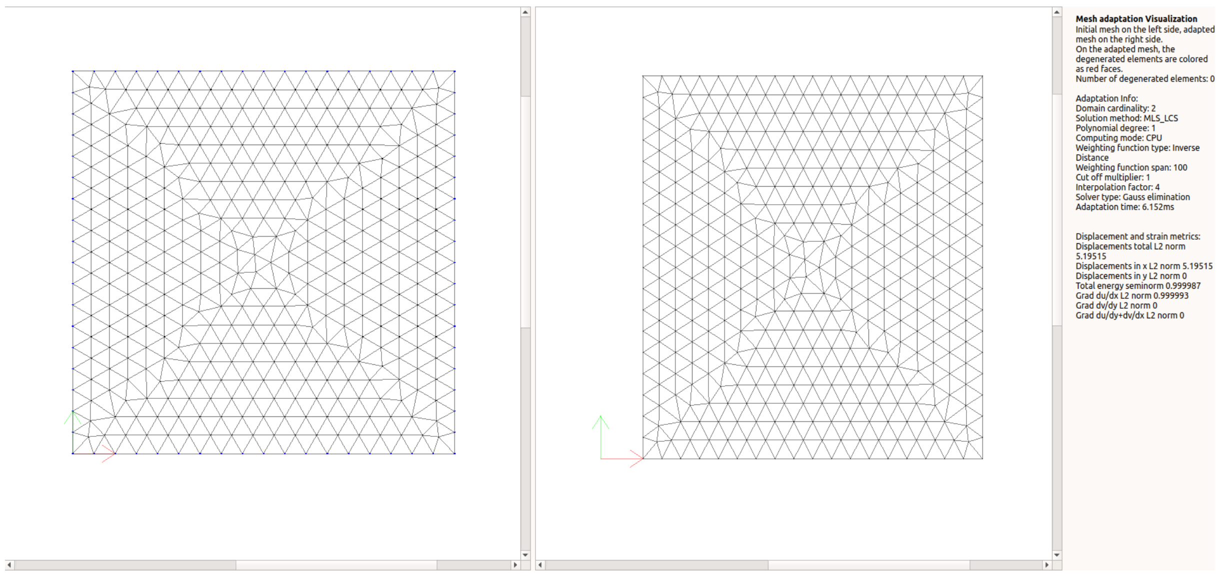
C.2.3 Consistency test on the constant grad du/dx

The constant gradients are tested to verify the consistency of the displacement fields with the expected ones.

The tested mesh is stretched with a unit displacement on the left boundary and holding still the right boundary. We expect the stretch to distribute in a uniform way within the interior and receive a constant gradient du/dx on entire the field,
$du/dx = 1/9$, $\|du/dx\| = 1$, $\|dv/dy\| = 0$, $\|du/dy + dv/dx\| = 0$

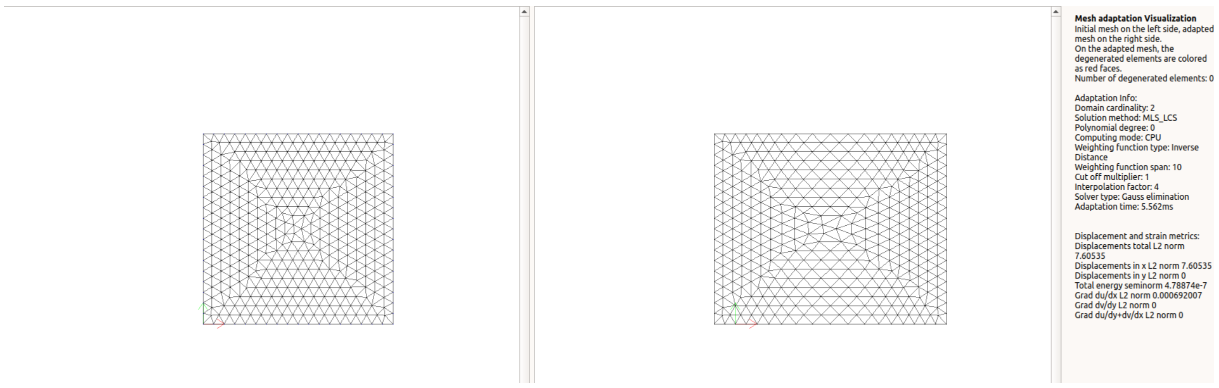
The test is successful for the 1st degree MLS solution, as the displacement gradient du/dx is constant and it has the expected value. The rest of the gradients are again correct with zero constant value.

The 0-degree solution exhibits a poor performance for a high interpolation parameter value $a = 4$. The results are improved to a sufficient practical degree, if the interpolation parameter is reduced to $a = 1.5$. The gradient fields for the two MLS solutions are shown in the figures C.14 and C.15, indicating no defects for the 1st degree and slight defects for the 0-degree.



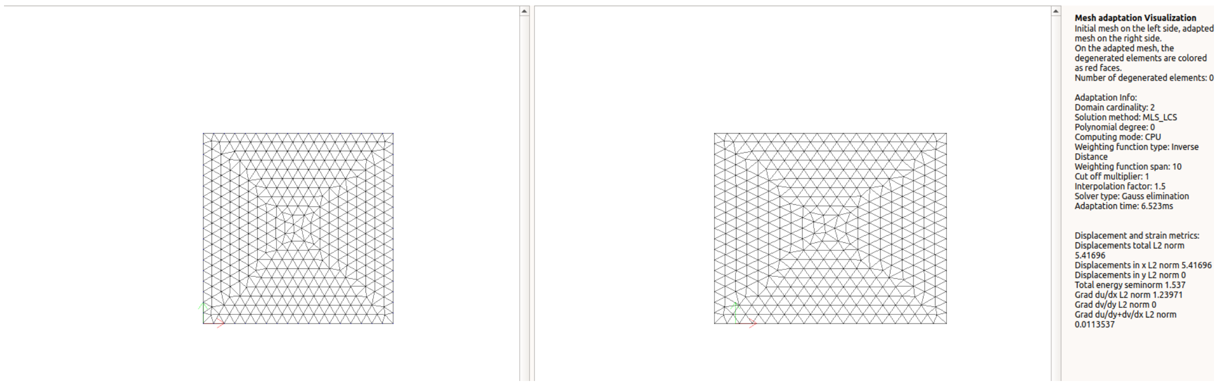
```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CPU
Weighting function type: Inverse
Distance
Weighting function span: 100
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 6.152ms


Displacement and strain metrics:
Displacements total L2 norm
5.19515
Displacements in x L2 norm 5.19515
Displacements in y L2 norm 0
Total energy seminorm 0.999987
Grad du/dx L2 norm 0.999993
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0
```

Figure C.11: The meshes before and after the unit stretch in x direction of the square domain with size 9x9. Adaptation with the 1st degree MLS method. On the right panel, the settings of the MLS methods are shown, as long as the L^2 displacement and gradient fields measurements.



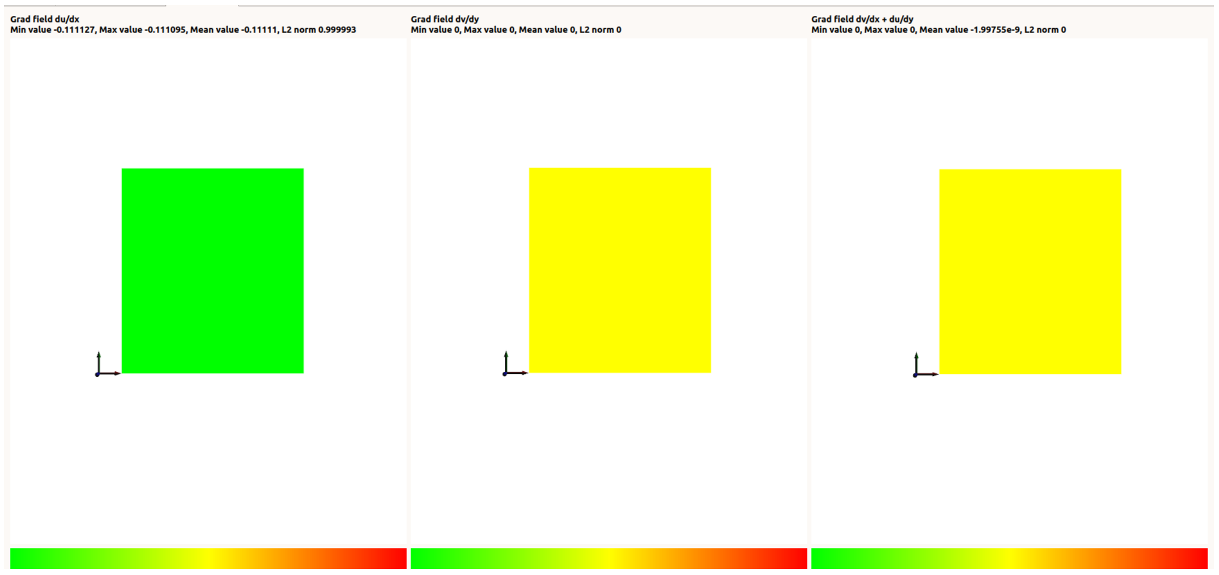
```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse
Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 5.562ms


Displacement and strain metrics:
Displacements total L2 norm
7.60535
Displacements in x L2 norm 7.60535
Displacements in y L2 norm 0
Total energy seminorm 4.78874e-7
Grad du/dx L2 norm 0.000692007
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0
```

Figure C.12: Adaptation with the 0-degree MLS method.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse
Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 1.5
Solver type: Gauss elimination
Adaptation time: 6.523ms


Displacement and strain metrics:
Displacements total L2 norm
5.41696
Displacements in x L2 norm 5.41696
Displacements in y L2 norm 0
Total energy seminorm 1.537
Grad du/dx L2 norm 1.23971
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm
0.0113537
```

Figure C.13: Adaptation with the 0-degree MLS method and an optimized interpolation parameter a.



```
Grad field du/dx
Min value -0.111127, Max value -0.111095, Mean value -0.11111, L2 norm 0.999993

Grad field dv/dy
Min value 0, Max value 0, Mean value 0, L2 norm 0

Grad field dv/dx + du/dy
Min value 0, Max value 0, Mean value -1.99755e-9, L2 norm 0
```

Figure C.14: The displacement gradient fields after a unit stretch in x direction of the square domain with size 9x9. Adaptation with the 1^{st} degree MLS method and an optimed interpolation parameter a.

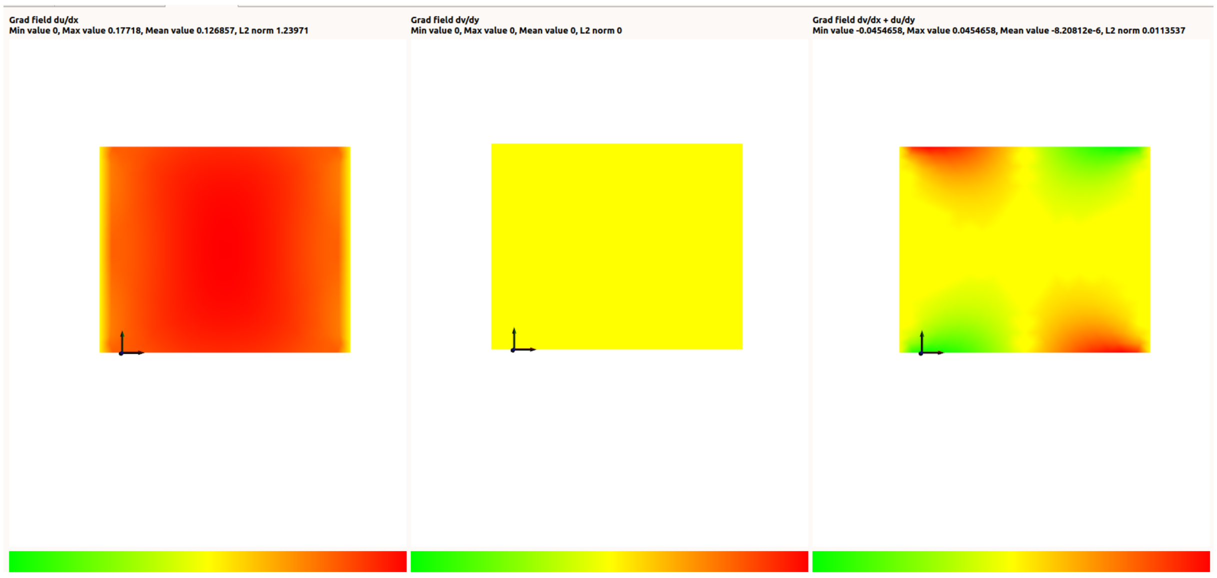


Figure C.15: The displacement gradient fields after a unit stretch in x direction of the square domain with size 9x9. Adaptation with the 0-degree MLS method and an optimed interpolation parameter a.

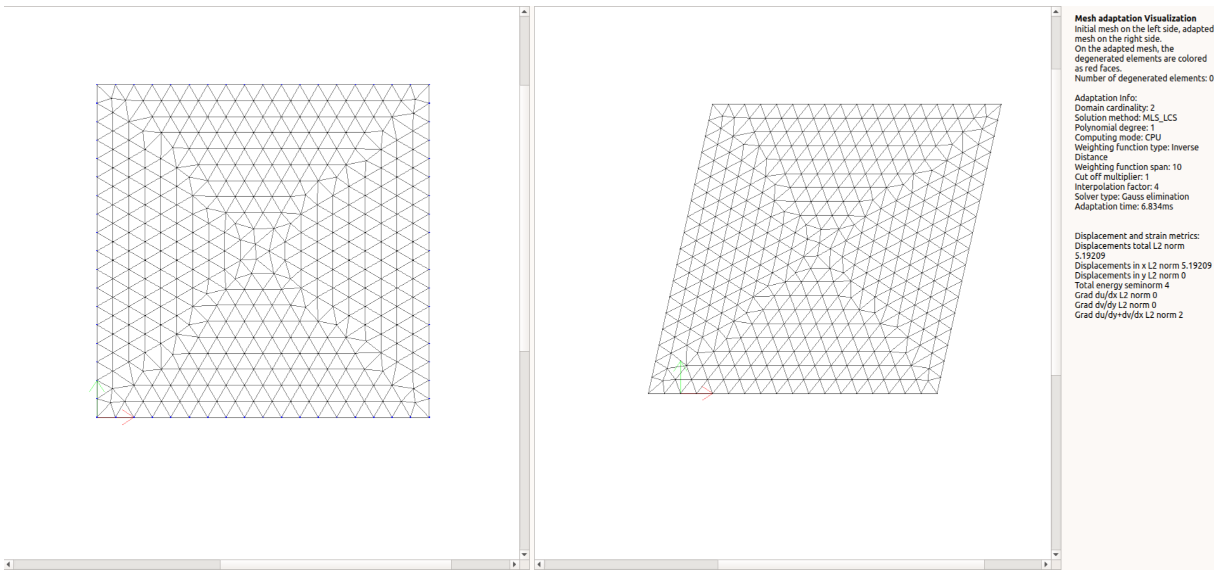
C.2.4 Consistency test on the constant grad du/dy+dv/dx

The constant gradients are tested to verify the consistency of the displacement fields with the expected ones.

The tested mesh is stretched in simple shear, with a unit displacement of the top boundary to the right and a unit displacement of the bottom boundary to the left. We expect the stretch to distribute in a uniform way within the interior and receive a constant gradient $du/dy + dv/dx$ on entire the field,
$du/dy + dv/dx = 2/9$, $\|du/dx\| = 0$, $\|dv/dy\| = 0$, $\|du/dy + dv/dx\| = 2$

The test is successful for the 1^{st} degree MLS solution, as the displacement gradient $du/dy + dv/dx$ is constant with the expected value. The rest of the gradients are also correct with zero constant value.

The 0-degree solution exhibits a poor performance for a high interpolation parameter value $a = 4$. The results are improved to a sufficient practical degree, if the interpolation parameter is reduced to $a = 1.3$. The gradient fields for the two MLS solutions are shown in the figures C.19 and C.20, indicating no defects for the 1^{st} degree and slight defects for the 0-degree.

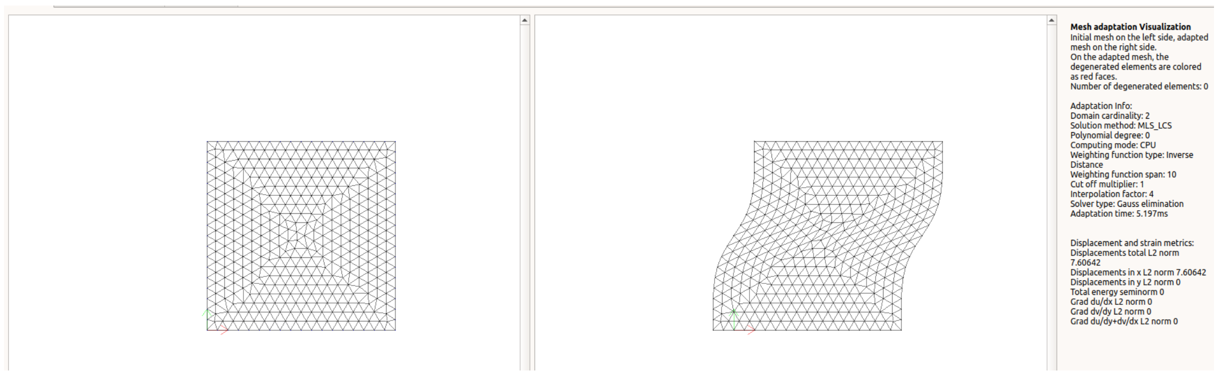


Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 6.834ms

Displacement and strain metrics:
Displacements total L2 norm 5.19209
Displacements in x L2 norm 5.19209
Displacements in y L2 norm 0
Total energy seminorm 4
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 2

Figure C.16: The meshes before and after a simple shear stretch (two opposite boundaries moved by a unit in opposite directions) of the square domain with size 9x9. Adaptation with the 1^{st} degree MLS method. On the right panel, the settings of the MLS methods are shown, as long as the L^2 displacement and gradient fields measurements.

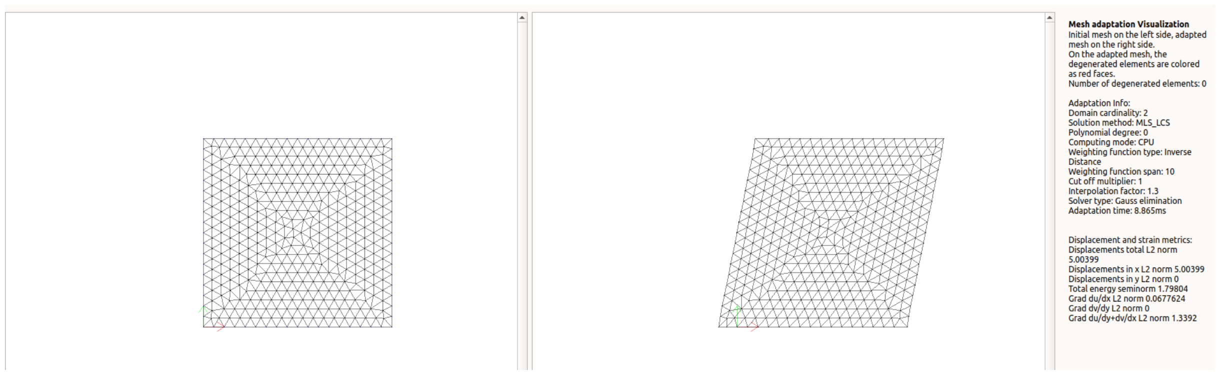


Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 4
Solver type: Gauss elimination
Adaptation time: 5.197ms

Displacement and strain metrics:
Displacements total L2 norm 7.60642
Displacements in x L2 norm 7.60642
Displacements in y L2 norm 0
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0

Figure C.17: Adaptation with the 0-degree MLS method.

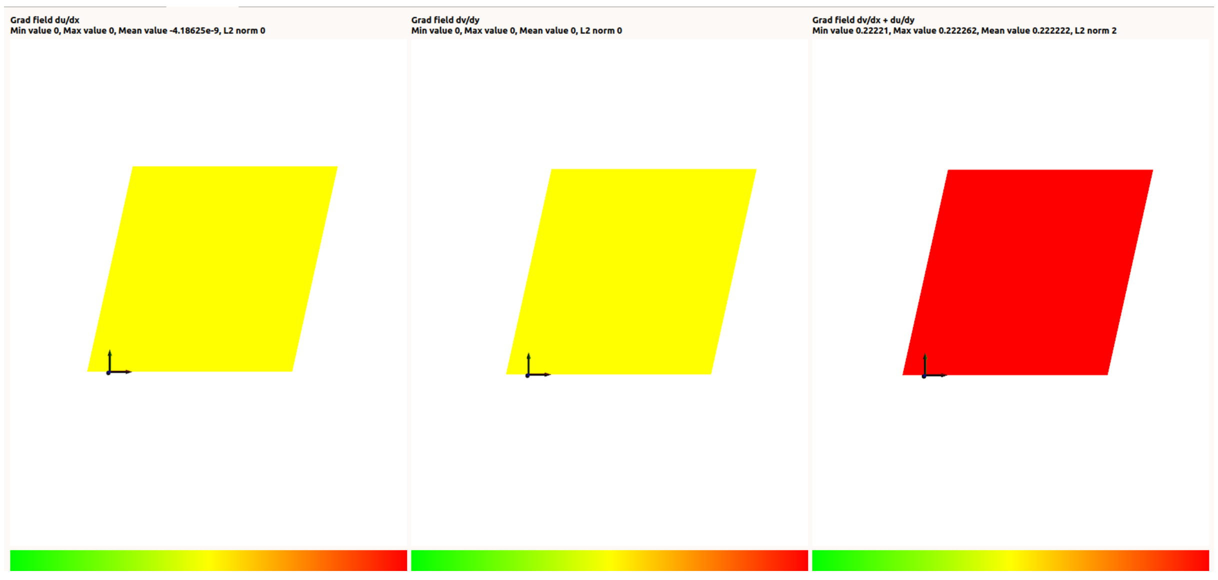


Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CPU
Weighting function type: Inverse Distance
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 1.3
Solver type: Gauss elimination
Adaptation time: 8.865ms

Displacement and strain metrics:
Displacements total L2 norm 5.00399
Displacements in x L2 norm 5.00399
Displacements in y L2 norm 0
Total energy seminorm 1.79804
Grad du/dx L2 norm 0.0677624
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 1.3392

Figure C.18: Adaptation with the 0-degree MLS method and an optimized interpolation parameter a.



Grad field du/dx
Min value 0, Max value 0, Mean value -4.18625e-9, L2 norm 0

Grad field dv/dy
Min value 0, Max value 0, Mean value 0, L2 norm 0

Grad field dv/dx + du/dy
Min value 0.22221, Max value 0.222262, Mean value 0.222222, L2 norm 2

Figure C.19: The displacement gradient fields after a simple shear stretch of the square domain with size 9x9. Adaptation with the 1st degree MLS method and an optimed interpolation parameter a.

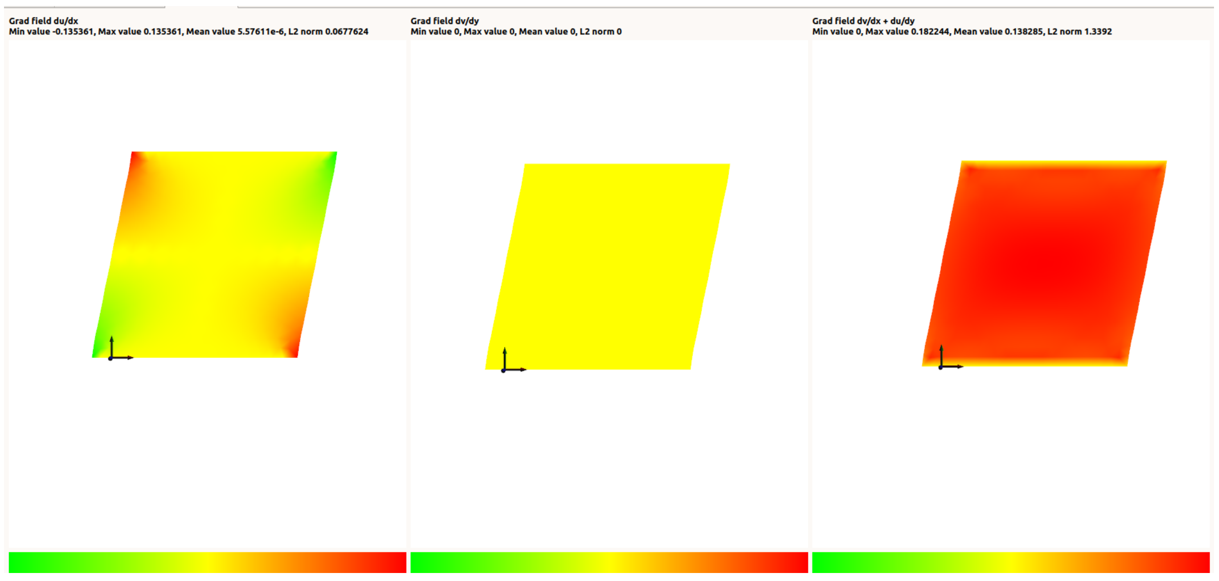

Grad field du/dx
Min value -0.135361, Max value 0.135361, Mean value 5.57611e-6, L2 norm 0.0677624
Grad field dv/dy
Min value 0, Max value 0, Mean value 0, L2 norm 0
Grad field dv/dx + du/dy
Min value 0, Max value 0.182244, Mean value 0.138285, L2 norm 1.3392

Figure C.20: The displacement gradient fields after a simple shear stretch (two opposite boundaries moved by a unit in opposite directions) of the square domain with size 9x9. Adaptation with the 0-degree MLS method and an optimed interpolation parameter a.

C.3 The performance of the MLS mesh adaptation with parametric analyses on feasibility and quality

In this section the performance of the MLS method is demonstrated in 2D mesh adaptations. The results are an extension of what is presented in Chapter 1 for 1D cases, see B. Though, here the performance analysis is not handled with interpolation error estimates, but with mesh quality metrics and a feasibility characterization by verifying non-degenerated mesh elements. The degree of the polynomial basis, consists again the main parameter of the analysis, as it can increase the performance of the method, though the goal is to find the limits of the low degrees (i.e. 0 & 1 degree), which give simple and easily handled fast algorithms. We limit the analyses up to the 4^{th} degree polynomial, as it consists the limit for accurate enough operations with single precision floating point numbers. Our interest in single precision floating point numbers is a particular one, as the computations on graphic processing units (i.e. CUDA and OpenCL implementations) with single precision are usually more than twice fast as with double precision. The use of single precision numbers has a slight effect on the quality results, especially for high degrees. There the linear systems are of poor condition, and sensitive information for high degrees are being lost or corrupted.

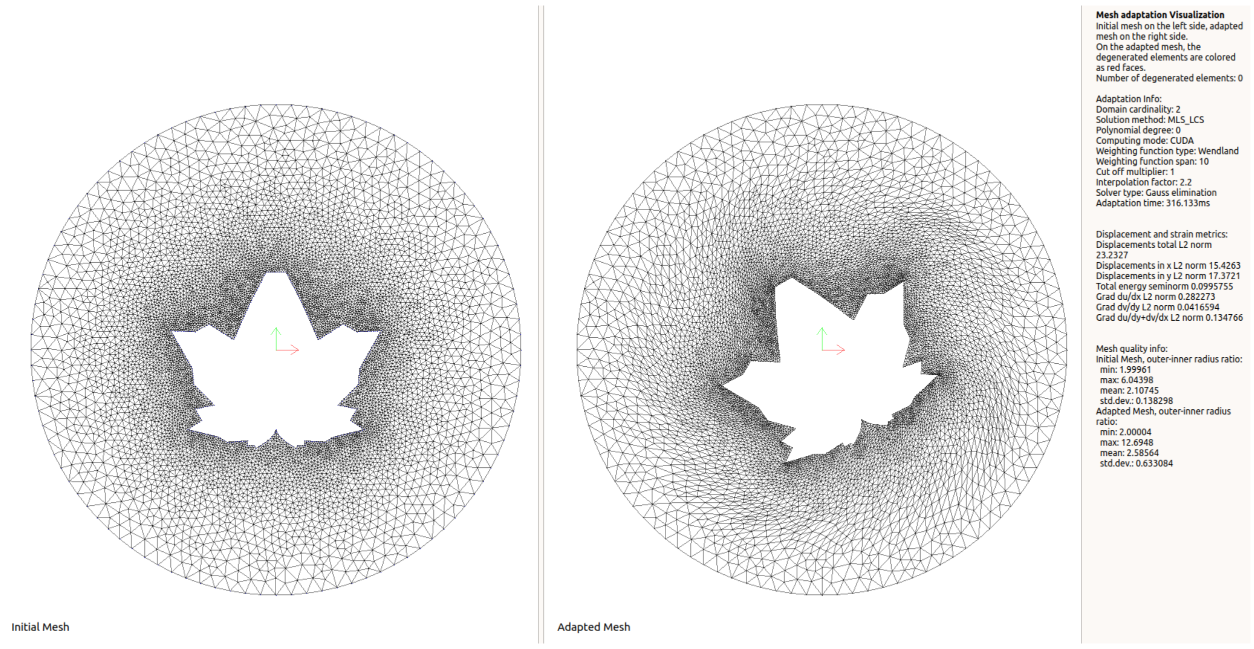
The analysis is oriented around the properties and the parameters of the weighting function, expressed in the general form of eq. B.3, where $Q(d) = \left(1 - \frac{d}{r}\right)^4 \left(4\frac{d}{r} + 1\right)$ is the Wendland C^2

$$w(d) = \frac{\left(1 - \frac{d}{r}\right)^4 \left(4\frac{d}{r} + 1\right)}{\left(\frac{d}{r}\right)^a + 10^{-12}} \tag{C.3}$$

where we identify, the interpolation parameter a and the support size parameter r. Both of the parameters can influence the success of the method, effecting on the form of the weighting function and consequently on the character of the approximation with the MLS method.

The application pool is narrowed down to a single benchmark problem, to investigate the method's properties and abilities. For more applications one can refer to the preceding work on the subject, see Τουρής [2016], altough the weighting functions used there are non-interpoland, hence one can find only a partial performance presentation of the MLS method. The test case is such that large displacement variations are occuried in closely spaced boundary nodes. Such cases emerge the need for a strict interpolation on the boundary displacements, as has been explained in the introduction of this chapter. Moreover, the inital poor quality of some mesh elements is an extra chalenge. The benchmarked mesh has an outer radius $R = 11$ and an inner radius $r = 5$, see Figure C.21. The total number of the nodes is 7547, out of which 337 are located on the inner boundary and 100 on the outer boundary. The total number of elements is 14657. The given mesh has the following quality statistics, $meanQ = 0.1672$, $maxQ = 0.793$, $std.dev.Q = 0.09$.

The tested displacements are various rotations of the inner "leaf" boundary as a rigid contour around the global origin point, and a fixed outer boundary. Such rotations induce displacements with various directions and magnitudes from the nodes on the inner boundary (sample points) to the interior nodes (stationary points). Hence, the data are strongly variable for close spaced nodes. The benchmark consists of various inner boundary rotation angles, i.e. 30deg, 45deg, 60deg and 80deg. The goal is to investigate the feasibility and the adapted mesh quality, for every polynomial degree, and for some combinations of the weighting function parameter values, i.e. a and r. The set of values of the interpolation parameter a is



Mesh adaptation Visualization
Initial mesh on the left side, adapted mesh on the right side.
On the adapted mesh, the degenerated elements are colored as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 10
Cut off multiplier: 1
Interpolation factor: 2.2
Solver type: Gauss elimination
Adaptation time: 316.133ms

Displacement and strain metrics:
Displacements total L2 norm 23.2327
Displacements in x L2 norm 15.4263
Displacements in y L2 norm 17.3721
Total energy seminorm 0.0995755
Grad du/dx L2 norm 0.282273
Grad dv/dy L2 norm 0.0416594
Grad du/dy+dv/dx L2 norm 0.134766

Mesh quality info:
Initial Mesh, outer-inner radius ratio:
min: 1.99961
max: 6.04398
mean: 2.10745
std.dev.: 0.138298
Adapted Mesh, outer-inner radius ratio:
min: 2.00004
max: 12.6948
mean: 2.58564
std.dev.: 0.633084

Figure C.21: Illustration of the benchmarked mesh. Outer radius $R = 11$, inner radius $r = 5$, 7547 total number of nodes, 337 nodes on the inner boundary and 100 nodes on the outer boundary. Quality metrics $meanQ = 0.1672$, $maxQ = 0.793$, $std.dev.Q = 0.09$.

defined according to the various published research on 2D surface reconstruction problems, the results in Chapter 1 for 1D cases, and numerical expirementation on the problem. The support size parameter r, is defined considering the mesh dimensions, and more specifically the inner and outer radii, the decaying speeds of the weighting function which is influenced by the parameter a, and the defined polynomial degrees. Both sets of values are as below

$$\begin{aligned} a &= \{0,\ 1,\ 2,\ 3,\ 4,\ 5,\ 6\} \\ r &= \{6,\ 8,\ 10,\ 12,\ 15,\ 18,\ 20,\ 25,\ 30,\ 40\} \end{aligned} \tag{C.4}$$

Higher values of the support parameter r have been tested up to $r = 200$ but the metrics seem to be invariable above the value of $r = 40$.

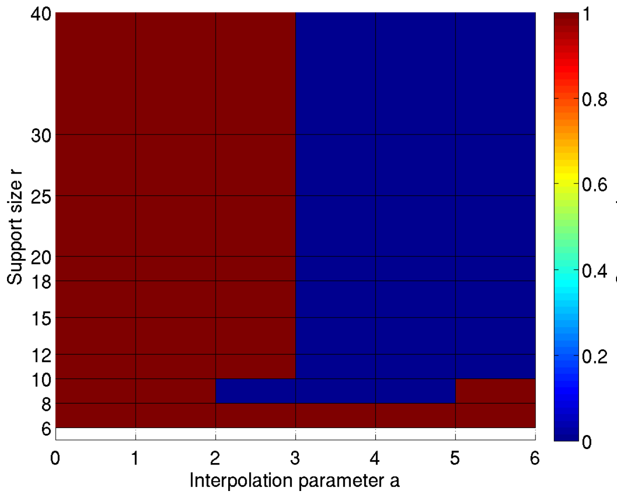
C.3.1 Feasibility analysis

In this section the feasibility of the MLS method is investigated, i.e. the ability to give a solution without degenerated elements.

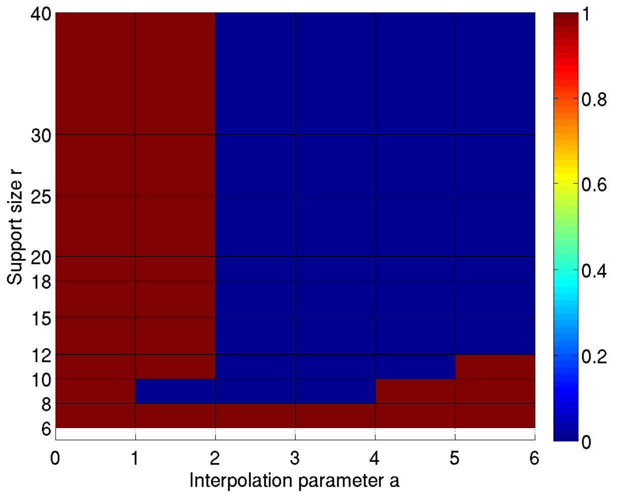
The results are presented in a graphical mode, i.e. color mapped 2D surfaces. There are five surfaces (i.e. the number of different polynomials cases) per rotation angle case, see the four rotation angles (i.e. 30deg, 45deg, 60deg, 80deg) in the Figures C.22, C.23, C.24 and C.25.

Each surface is plotting on the horizontal axis the interpolation parameter a, and the support size parameter r on the vertical axis. There, every test case is represented by a grid lines crossing point. The color map is set to represent the 0 degenerated elements cases with the dark blue color (0 value), and every other case with one or more degenerated elements with the dark red color corresponding to the value 1.

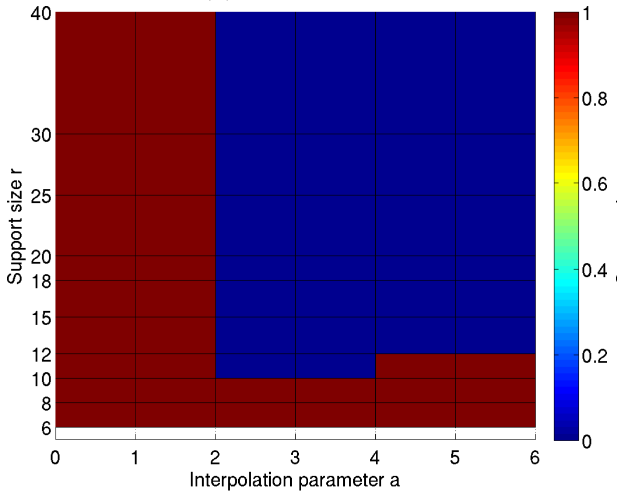
The parametric graphs, are indicating a broad enough domain of robustness in the two parameters, in most of the cases eccentrically located on higher values of a and r for all the polynomial degrees. Higher polynomial degrees seem to possess a broader robust range in a parameter and a narrower one for the r parameter, compared to the lower degrees. As the induced rotations are becoming higher and the problem more tough, for all the polynomials the parametric domain of robustness is getting reduced, in a faster sense on lower degree polynomials. For all the polynomials, as the rotation increases the range of a values is narrowing down to $a = 3$, faster than the range of r, which tends to higher values in most of the cases.



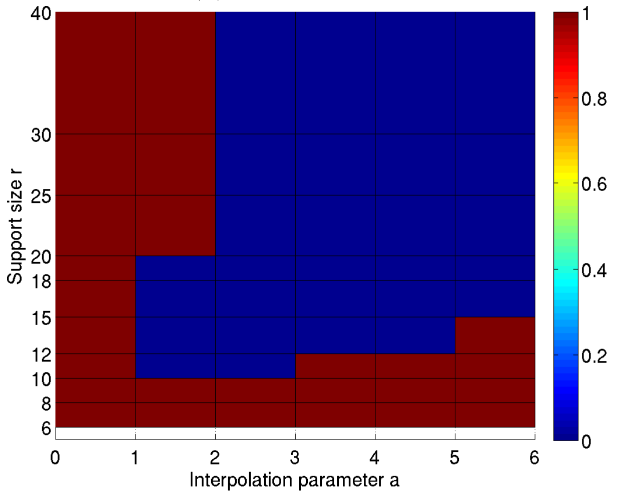
(a) 0-degree MLS.



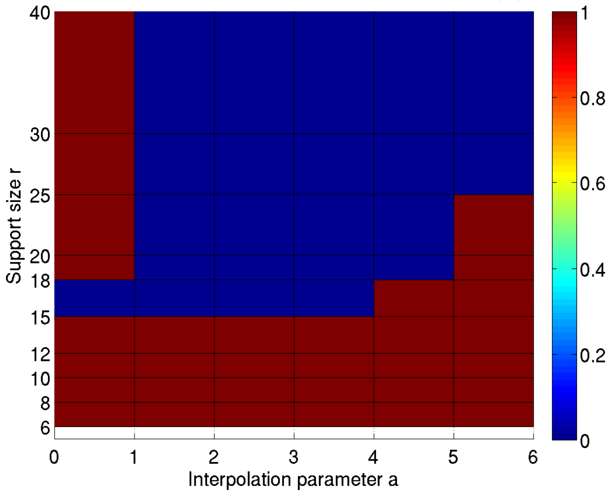
(b) 1st degree MLS.



(c) 2nd degree MLS.

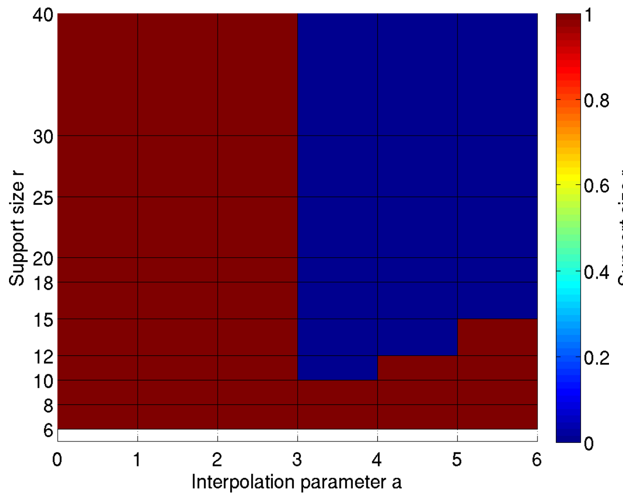


(d) 3rd degree MLS.

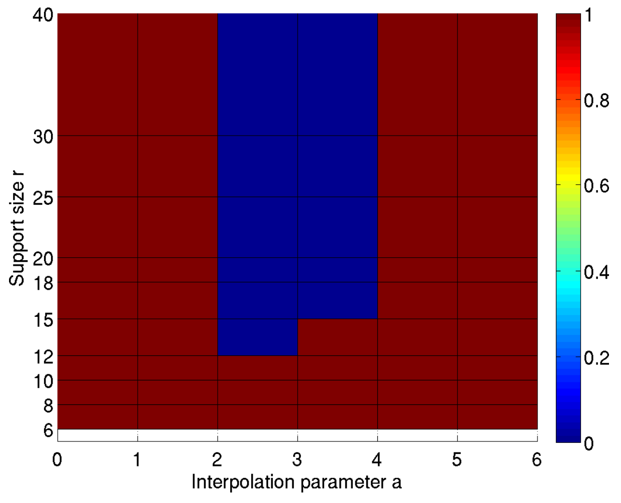


(e) 4th degree MLS.

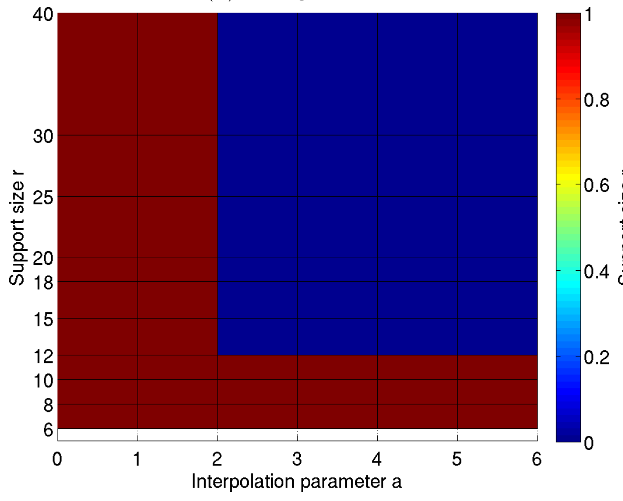
Figure C.22: Feasibility surfaces of MLS adaptations, for rotation angle of 30deg in the benchmark problem. The blue colored parametric space corresponds to a feasible adaptation with no degenerated elements. Each surface corresponds to a polynomial degree, 0-4.



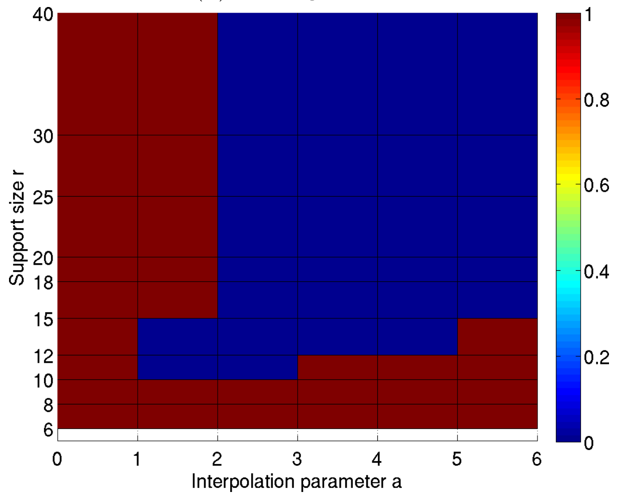
(a) 0-degree MLS.



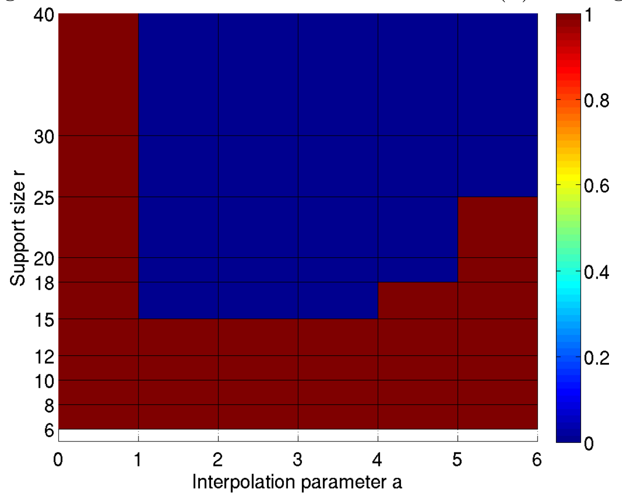
(b) 1st degree MLS.



(c) 2nd degree MLS.

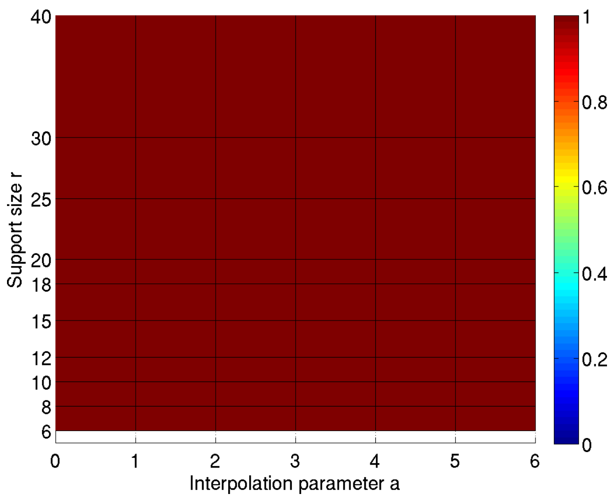


(d) 3rd degree MLS.

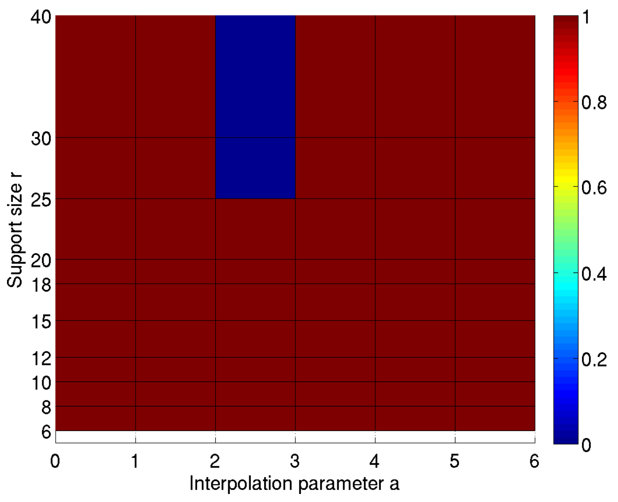


(e) 4th degree MLS.

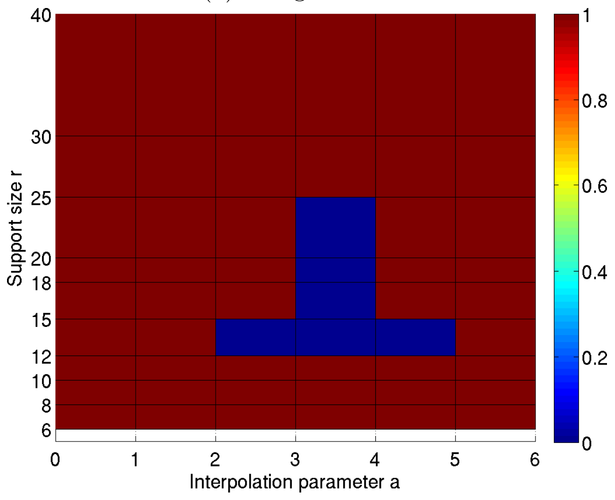
Figure C.23: Feasibility surfaces of MLS adaptations, for rotation angle of 45deg in the benchmark problem. The blue colored parametric space corresponds to a feasible adaptation with no degenerated elements. Each surface corresponds to a polynomial degree, 0-4.



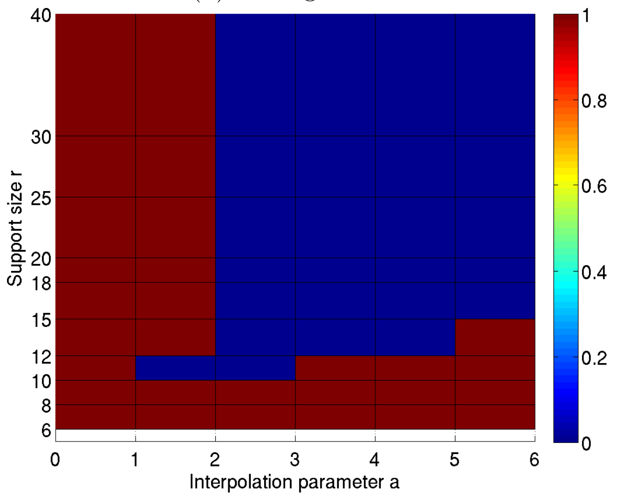
(a) 0-degree MLS.



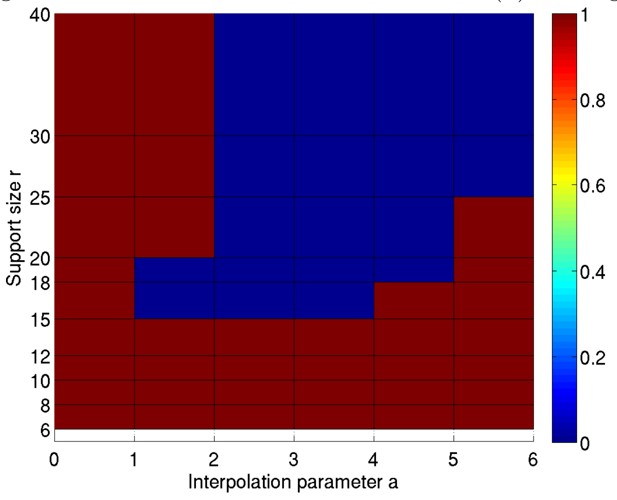
(b) 1^{st} degree MLS.



(c) 2^{nd} degree MLS.

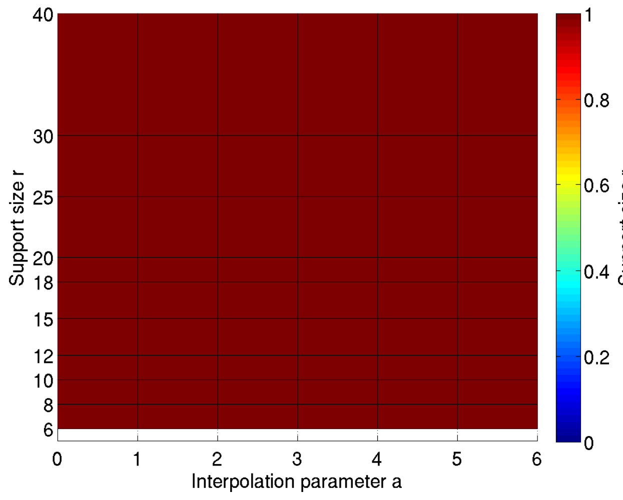


(d) 3^{rd} degree MLS.

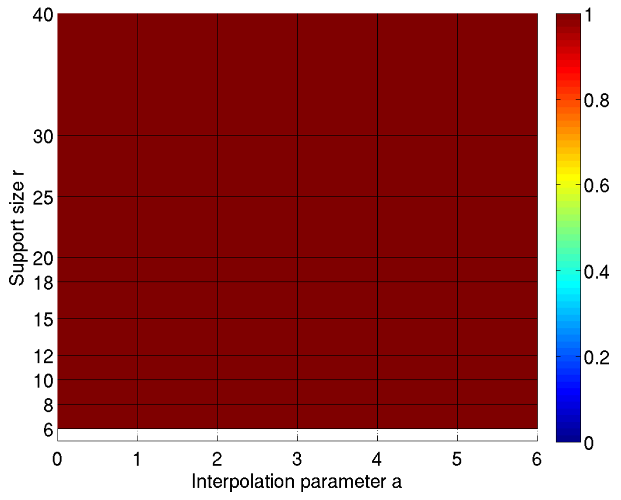


(e) 4^{th} degree MLS.

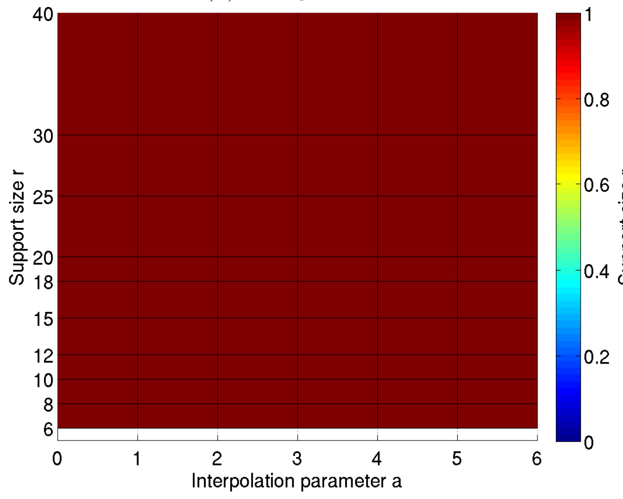
Figure C.24: Feasibility surfaces of MLS adaptations, for rotation angle of 60deg in the benchmark problem. The blue colored parametric space corresponds to a feasible adaptation with no degenerated elements. Each surface corresponds to a polynomial degree, 0-4.



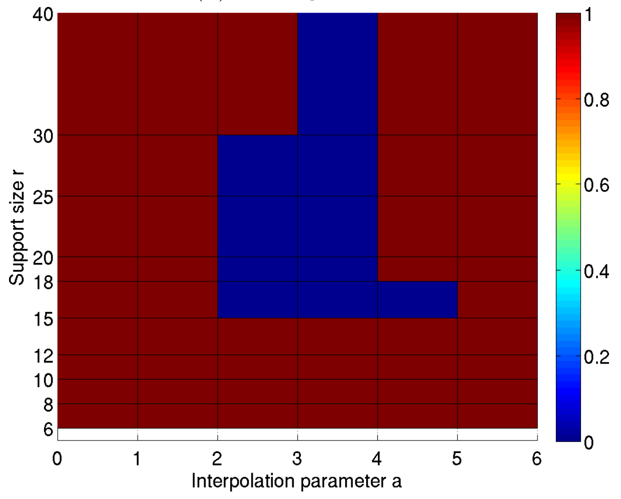
(a) 0-degree MLS.



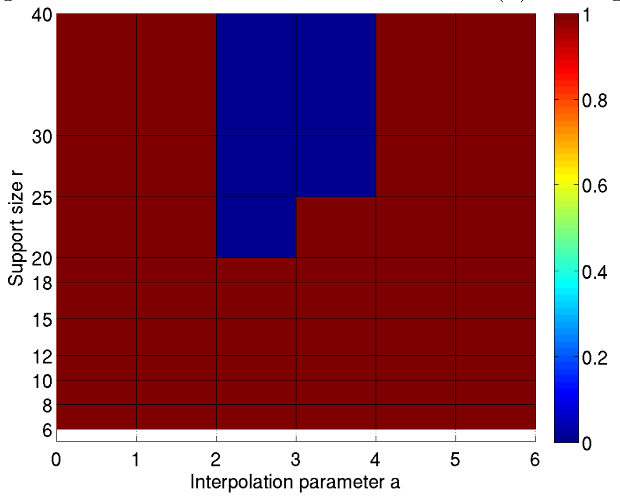
(b) 1st degree MLS.



(c) 2nd degree MLS.



(d) 3rd degree MLS.



(e) 4th degree MLS.

Figure C.25: Feasibility surfaces of MLS adaptations, for rotation angle of 80deg in the benchmark problem. The blue colored parametric space corresponds to a feasible adaptation with no degenerated elements. Each surface corresponds to a polynomial degree, 0-4.

C.3.2 Mesh quality analysis

Besides the meaning of mesh quality regarding the correct and sufficient refinement fitting a specific problem, another useful meaning is the mesh quality expressed through the regularity of the mesh elements, i.e. elements which keep a symmetry or equivalence of the edges or faces. This geometrical property has been proved important to describe mathematical models with better accuracy, as they become less prone to the numerical accuracy limits of a computing unit. The most common quality measures, representing the regularity of the elements, are the skewness, the elements size variation smoothness, and the aspect ratio.

One of the most commonly accepted practice, is to measure the deviation of the mininum and maximum angles, from an equiangle shape. For instance, such a mesh quality metric for triangular mesh elements is the following

$$\max\left[\frac{\theta_{max} - 60deg}{120deg}, \frac{60deg - \theta_{min}}{60deg}\right] \tag{C.5}$$

The above values can be mapped by an analyst to a mesh quality characterization, as for example in table C.1.

Mapping table of Equiangle skweness - Mesh Quality						
Value of Skewness	0-0.25	0.25-0.5	0.5-0.8	0.8-0.95	0.95-0.99	0.99-1.00
Cell Quality	Excellent	Good	Acceptable	Poor	Sliver	Degenerate

Table C.1: A mapping table of equiangle skewness values and mesh quality. (Andre Bakker, mesh quality in CFD)

The mapping of the table C.1 can be different and dependent on the special needs of the solved problem, the required quality of the results, the faster convergence and analysis time of some problems, the machine precision, etc.

With the above quality metric of eq. (C.5), the initial mesh has the following statistical measures
$meanQ = 0.1672$, $maxQ = 0.793$, $std.dev.Q = 0.09$. The max quality measure indicates that some mesh elements since the beginning are close to be qualified as poor.

The results are presented in a graphical mode, i.e. color mapped 2D surfaces. There are five surfaces (which is the number of different polynomials cases) per rotation angle case, plotting the $meanQ$ values (each $meanQ$ on the surface, is the mean value of Q among all the elements in the adapted mesh), and five more surfaces plotting the standard deviation values of the quality metric. The results for the 4 rotation angles (i.e. 30deg, 45deg, 60deg, 80deg), are shown in the Figures C.26, C.28, C.30 and C.32. Furthermore, the max quality metric values for the same adaptation cases are given in Figures C.27, C.29 and C.31.

Each surface is plotting on the horizontal axis the interpolation parameter a, and the support size parameter r on the vertical axis. There, every test case is represented by a grid lines crossing point. The color map is set to represent the values, starting from 0 value (the dark blue color, that corresponds to a perfect equilateral triangle), and up to the value 1 (the dark red color, degenerated solution). The color map for the surfaces of the standard deviation is set up to the value of 0.5.

Comments

First, it is observed that the max quality surfaces (max is worse quality as given in the table C.1) are closely following the patterns of the feasibility surfaces. This proves a robustness of the method, as the values of the weighting function parameters that give high quality metrics in early time for lower deformation in the mesh, the same values of the parameters will exhibit the degenerated solutions. And conversely, the most durable parameter values have the better mesh quality for lower deformation. Generally, the max value surfaces can be a reliable criterion with no prediction failure, and the necessary criterion if one is interested in strict quality of each element.

The mean quality surfaces can indicate a globally good condition even for cases where the solution is heavily degenerated, but locally. This case is the misinterpolation of the mesh around the displaced nodes, where the deformation energy is all exhausted locally, the rest of the nodes almost still, with the result of indicating the global condition seamingly good. This fact is justified by prior analysis and comments on the interpolation properties of the method, where it has been shown that the interpolation parameter (a) should be high enough for the used polynomial. In all the figures of the mean quality value we can see that spurious pattern of a seamingly good mesh overall. Thus, the mean quality surfaces can be a good index for an overall good condition and smoothness, but to verify the local good condition the max quality value surfaces is the best index, along with the standard deviation index to determine if the bad quality is severe, or limited where some repair operations could take place.

Regarding the quality metrics with variation of the polynomial degree, the higher degrees are the most robust in acceptable deformation, more robust on the parameters on the interpolation parameter a, but slightly less robust on the weighting function support parameter r, where higher values are preferred.

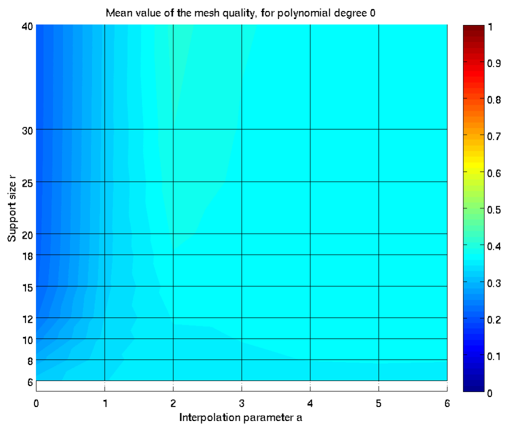
Observing the maximum mesh quality surfaces, we can deduce that the interpolation parameter a, with variation of the mesh deformation, is more robust around the value $a \approx 3$ for the 0-degree polynomial, $a \approx 2$ for the 1^{st} degree, and $a \approx 2.5$ for the polynomial degrees 2, 3 and 4. Furthermore, high values of the weighting function support size r seem to be robust through various magnitudes of the mesh deformation.

We observe, that the 0-degree polynomial can handle a rotation up to 30deg with acceptable max quality metric < 0.8 for an interpolation parameter $a \approx 3.5$ and $r > 15$. The result is illustrated in Figure C.34a.

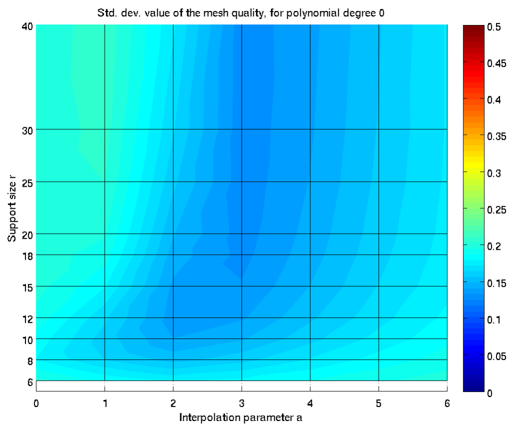
The 45deg rotation can be performed with an acceptable max quality metric value $Q \approx 0.8$ from the 1^{st} degree polynomial, and the parameters $a \approx 2$ and $r > 25$, see Figure C.34b.

And finally 60deg rotation can be performed with an acceptable max quality metric value $Q \approx 0.8$ from the 3^{rd} degree polynomial, with $a \approx 3$ and $r > 30$, the result is shown in Figure C.34c.

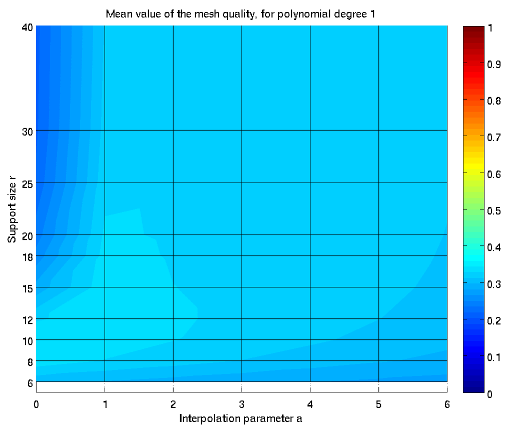
For the two last cases, one can see the mesh being distorted closer to the exterior boundary, which fact can be convenient to preserve the quality of the solution closer to the essential inner boundary. Moreover, few large mesh elements with poor quality can be corrected by the analyst mannually, or some additional algorithm.



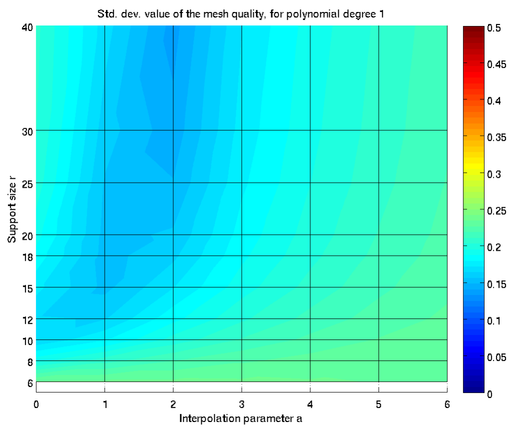
(a) 0-degree MLS, mean quality.



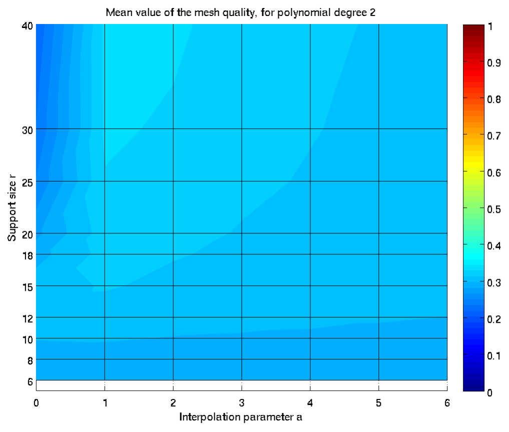
(b) 0-degree MLS, std. dev. of quality.



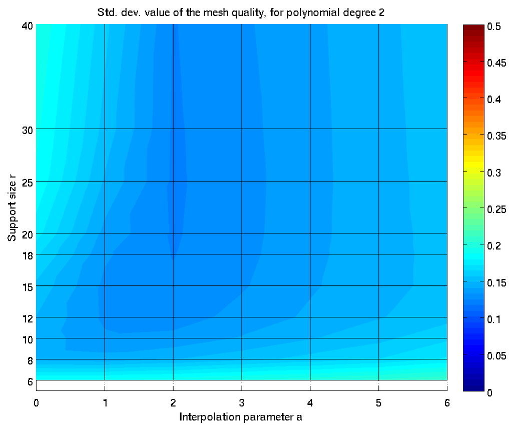
(c) 1^{st} degree MLS, mean quality.



(d) 1^{st} MLS, std. dev. of quality.

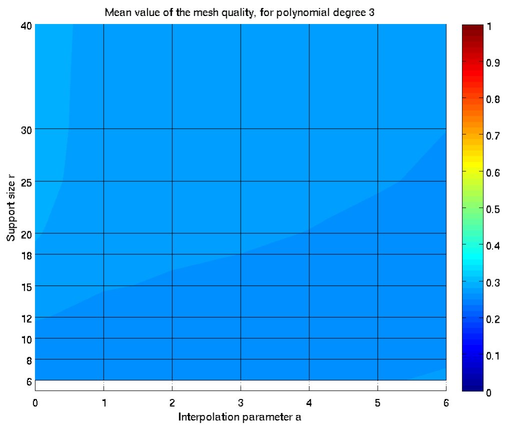


(e) 2^{nd} degree MLS, mean quality.

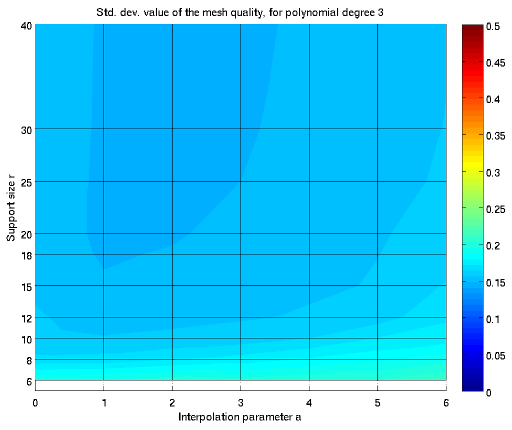


(f) 2^{nd} MLS, std. dev. of quality.

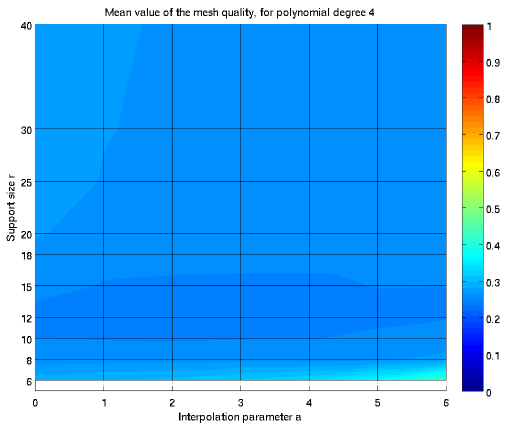
Figure C.26: Mesh quality surfaces, for a rotation angle of 30deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



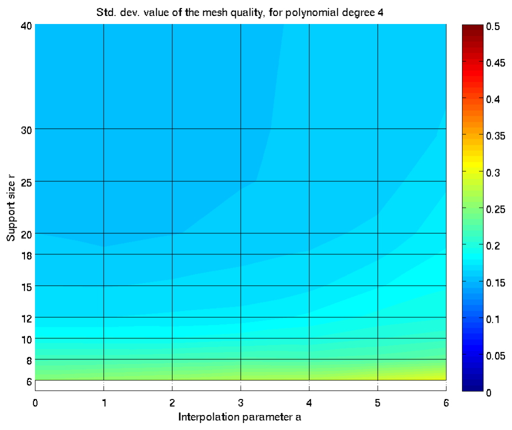
(g) 3rd degree MLS, mean quality.



(h) 3rd MLS, std. dev. of quality.

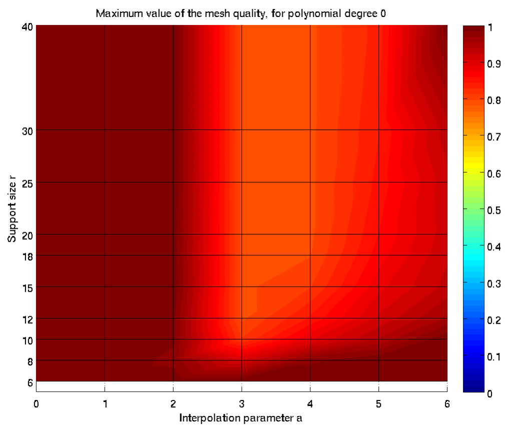


(i) 4th degree MLS, mean quality.

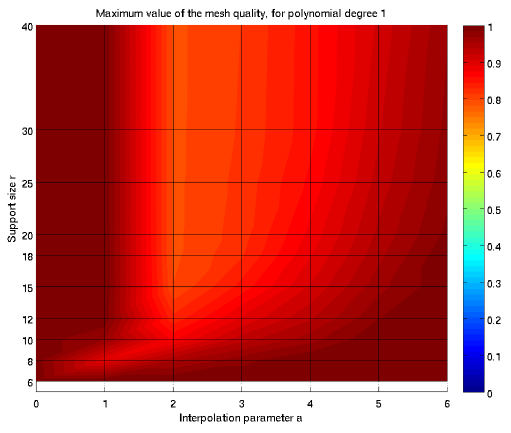


(j) 4th MLS, std. dev. of quality.

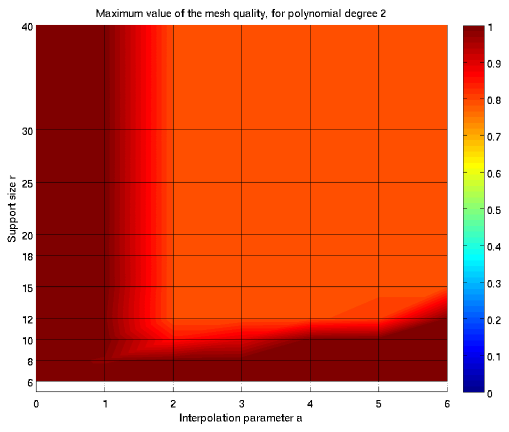
Figure C.26: Mesh quality surfaces, for a rotation angle of 30deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



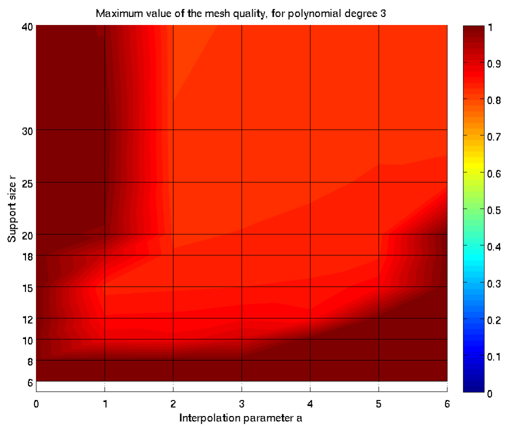
(a) 0-degree MLS.



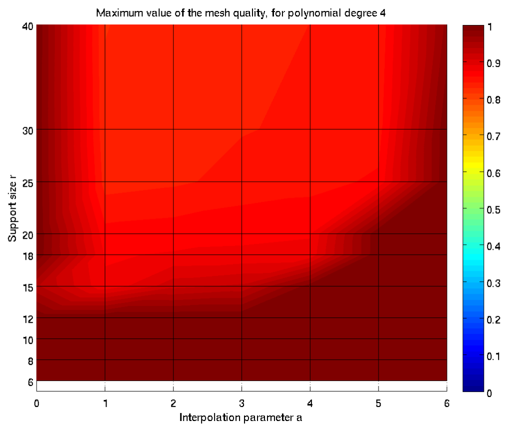
(b) 1st degree MLS.



(c) 2nd degree MLS.

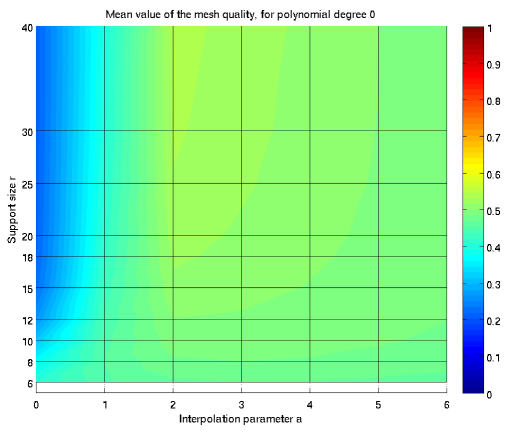


(d) 3rd degree MLS.

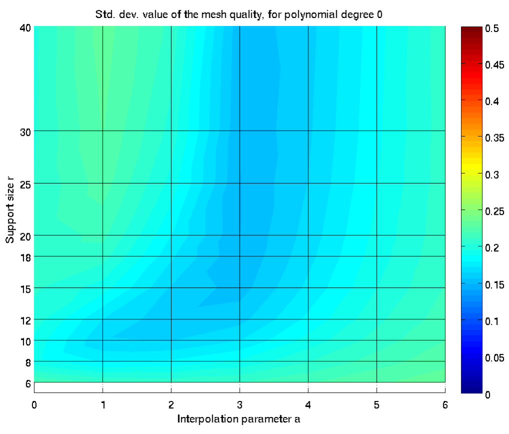


(e) 4th degree MLS.

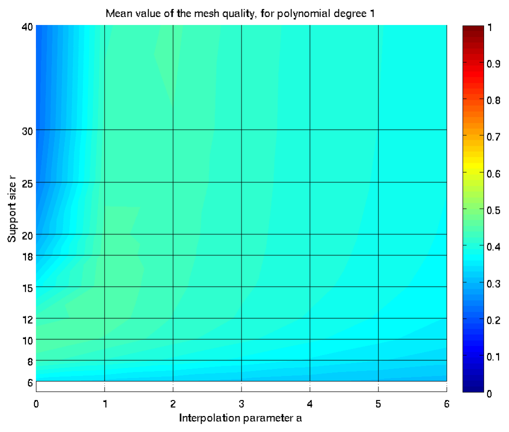
Figure C.27: Max quality surfaces of MLS adaptations, for rotation angle of 30deg in the benchmark problem. Each surface corresponds to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



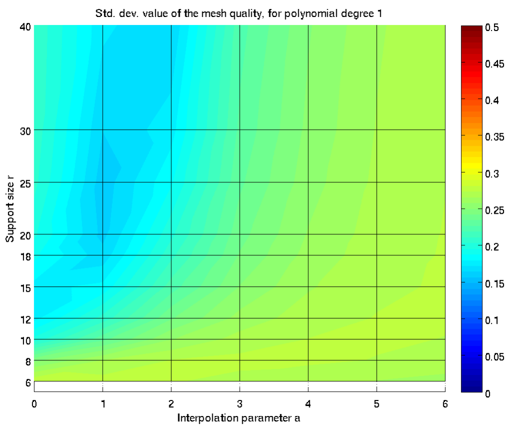
(a) 0-degree MLS, mean quality.



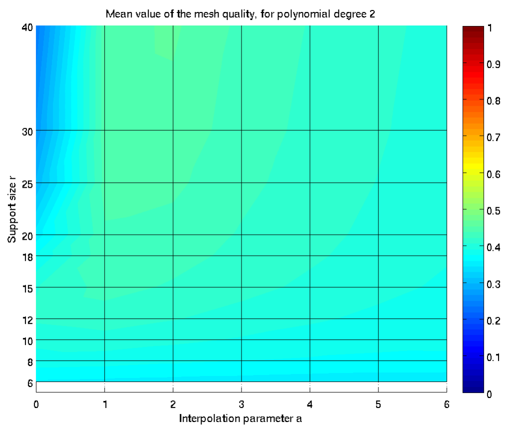
(b) 0-degree MLS, std. dev. of quality.



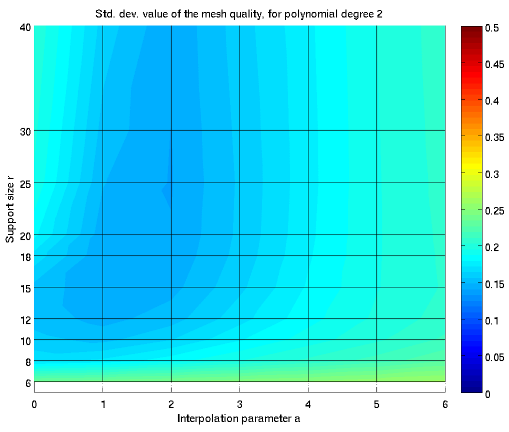
(c) 1st degree MLS, mean quality.



(d) 1st MLS, std. dev. of quality.

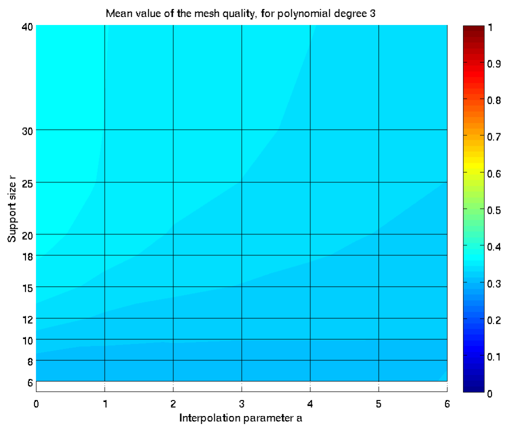


(e) 2nd degree MLS, mean quality.

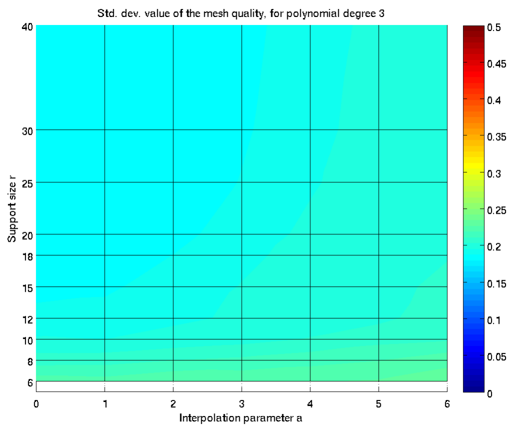


(f) 2nd MLS, std. dev. of quality.

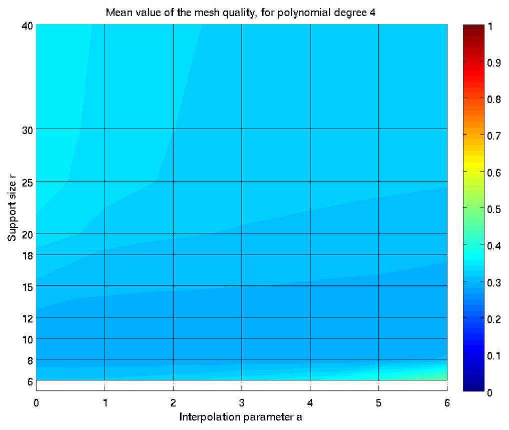
Figure C.28: Mesh quality surfaces, for a rotation angle of 45deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



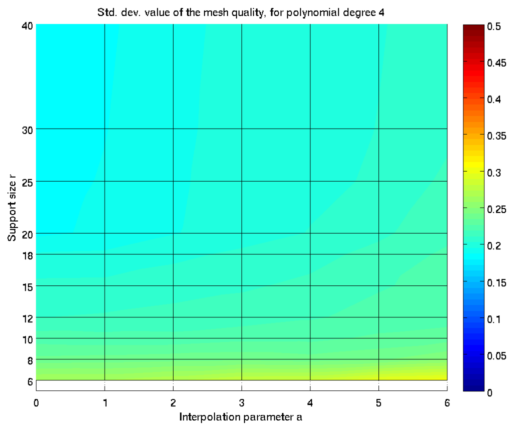
(g) 3^{rd} degree MLS, mean quality.



(h) 3^{rd} MLS, std. dev. of quality.

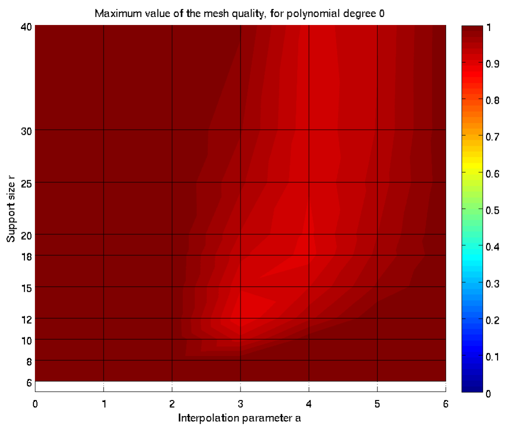


(i) 4^{th} degree MLS, mean quality.

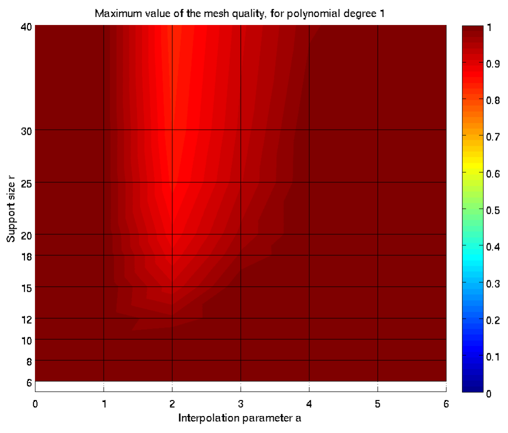


(j) 4^{th} MLS, std. dev. of quality.

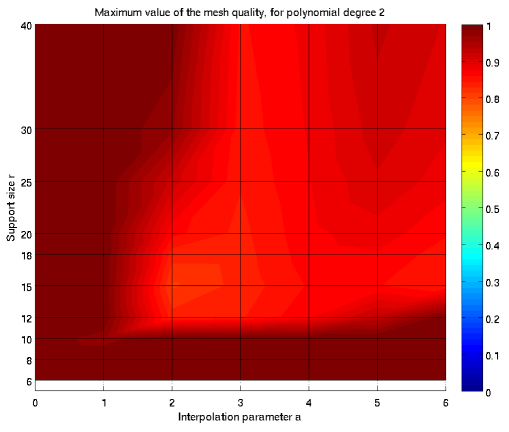
Figure C.28: Mesh quality surfaces, for a rotation angle of 45deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



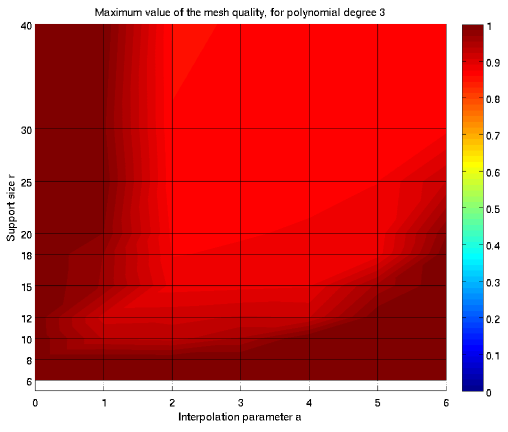
(a) 0-degree MLS.



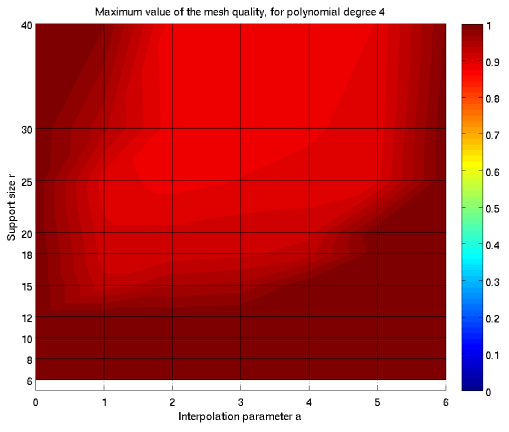
(b) 1st degree MLS.



(c) 2nd degree MLS.

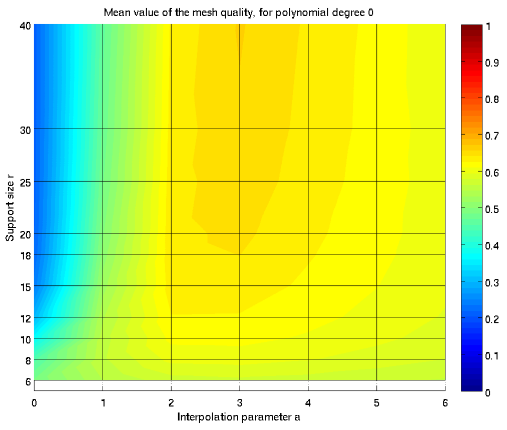


(d) 3rd degree MLS.

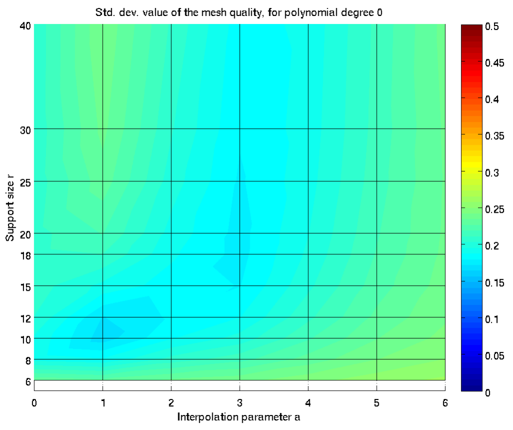


(e) 4th degree MLS.

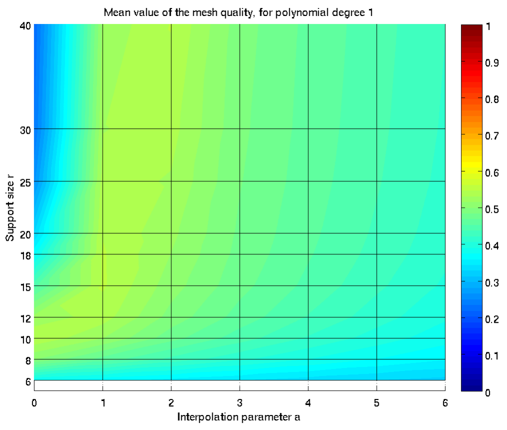
Figure C.29: Max quality surfaces of MLS adaptations, for rotation angle of 45deg in the benchmark problem. Each surface corresponds to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



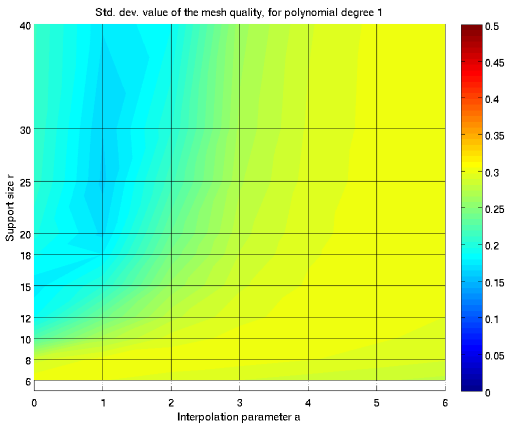
(a) 0-degree MLS, mean quality.



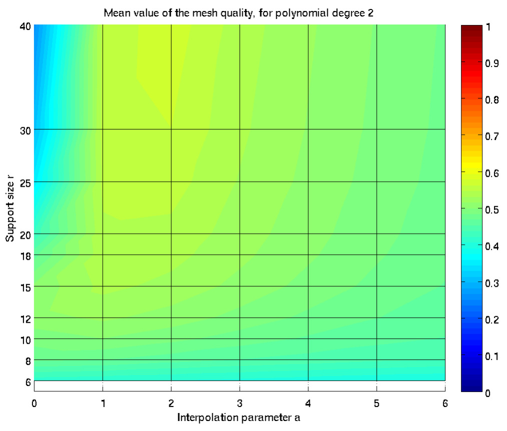
(b) 0-degree MLS, std. dev. of quality.



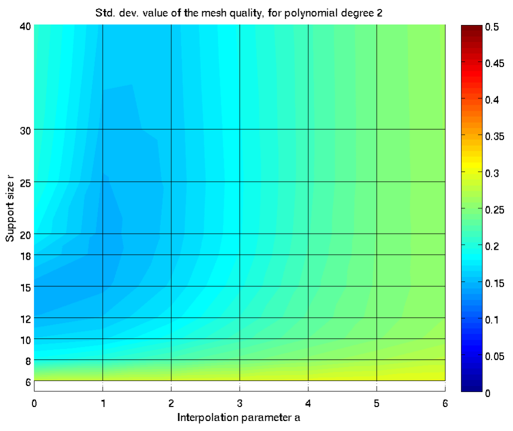
(c) 1^{st} degree MLS, mean quality.



(d) 1^{st} MLS, std. dev. of quality.

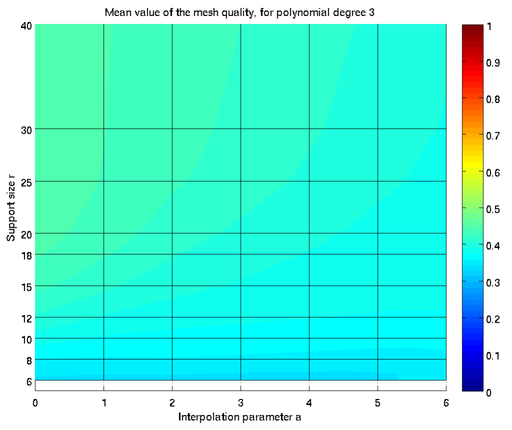


(e) 2^{nd} degree MLS, mean quality.

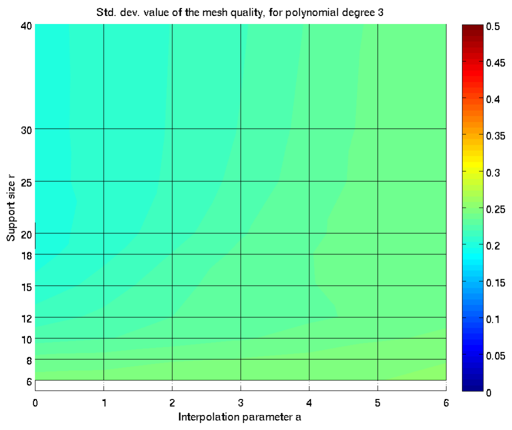


(f) 2^{nd} MLS, std. dev. of quality.

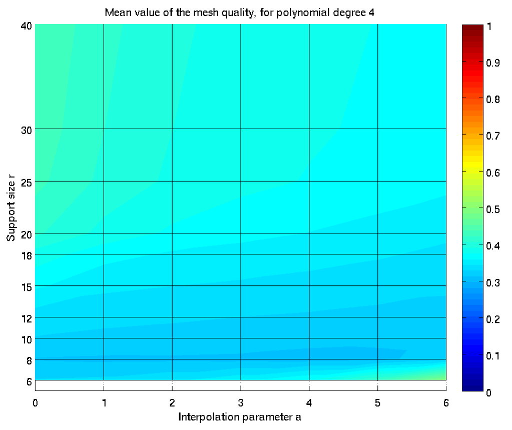
Figure C.30: Mesh quality surfaces, for a rotation angle of 60deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



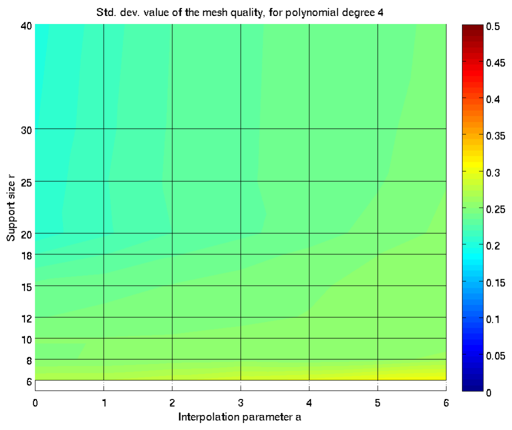
(g) 3rd degree MLS, mean quality.



(h) 3rd MLS, std. dev. of quality.

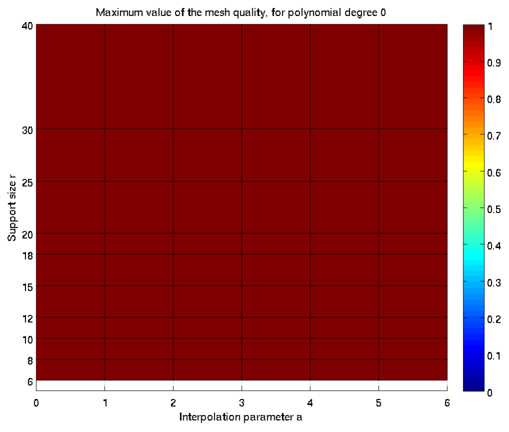


(i) 4th degree MLS, mean quality.

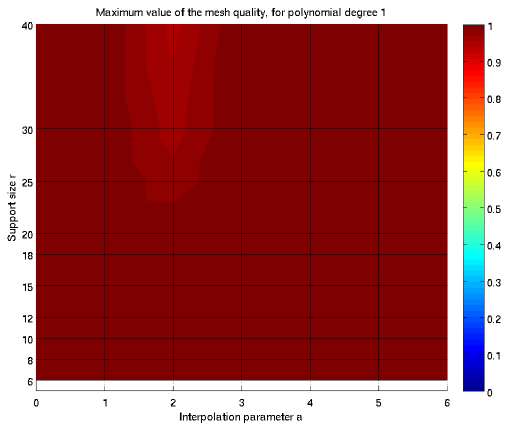


(j) 4th MLS, std. dev. of quality.

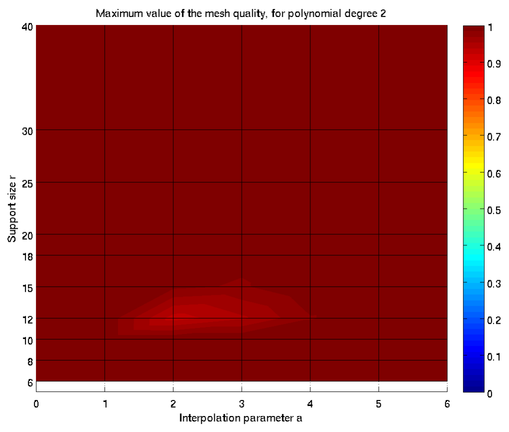
Figure C.30: Mesh quality surfaces, for a rotation angle of 60deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



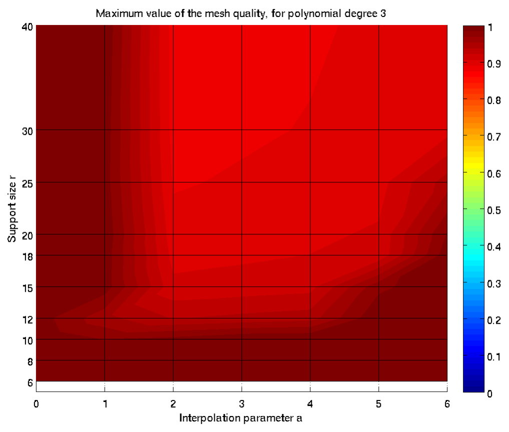
(a) 0-degree MLS.



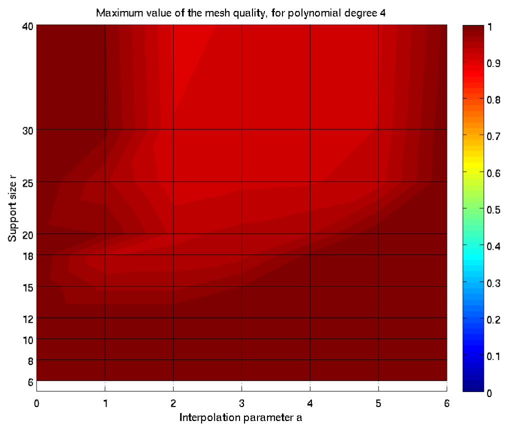
(b) 1st degree MLS.



(c) 2nd degree MLS.

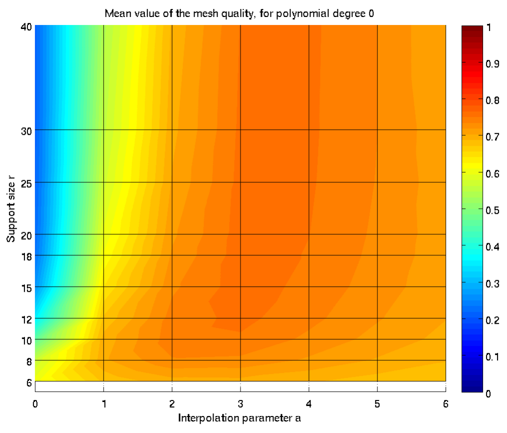


(d) 3rd degree MLS.

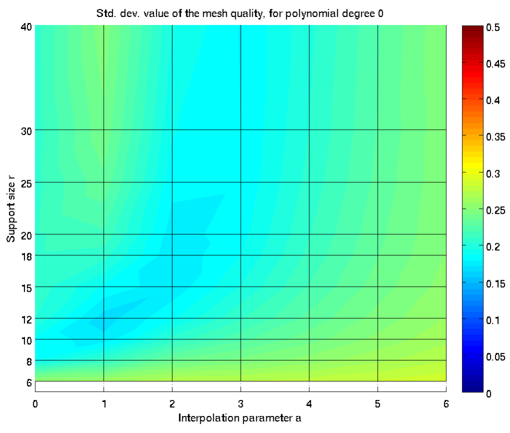


(e) 4th degree MLS.

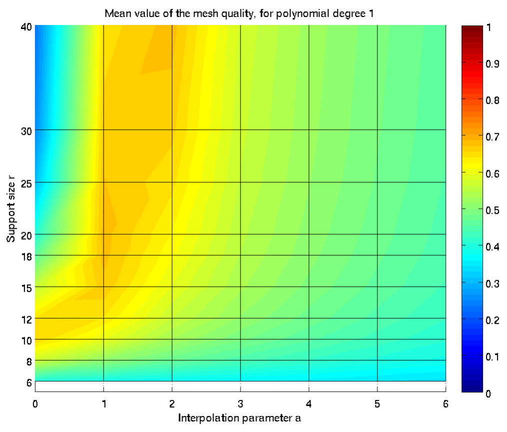
Figure C.31: Max quality surfaces of MLS adaptations, for rotation angle of 60deg in the benchmark problem. Each surface corresponds to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



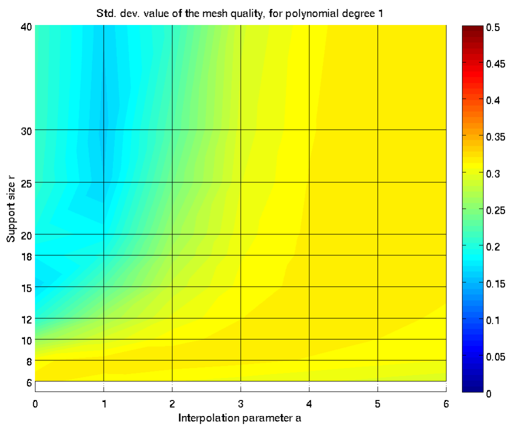
(a) 0-degree MLS, mean quality.



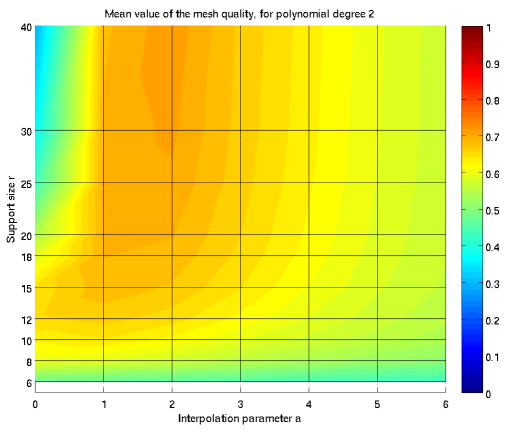
(b) 0-degree MLS, std. dev. of quality.



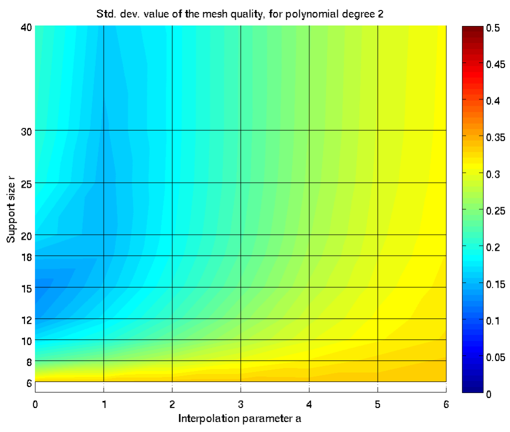
(c) 1^{st} degree MLS, mean quality.



(d) 1^{st} MLS, std. dev. of quality.

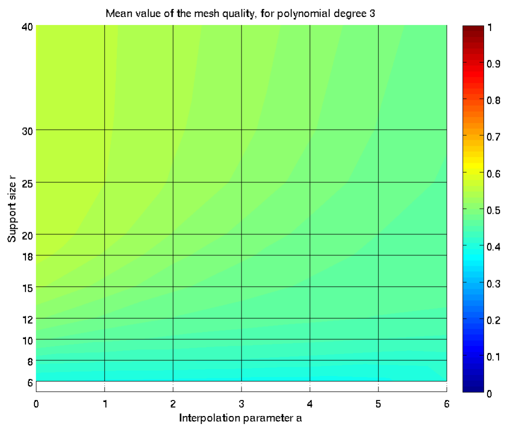


(e) 2^{nd} degree MLS, mean quality.

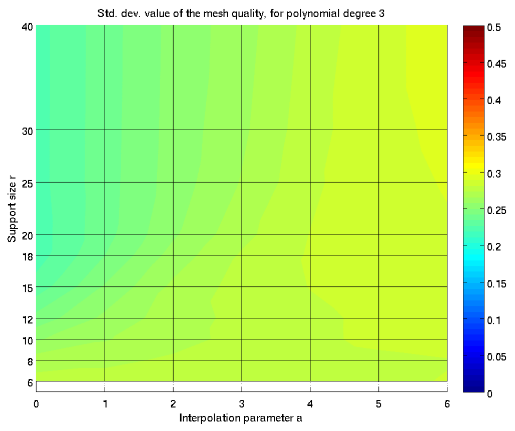


(f) 2^{nd} MLS, std. dev. of quality.

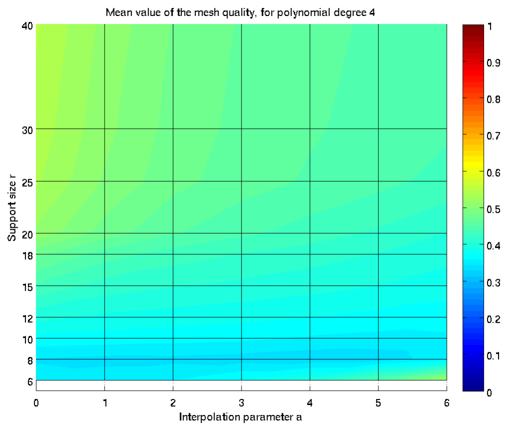
Figure C.32: Mesh quality surfaces, for a rotation angle of 80deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



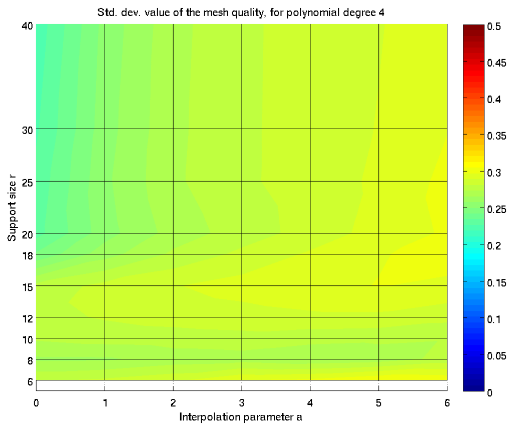
(g) 3rd degree MLS, mean quality.



(h) 3rd MLS, std. dev. of quality.

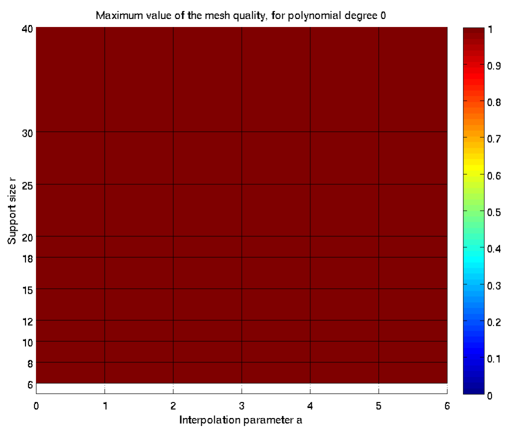


(i) 4th degree MLS, mean quality.

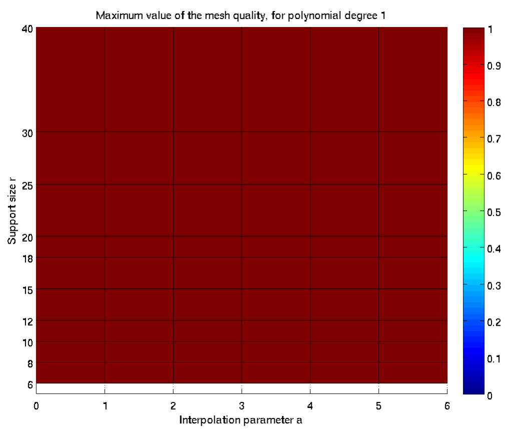


(j) 4th MLS, std. dev. of quality.

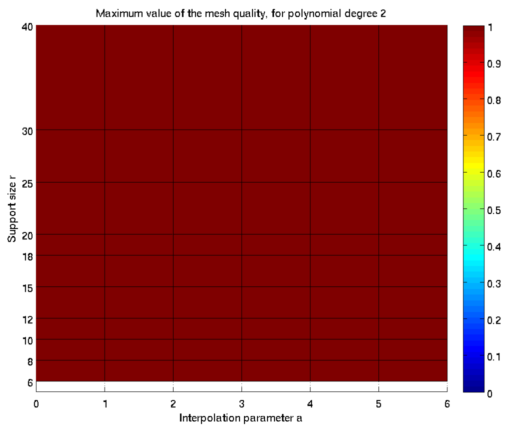
Figure C.32: Mesh quality surfaces, for a rotation angle of 80deg in the benchmark problem. Each pair in a row illustrates the mean & std.dev. of the quality, corresponding to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



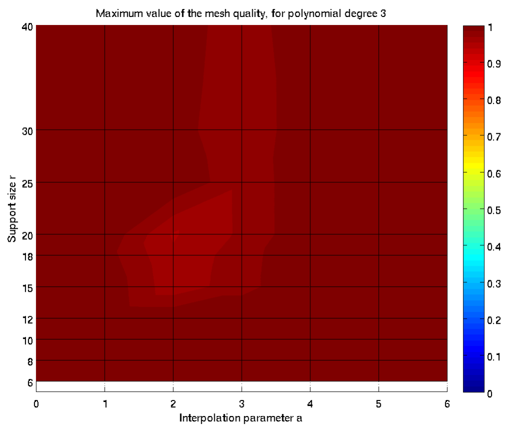
(a) 0-degree MLS.



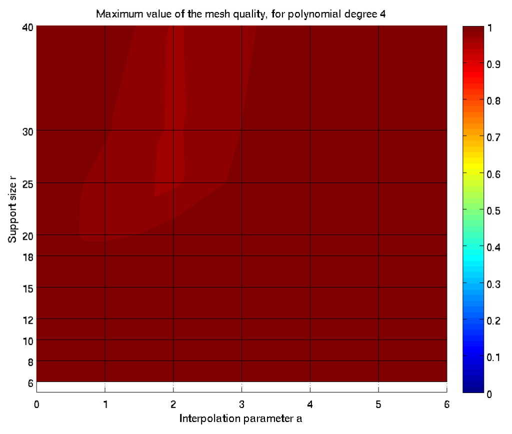
(b) 1st degree MLS.



(c) 2nd degree MLS.

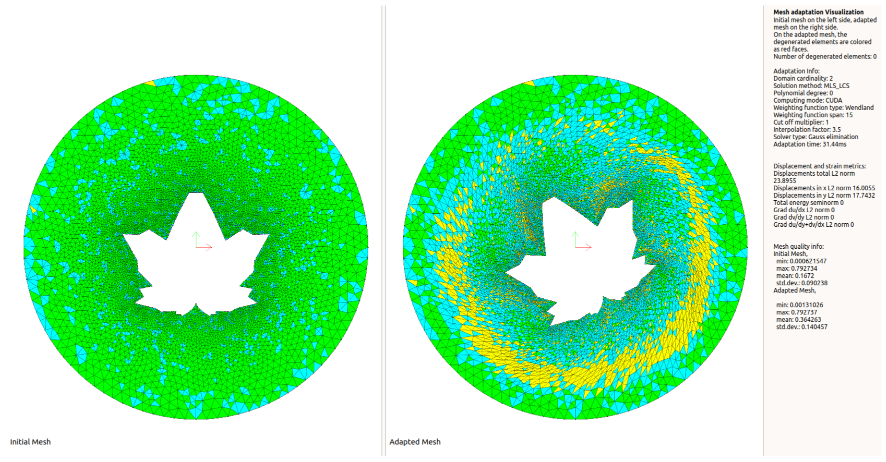


(d) 3rd degree MLS.



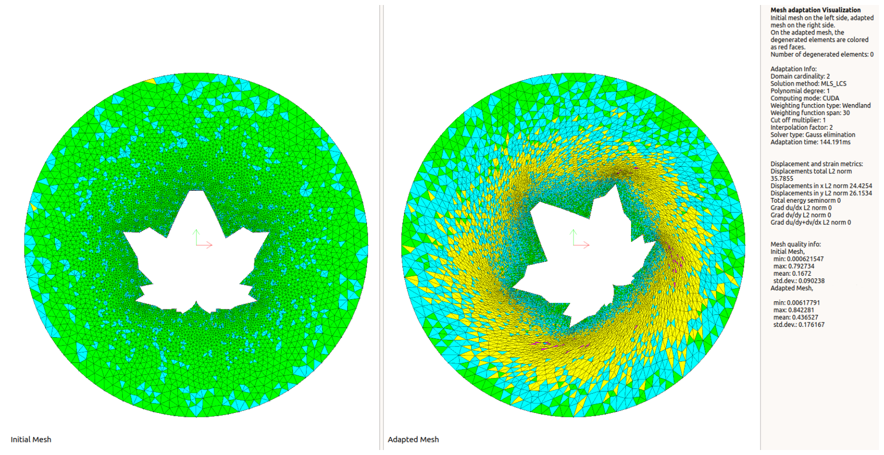
(e) 4th degree MLS.

Figure C.33: Max quality surfaces of MLS adaptations, for rotation angle of 80deg in the benchmark problem. Each surface corresponds to a polynomial degree, 0-4. The excellent quality is at zero level and the degenerated at the max value 1.



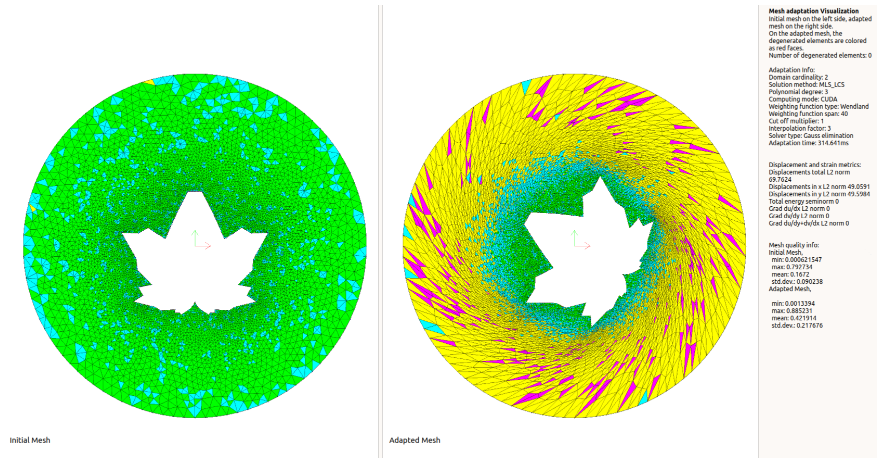
```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 0
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 15
Cut off multiplier: 1
Interpolation factor: 3.5
Solver type: Gauss elimination
Adaptation time: 31.44ms


Displacement and strain metrics:
Displacements total L2 norm
23.8955
Displacements in x L2 norm 16.0055
Displacements in y L2 norm 17.7432
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0


Mesh quality info:
Initial Mesh,
  min: 0.000621547
  max: 0.792734
  mean: 0.1672
  std.dev.: 0.090238
Adapted Mesh,

  min: 0.00131026
  max: 0.792737
  mean: 0.364263
  std.dev.: 0.140457
```

(a) The 0-degree MLS performing an acceptable solution with max metric value $maxQ < 0.8$, and a rotation of the interior boundary of 30deg.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 1
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 30
Cut off multiplier: 1
Interpolation factor: 2
Solver type: Gauss elimination
Adaptation time: 144.191ms


Displacement and strain metrics:
Displacements total L2 norm
35.7855
Displacements in x L2 norm 24.4254
Displacements in y L2 norm 26.1534
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0


Mesh quality info:
Initial Mesh,
  min: 0.000621547
  max: 0.792734
  mean: 0.1672
  std.dev.: 0.090238
Adapted Mesh,

  min: 0.00617791
  max: 0.842281
  mean: 0.436527
  std.dev.: 0.176167
```

(b) The 1st degree MLS performing an acceptable solution with max metric value $maxQ \approx 0.8$, and a rotation of the interior boundary of 45deg.



```
Mesh adaptation Visualization
Initial mesh on the left side, adapted
mesh on the right side.
On the adapted mesh, the
degenerated elements are colored
as red faces.
Number of degenerated elements: 0

Adaptation Info:
Domain cardinality: 2
Solution method: MLS_LCS
Polynomial degree: 3
Computing mode: CUDA
Weighting function type: Wendland
Weighting function span: 40
Cut off multiplier: 1
Interpolation factor: 3
Solver type: Gauss elimination
Adaptation time: 314.641ms


Displacement and strain metrics:
Displacements total L2 norm
69.7624
Displacements in x L2 norm 49.0591
Displacements in y L2 norm 49.5984
Total energy seminorm 0
Grad du/dx L2 norm 0
Grad dv/dy L2 norm 0
Grad du/dy+dv/dx L2 norm 0


Mesh quality info:
Initial Mesh,
  min: 0.000621547
  max: 0.792734
  mean: 0.1672
  std.dev.: 0.090238
Adapted Mesh,

  min: 0.0013394
  max: 0.885231
  mean: 0.421914
  std.dev.: 0.217676
```

(c) The 3rd degree MLS performing an acceptable solution with max metric value $maxQ \approx 0.8$, and a rotation of the interior boundary of 60deg.

Figure C.34: The illustration of MLS solutions, with acceptable mesh qualities. Three deformed meshes are presented for the lowest possible polynomial degree. The color indicates the quality of the mesh element according to the ranges of the table C.1, from better to worse: green, cyan, yellow (acceptable level), magenta, red, dark red.

Bibliography

Advanced Micro Devices. AMD Graphics Core Next Architecture, Generation 3, 2016.

K. Atkinson and W. Han. *Theoretical Numerical Analysis: A Functional Analysis Framework*. Texts in Applied Mathematics. Springer, 2009. ISBN 9781441904584. URL http://books.google.gr/books?id=Tg6p72yydEMC.

John T. Batina. Unsteady euler algorithm with unstructured dynamic mesh for complex-aircraft aerodynamic analysis. *AIAA Journal*, 29, 03 1991. doi: 10.2514/3.10583.

Gerrit Becker, Michael Schäfer, and Antony Jameson. An advanced nurbs fitting procedure for post-processing of grid-based shape optimizations. *49th AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition*, 01 2011. doi: 10.2514/6.2011-891.

T. Belytschko, Y. Y. Lu, and L. Gu. Element-free galerkin methods. *International Journal for Numerical Methods in Engineering*, 37(2):229–256, 1994. doi: 10.1002/nme.1620370205. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620370205.

L.P Bos and K Salkauskas. Moving least-squares are backus-gilbert optimal. *Journal of Approximation Theory*, 59(3):267 – 275, 1989. ISSN 0021-9045. doi: https://doi.org/10.1016/0021-9045(89)90090-7. URL http://www.sciencedirect.com/science/article/pii/0021904589900907.

Clarence Burg. Analytic study of 2d and 3d grid motion using modified laplacian. *International Journal for Numerical Methods in Fluids*, 52:163 – 197, 09 2006. doi: 10.1002/fld.1173.

Lennart Carleson. On bernstein's approximation problem. *Proceedings of the American Mathematical Society*, 2(6):953–961, 1951. ISSN 00029939, 10886826. URL http://www.jstor.org/stable/2031715.

Henri Casanova, Arnaud Legrand, and Yves Robert. *Parallel Algorithms*. Chapman & Hall/CRC, 1st edition, 2008. ISBN 9781584889458.

J. Cheng, M. Grossman, and T. McKercher. *Professional CUDA C Programming*. EBL-Schweitzer. Wiley, 2014. ISBN 9781118739327. URL https://books.google.lu/books?id=q3DvBQAAQBAJ.

William S. Cleveland. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association*, 74(368):829–836, 1979. ISSN 01621459. URL http://www.jstor.org/stable/2286407.

William S. Cleveland and Susan J. Devlin. Locally weighted regression: An approach to regression analysis by local fitting. *Journal of the American Statistical Association*, 83 (403):596–610, 1988. ISSN 01621459. URL http://www.jstor.org/stable/2289282.

P.J. Davis. *Interpolation and Approximation*. Dover Books on Mathematics. Dover Publications, 1975. ISBN 9780486624952. URL https://books.google.lu/books?id=2PaJAwAAQBAJ.

A. de Boer, M.S. van der Schoot, and H. Bijl. Mesh deformation based on radial basis function interpolation. *Computers & Structures*, 85(11):784 – 795, 2007. ISSN 0045-7949. doi: https://doi.org/10.1016/j.compstruc.2007.01.013. URL http://www.sciencedirect.com/science/article/pii/S0045794907000223. Fourth MIT Conference on Computational Fluid and Solid Mechanics.

Christoph Degand and Charbel Farhat. A three-dimensional torsional spring analogy method for unstructured dynamic meshes. *Computers & Structures*, 80(3):305 – 316, 2002. ISSN 0045-7949. doi: https://doi.org/10.1016/S0045-7949(02)00002-0. URL http://www.sciencedirect.com/science/article/pii/S0045794902000020.

J. Donea, S. Giuliani, and J.P. Halleux. An arbitrary lagrangian-eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33(1):689 – 723, 1982. ISSN 0045-7825. doi: https://doi.org/10.1016/0045-7825(82)90128-1. URL http://www.sciencedirect.com/science/article/pii/0045782582901281.

C. Farhat, C. Degand, B. Koobus, and M. Lesoinne. Torsional springs for two-dimensional dynamic unstructured fluid meshes. *Computer Methods in Applied Mechanics and Engineering*, 163(1):231 – 245, 1998. ISSN 0045-7825. doi: https://doi.org/10.1016/S0045-7825(98)00016-4. URL http://www.sciencedirect.com/science/article/pii/S0045782598000164.

M Fenn and G Steidl. Robust local approximation of scatterred data. 31:317–334, 01 2006.

B.A. Finlayson. *The Method of Weighted Residuals and Variational Principles: With Application in Fluid Mechanics, Heat and Mass Transfer*. Educational Psychology. Academic Press, 1972. ISBN 9780122570506. URL http://books.google.gr/books?id=KHHVNESp5UoC.

B.A Finlayson and L.E. Scriven. The method of weighted residuals - a review. *Applied Mechanics Reviews*, 19, 1966.

W. J. Gordon and J. A. Wixom. Shepard's method of "Metric Interpolation" to bivariate and multivariate interpolation. *Mathematics of Computation*, 32:253–264, 1978.

M Harris. Optimizing parallel reduction in cuda. 21:104–110, 01 2007.

Brian Helenbrook. Mesh deformation using the biharmonic operator. *International Journal for Numerical Methods in Engineering*, 56:1007 – 1021, 02 2003. doi: 10.1002/nme.595.

G. Dan Hutcheson. Moore's law, lithography, and how optics drive the semiconductor industry, 2018. URL https://doi.org/10.1117/12.2308299.

Frederic J. Blom. Considerations on the spring analogy. *International Journal for Numerical Methods in Fluids*, 32:647 – 668, 03 2000. doi: 10.1002/(SICI)1097-0363(20000330)32:6⟨647::AID-FLD979⟩3.0.CO;2-K.

Antony Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3, 12 1988. doi: 10.1007/BF01061285.

Antony Jameson. Optimum aerodynamic design using cfd and control theory. *CFD Review*, 3, 06 1995. doi: 10.2514/6.1995-1729.

Georgios K. Karpouzas, Evangelos M. Papoutsis-Kiachagias, Thomas Schumacher, Eugene de Villiers, Kyriakos C. Giannakoglou, and Carsten Othmer. Adjoint optimization for vehicle external aerodynamics. *International Journal of Automotive Engineering*, 7(1): 1–7, 2016. doi: 10.20485/jsaeijae.7.1_1.

David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2nd edition, 2012. ISBN 0124159923, 9780124159921.

Peter Lancaster and K Salkauskas. Surface generated by moving least square methods. 37: 141–141, 07 1981.

David Levin. The approximation power of moving least-squares. *Math. Comput.*, 67(224): 1517–1531, October 1998. ISSN 0025-5718. doi: 10.1090/S0025-5718-98-00974-0. URL http://dx.doi.org/10.1090/S0025-5718-98-00974-0.

H. Li and S.S. Mulay. *Meshless Methods and Their Numerical Properties*. Taylor & Francis, 2013. ISBN 9781466517462. URL https://books.google.lu/books?id=WFFwGJdRHrUC.

E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym. Nvidia tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, March 2008. ISSN 0272-1732. doi: 10.1109/MM.2008.31.

Erik Lindholm, Mark J. Kilgard, and Henry Moreton. A user-programmable vertex engine. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 149–158, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. doi: 10.1145/383259.383274. URL http://doi.acm.org/10.1145/383259.383274.

G.R. Liu. *Mesh Free Methods: Moving Beyond the Finite Element Method*. CRC Press, 2002. ISBN 9781420040586. URL https://books.google.lu/books?id=61rMBQAAQBAJ.

Wing-Kam Liu, Shaofan Li, and Ted Belytschko. Moving least-square reproducing kernel methods (i) methodology and convergence. *Computer Methods in Applied Mechanics and Engineering*, 143(1):113 – 154, 1997. ISSN 0045-7825. doi: https://doi.org/10.1016/S0045-7825(96)01132-2. URL http://www.sciencedirect.com/science/article/pii/S0045782596011322.

Xueqiang Liu, Ning Qin, and Hao Xia. Fast dynamic grid deformation based on delaunay graph mapping. *J. Comput. Phys.*, 211(2):405–423, January 2006. ISSN 0021-9991. doi: 10.1016/j.jcp.2005.05.025. URL http://dx.doi.org/10.1016/j.jcp.2005.05.025.

D.R. Lynch and K. O'Neil. Elastic grid deformation for moving boundary problems in two space dimensions. *3rd International Conference on Finite Elements in Water Resources*, 2, 1980.

Gia G. Maisuradze, Donald L. Thompson, Albert F. Wagner, and Michael Minkoff. Interpolating moving least-squares methods for fitting potential energy surfaces: Detailed analysis of one-dimensional applications. *The Journal of Chemical Physics*, 119(19):10002–10014, 2003. doi: 10.1063/1.1617271. URL https://doi.org/10.1063/1.1617271.

D. H. McLain. Drawing contours from arbitrary data points. *The Computer Journal*, 17 (4):318–324, 1974. doi: 10.1093/comjnl/17.4.318. URL http://dx.doi.org/10.1093/comjnl/17.4.318.

H.N. Mhaskar. Weighted polynomial approximation. *Journal of Approximation Theory*, 46 (1):100 – 110, 1986. ISSN 0021-9045. doi: https://doi.org/10.1016/0021-9045(86)90089-4. URL http://www.sciencedirect.com/science/article/pii/0021904586900894.

H.N. Mhaskar. *Introduction to the Theory of Weighted Polynomial Approximation*. Series in approximations and decompositions. World Scientific, 1996. ISBN 9789810213121. URL https://books.google.lu/books?id=pQ8ET7hx_RMC.

Davoud Mirzaei. Analysis of moving least squares approximation revisited. *J. Comput. Appl. Math.*, 282(C):237–250, July 2015. ISSN 0377-0427. doi: 10.1016/j.cam.2015.01.007. URL http://dx.doi.org/10.1016/j.cam.2015.01.007.

J Montrym and Henry Moreton. The geforce 6800. *Micro, IEEE*, 25:41 – 51, 04 2005. doi: 10.1109/MM.2005.37.

B Nayroles, Gilbert Touzot, and Pierre Villon. Generalizing the finite element method: Diffuse approximation and diffuse elements. 10:307–318, 09 1992.

Andrew Nealen. An As-Short-As-Possible Introduction to the Least Squares, Weighted Least Squares and Moving Least Squares Methods for Scattered Data Approximation and Interpolation, 2004.

NVIDIA Corporation. NVIDIA GeForce 8800 GPU Architecture Overview, 2006. Tesla G80 Architecture.

NVIDIA Corporation. NVIDIA GeForce GTX 200 GPU Architecture Overview, 2008. Tesla GT200 Architecture.

NVIDIA Corporation. Whitepaper CUDA Compute Architecture: Fermi, 2009.

NVIDIA Corporation. Whitepaper NVIDIA Fermi GF100 Architecture, 2010.

NVIDIA Corporation. NVIDIA GeForce GTX 680 GPU Technology Overview, 2012. Kepler GK104 Architecture.

NVIDIA Corporation. Whitepaper CUDA Compute Architecture: Kepler GK110/210, 2014a.

NVIDIA Corporation. Whitepaper NVIDIA GeForce GTX 750 Ti, 2014b. Maxwell GM107 Architecture.

NVIDIA Corporation. Whitepaper NVIDIA Tesla P100, 2016. Pascal GP100 Architecture.

NVIDIA Corporation. Whitepaper NVIDIA Tesla V100 GPU Architecture, 2017. Volta GV100 Architecture.

NVIDIA Corporation. CUDA C Best Practices Guide, 2018a. Version 10.0.

NVIDIA Corporation. Whitepaper NVIDIA Turing GPU Architecture, 2018b. Turing TU102 Architecture.

NVIDIA Corporation. NVIDIA CUDA C programming guide, 2018c. Version 10.0.

David Padua. *Encyclopedia of Parallel Computing*. Springer Publishing Company, Incorporated, 2011. ISBN 0387097651, 9780387097657.

David A. Patterson and John L. Hennessy. *Computer Organization and Design, Fifth Edition: The Hardware/Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition, 2013. ISBN 0124077269, 9780124077263.

Peter N. Glaskowsky. NVIDIA's Fermi: The first Complete GPU Computing Architecture, 2009. A whitepaper for NVIDIA Corporation.

Philippe P. Pébay and Timothy J. Baker. Analysis of triangle quality measures. *Mathematics of Computation*, 72(244):1817–1839, 2003. ISSN 00255718, 10886842. URL http://www.jstor.org/stable/4100021.

Giuseppe Quaranta, Pierangelo Masarati, and Paolo Mantegazza. A conservative mesh-free approach for fluid structure problems. 01 2005.

V. Rajaraman and C. Siva Ram Murthy. *Parallel Computers: Architecture and Programming*. Prentice-Hall of India Pvt.Ltd, 2nd edition, 2016. ISBN 8120352629, 9788120352629.

Ryozi Sakai. A study of weighted polynomial approximations with several variables (i). *Applied Mathematics*, Vol.08No.09:41, 2017. doi: 10.4236/am.2017.89095. URL //www.scirp.org/journal/PaperInformation.aspx?PaperID=79144.

Scott Schaefer, Travis McPhail, and Joe Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, July 2006. ISSN 0730-0301. doi: 10.1145/1141911.1141920. URL http://doi.acm.org/10.1145/1141911.1141920.

Thomas Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. volume 20, pages 151–160, 08 1986. doi: 10.1145/15886.15903.

Mohamed M. Selim and Roy P. Koomullil. Mesh deformation approaches - a survey. *Journal of Physical Mathematics*, 7(2):–, 2016. ISSN 2090-0902. doi: 10.4172/2090-0902.1000181. URL https://www.omicsonline.org/open-access/mesh-deformation-approaches--a-survey-2090-0902-1000181.php?aid=76056.

Shuvam Sen, Guillaume De Nayer, and Michael Breuer. A fast and robust hybrid method for block-structured mesh deformation with emphasis on fsi-les applications. *International Journal for Numerical Methods in Engineering*, 111(3):273–300, 2017. doi: 10.1002/nme.5465. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.5465.

Donald Shepard. A two-dimensional interpolation function for irregularly-spaced data. In *Proceedings of the 1968 23rd ACM National Conference*, ACM '68, pages 517–524, New York, NY, USA, 1968. ACM. doi: 10.1145/800186.810616. URL http://doi.acm.org/10.1145/800186.810616.

Stuart R. Slattery. Mesh-free data transfer algorithms for partitioned multiphysics problems. *J. Comput. Phys.*, 307(C):164–188, February 2016. ISSN 0021-9991. doi: 10.1016/j.jcp.2015.11.055. URL https://doi.org/10.1016/j.jcp.2015.11.055.

K Stein, Tayfun Tezduyar, and R Benney. Mesh moving techniques for fluid-structure interactions with large displacements. *Journal of Applied Mechanics*, 70:58–63, 01 2003. doi: 10.1115/1.1530635.

Tilo Strutz. *Data Fitting and Uncertainty: A Practical Introduction to Weighted Least Squares and Beyond*. Vieweg and Teubner, Germany, 2010. ISBN 3834810223, 9783834810229.

Cassiano Tecchio, Edson Basso, Joao Luiz Azevedo, and Diogo Pio. Mesh improvement for multiblock grids in store separation problems. 08 2014.

Laura Uyttersprot. *Inverse Distance Weighting Mesh Deformation, A Robust and Efficient Method for Unstructured Meshes*. Master's Thesis. Delft University of Technology, 2014.

Julien Vanharen, Rémi Feuillet, and Frédéric Alauzet. Mesh adaptation for fluid-structure interaction problems. 06 2018. doi: 10.2514/6.2018-3244.

Vasily Volkov and James W. Demmel. Benchmarking gpus to tune dense linear algebra. In *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, SC '08, pages 31:1–31:11, Piscataway, NJ, USA, 2008. IEEE Press. ISBN 978-1-4244-2835-9. URL http://dl.acm.org/citation.cfm?id=1413370.1413402.

H. Wendland. *Konstruktion und Untersuchung radialer Basisfunktionen mit kompaktem Träger*. 1996. URL https://books.google.gr/books?id=4AG5HAAACAAJ.

Jeroen Witteveen and Hester Bijl. Explicit mesh deformation using inverse distance weighting interpolation. 06 2009. doi: 10.2514/6.2009-3996.

Xingchen Zhang, Rejish Jesudasan, and Jens-Dominik Mueller. *Adjoint-Based Aerodynamic Optimisation of Wing Shape Using Non-uniform Rational B-Splines*, pages 143–158. 09 2019. doi: 10.1007/978-3-319-89890-2_10.

Yong Zhao and Ahmed Forhad. A general method for simulation of fluid flows with moving and compliant boundaries on unstructured grids. *Computer Methods in Applied Mechanics and Engineering*, 192(39):4439 – 4466, 2003. ISSN 0045-7825. doi: https://doi.org/10.1016/S0045-7825(03)00424-9. URL http://www.sciencedirect.com/science/article/pii/S0045782503004249.

Πέτρος Αποστόλου. *Μετακίνηση-Προσαρμογή 2Δ και 3Δ Μη-Δομημένων Πλεγμάτων με την Τεχνική των Στρεπτικών Ελατηρίων*. Master's Thesis. National Technical University of Athens, 2015.

Κ. Χ. Γιαννάκογλου. *Γένεση και προσαρμογή αριθμητικών πλεγμάτων*. ΕΜΠ, Υπολογιστική μηχανική, Αθήνα, 1999.

Χρήστος Ζέρβας. *Παραμετροποίηση Μορφών και Προσαρμοστική Παραμόρφωση 3Δ Υπολογιστικών Πλεγμάτων με χρήση Αρμονικών Συντεταγμένων. Εφαρμογή στην Αεροδυναμική Βελτιστοποίηση*. Master's Thesis. National Technical University of Athens, 2018.

Κωνσταντίνος Κολοβός. *Προγραμματισμός μεθόδου συναρτήσεων ακτινικής βάσης (RBF) για τη μετατόπιση υπολογιστικών πλεγμάτων σε κάρτες γραφικών*. Master's Thesis. National Technical University of Athens, 2015.

Αθανάσιος Κοντός. *Προσαρμοστική Παραμόρφωση 2Δ/3Δ Δομημένων και Μη-Δομημένων Πλεγμάτων με το Πρότυπο Κίνησης του Απαραμόρφωτου Σώματος*. Master's Thesis. National Technical University of Athens, 2018.

Αθανάσιος Λιατσικούρας. *Προγραμματισμός Μεθόδου Παραμόρφωσης Πλέγματος με Συναρτήσεις Ακτινικής Βάσης και Προσταθεροποιητή για χρήση στην Αεροδυναμική Βελτιστοποίηση*. Master's Thesis. National Technical University of Athens, 2015.

Μαρία Μαυρονικόλα. *Παραμετροποίηση Μορφών και Προσαρμογή Υπολογιστικών Πλεγμάτων με χρήση Αρμονικών Συντεταγμένων. Εφαρμογή στην Αεροδυναμική Βελτιστοποίηση Μορφής 2Δ Πτερυγώσεων και Αγωγών.* Master's Thesis. National Technical University of Athens, 2017.

Σ. Νεγρεπόντης, Θ. Ζαχαριάδης, Ν. Καλαμίδας, and Β. Φαρμάκη. *Γενική Τοπολογία και Συναρτησιακή Ανάλυση.* Εκδόσεις Συμμετρία, 1997. ISBN 978-960-266-178-9.

Δημήτριος Παπαδημητράκης. *Βελτιστοποίηση μορφής 2Δ πτερυγώσεων με εκειδικευμένη μορφοποίηση βασισμένη σε καμπύλες και επιφάνειες NURBS.* Master's Thesis. National Technical University of Athens, 2016.

Ιωάννης Τουρής. *Προσαρμογή 2Δ και 3Δ πλεγμάτων σε μεταβαλλόμενα όρια με τη μέθοδο των κινούμενων ελαχίστων τετραγώνων (MLS).* Master's Thesis. National Technical University of Athens, 2016.

Αλέξανδρος Γ. Τσολοβίκος. *Προσαρμογή Υπολογιστικών Πλεγμάτων με Χρήση Γράφων Delaunay - Εφαρμογές στη Βελτιστοποίηση με Χρήση της Συζυγούς Μεθόδου.* Master's Thesis. National Technical University of Athens, 2018.