Computational Science and Engineering
(Int. Master's Program)

Technische Universität München

Master's Thesis

Towards A Spatial Adaptive Combination Technique

Mahyar Valizadeh

Computational Science and Engineering (Int. Master's Program)

Technische Universität München

Master's Thesis

Towards A Spatial Adaptive Combination Technique

Author:	Mahyar Valizadeh
1st examiner:	Univ.-Prof. Dr. Hans-Joachim Bungartz
2nd examiner:	Univ.-Prof. Dr. Thomas Huckle
Advisor:	Christoph Kowitz, M.Sc. (hons)
Assistant advisor:	Alfredo Parra Hinojosa, M.Sc.
Thesis handed in on:	September 15, 2016

Disclaimer

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

September 15, 2016 Mahyar Valizadeh

Acknowledgments

First and foremost, I would like to express my very profound gratitude to Univ.-Prof. Dr. Hans-Joachim Bungartz and Univ.-Prof. Dr. Thomas Huckle for the opportunity to do my master thesis under their supervision and their support along the way any time I needed it.
My deepest sense of appreciation and thankfulness is towards my exemplary advisor Christoph Kowitz for his unconditional patience and support during the process of my master thesis. Without his patience I would not have been able to accomplish my task. His vision and knowledge on the subject and consequently, his continuous feedback have made my assignment much more comprehensible and straightforward. I would also like to thank my assistant advisor Alfredo Parra Hinojosa for his extra effort to steer me in the right direction and help me gain the confidence and encourage me to work harder later in the process.
I would like to acknowledge the great and endless opportunities I have been given to during my studies, from the admission to the program till the exceptional understanding of the CSE Examination Board to give me additional time to finish my thesis. I would also like to thank all the individuals involved in this regard. I feel obliged to say I am gratefully indebted to all.
Finally, I must articulate my very profound gratitude to my parents for nurturing me with immeasurable support throughout my life and to my friends for their heartfelt contribution along the process of writing this thesis. This accomplishment could not have been successfully achieved with you.

Abstract

This new algorithmic concept is based on the independent solution of many problems with reduced size and their linear combination using combination technique[5]. This technique has nearly same optimal computational complexity as the conventional multigrid methods. The main advantages of this method is its robustness and less memory consumption. The natural characteristic parallelism of the combination method makes it prefectly suited for MIMD parallel computers, distributed processing on workstation networks and multicore processors.[5]

One of the main advantages of the combination technique in comparison to the standard full grid is that it requires significantly less grid points. Another advantage is seen in the simplicity of the combination concept its characteristic parallel structure and its fundamental property allowing the integration of legacy solvers for partial differential equations.[12]

One way of using sparse grids efficiently involves hierarchical, tree-like data structures and special algorithms for both the discretization and the solution. Since conventional solvers usually do not provide means for dealing with hierarchical data structures, they cannot be employed for solving problems on sparse grids. Thus, new algorithms and new codes have to be developed in order to compute solutions on sparse grids efficiently. The method that is implemented in this thesis employs the concept of non-linear propagation in the data structure. For this purpose, the thesis focuses on quad trees and related tree traversal techniques. The combination method that is used in the thesis aims to operate using fewer FLOPs per iteration than the conventional full grid approach. The combination technique used in this thesis differs from the other combination technique in a way that it projects all the sub-levels grids into a full grid and then combine them to find a solution. The problem of interpolation is also under investigation in the thesis as different interpolation can lead difference in accuracy of the result. For the simulations, four test functions are proposed for the validity of the proposed combination technique and hence are proof-checked. Also, simulations are done to test the local adaptive mesh refinement. Finally the the problem at hand, the interpolation and the adaptive combination technique are tested against a predefined error function similar to adaptive grid generation methods. Lastly, an error analysis is performed with respect to threshold value for error and with maximum

refinement level of trees. The implemented adaptive combination technique promising as the result as accuracy is similar or better as compared to the full grid method.

Contents

List of Figures

Outline of the Thesis

Part I: Introduction and Theory

CHAPTER 1: INTRODUCTION
This chapter presents a general overview of the thesis including the reason and motivation behind this research, and its main objectives.

CHAPTER 2: RELATED WORKS
As any scientific work requires, this chapter presents all the related works and gives comprehensive literature review. It can be seen how the research evolved to this point. All the material will be presented in the order of how progress has been done.

CHAPTER 3: METHODS
By methods in this chapter basically means underlying principles and theories required to achieve the goal. All presented materials will be shown how they are related to the work at hand. Three primary methods presented are idea of sparse grid method and its relation to hierarchical basis, combination technique used to give another representation for sparse grid methods and finally data structure of trees specifically quadtrees in two-dimensional case. Finally some general important points related to the current thesis will be discussed.

Part II: Implementation, Results and Conclusion

CHAPTER 4: MY IMPLEMENTATION
This chapter presents the general idea of combination technique including the spatial adaptivity and its implementation on this thesis. Lastly, in this chapter different schemes are described.

CHAPTER 5: RESULTS
Firstly in this chapter we verify the result of the main block used in the implementation and later we present the results from different schemes and effects of the two factors of adaptivity level will be studied.

CHAPTER 6: CONCLUSION
Similar to all scientific researches last chapter is dedicated to summary of what has been accomplished, what usage it can have and how to proceed from here for future related research and developments.

Part I.

Introduction and Theory

1. Introduction

This chapter presents a very general and broad overview of the current thesis. Firstly, we present how the author thinks about the method at hand, i.e. combination technique [7]. As it happens in every particular area of science, several researchers have basically done similar and closely related works, but they have articulated their methods differently and in result multiple names have been introduced. The name combination technique has been chosen based on the fact that multiple results on different grids have been combined and projected onto one final grid. Secondly, the reason and motivation behind this research will be presented and also marking the benefits of the method in use.
Clearly motivation is important only when we are transparent about the objectives and goals of the project. So, we state these goals in detail.

1.1. Motivation

In numerical or computational science and more specifically numerical mathematics efficient discretization procedures are of crucial importance for different types of problems we encounter in various applications. All of these applications have multiple tasks in common, for example definition of sets of points known as grid generation, computation and evaluation of function values on these point, interpolation of function to estimate a value at an arbitrary point, or integrating or differentiating functions for solving differential equations using different schemes.
In particular we are interested in interpolation problem in our research certainly because it is the baseline problem to prove the combination technique can even be used for other more advanced problems. Through extensive search and review of previous related works, it seems that the base of our method comes from classical Richardson extrapolation[13]. On the other hand, Russian mathematician Smolyak using the basic of Richardson extrapolation presented a method which he used for numerical integration. One can imagine this gave birth to a foundation of sparse grids [1]. The underlying idea of Richardson was to use of different discretizations with different resolution such that fine mesh approximations are fine-tuned recursively by approximations on coarser levels. This idea is really close to multigrid methods and uses the same hierarchical structure. Such strategies, where multiple distinct grids participate to define a combined result, are generally called extrapolation methods such as Classical Richardson extrapolation.
Classical Richardson extrapolation can be extended and generalized in many ways. If this

generalization is being done using mesh widths in different dimensions, coordinate directions as the major playing parameters it will lead to the so-called multivariate extrapolation[13]. Note that combination extrapolation can be interpreted as a special case of multivariate extrapolation [5, 8]. Even further, it has been shown that one case of combination extrapolation is exactly the combination technique proposed in [7]. The primary idea of combinaton technique is identical for multilevel splitting of finite element spaces and it is to replace hierarchical bases of the finite element spaces instead of the usual nodal ones [3]. As explained earlier various names, such as (discrete) blending method [2], Boolean method [43], sparse grid method [4], technique of hyperbolic crosses[11], or splitting extrapolation [23] are practically interchangeable [19]. As the name is just a representation for the idea, we stick to the combination technique introduced by Zenger et. al in [7]
The advantages of this method discussed extensively in literature are as follows:

1. **Reducing the number of grid points:** The combination technique borrows this characteristic from the properties of sparse grids. In comparison to the full grid approach, the number of grid points (unknowns) can be drastically reduced.

2. **Inherently parallelizable:** Since the grids to be combined are usually independent of each other, it makes it easy to imagine how it can be parallelized using MIMD[6], Network[5] or GPGPUs[32]. The coarse grain parallelism of the combination method makes it perfectly suited for MIMD parallel computers and distributed system on workstation networks. Parallelization of combination technique supports both modularity and portability by separation of sequential modules. The gain is expected to be even more dramatic for higher dimensions. However, the collection of the results is not trivial as we need a strategy like trees to combine the solutions.

3. **Simplicity of the concept:** its framework allows the usage and integration of existing solvers and methods[12].

4. **Good accuracy:** the combination technique usually doesn't require as much storage space and computing time as the usual full grid but achieves nearly the same accuracy[14].

5. **Robustness:** Later it will be shown how this method can be exploited to various problems under certain conditions. Specially compared to sparse grids since it doesn't involves hierarchical, tree-like data structures and special algorithms for both the discretization and the solution it can be used in conjunction with conventional solvers.

6. **Speed of convergence:** its speed of convergence does not depend on the regularity properties of the considered boundary value problem or the refinement resolution[3].

7. **Optimal computational complexity:** The computational complexity of combination technique is almost the same as the conventional multigrid methods but without their restrictions[3].

Given these many advantages we are going to proceed and further improve the combination technique towards spatial adaptivity. Note that only major draw back which can not be ignored here is the error and convergence analysis. Later we see for various applications, there are certain conditions needed to be ensured of convergence.

1.2. Goals

As you will observe, comprehensive literature review has been made on this subject including areas such as error analysis and convergence, solution in complicated domains, numerical integration, regression and data fitting problem, eigenvalue problems, parallelization schemes, data mining and machine learning, fluid mechanics, heat transfer and some stochastic or random partial differential equations, and finally adaptive sparse grid methods been presented throughout years. There hasn't been any attention to possibility of introducing a spatial adaptive combination technique. Perhaps, the reason is that it requires more complex data structure, projection and interpolation methods to achieve it. However in this research we will tackle this idea with a greedy but simple algorithm. Details of the algorithm is explained later in the implementation chapter.

2. Related Works

Comprehensive literature study will be presented here. The researches have been categorized in separate sections. By doing so, faster access to the different applications for a seeking scientist can be achieved. While each work is referenced, the significant points and conclusions of each research have been summarized here so that it can be a good preparation and start for deeper search.

2.1. Works on Complicated Domains

The combination technique is not restricted to the unit square. Successful treatment of problems on distorted quadrilaterals, triangles, polygonal boundaries domain have been investigated by Griebel. The solution of problems with a nonlinear operator and partial differential equations like the Stokes and Navier-Stokes equations have been presented. [6, 9]

It has been shown that to some extent the combination technique works even in the case of non-smooth solutions like complicated domains explained earlier but replacement of h_i^2 and h_j^2 by h_i^α , h_j^β with appropriate α and β in the given problem is required. Because of the properties of the combination technique, the major error terms still cancel each other this way. However, for the problems with severe singularities, the appropriate combination of adaptively refined grids is recommended[6]. This results that our research can also be applied to this type of problems.

2.2. Works on Error Analysis and Convergence

Although, the implementation of the combination technique seems trivial, general convergence of the method cannot be proved with usual standard arguments from finite element theory. Therefore, there have been many works which we present in chronological order to give insight on what has been done. Recent studies [39] use more general convergence scheme with definition of error bounds but still there are plenty of assumptions there. Convergence and error analysis of the combination technique for the finite element solution of Poisson's equation and 2nd-order elliptic differential equations which is general form of Poisson equation has been investigated in early works and it has been the base for error

analysis of the combination technique.[10, 16]

Bungartz et al have investigated a model problem of Laplace equation on the unit square with a Dirichlet boundary function based on finite difference and Fourier techniques on a point-wise manner [12]. They proposed that there exists an error splitting if the Dirichlet boundary function satisfies a certain smoothness requirement i.e. Fourier coefficients in case of Laplace equation. Multiple numerical solutions for both smooth and non-smooth boundary functions have been studied in the convergence of the point-wise error.[12]

A technique to analyze the convergence rate of the combination technique applied to general second order elliptic differential equations in two dimensions and its proof for Poisson's equation convergences in arbitrary dimensions is later inspected by Pflaum [21]. The difference to his early work is the removal of requirement that the normal derivative of some coefficients should be zero at the boundary. The significant consequence is the proof of the convergence of the combination solution on a complicated domain or more precisely curvilinear bounded domain. The scheme is to divide the curvilinear bounded domain in several blocks and to transform each block onto the unit square. [21]

Further improvement has been done which is the base for the Introduction of optimized combination technique. Since we hardly know about the reasons of this effectiveness or divergence criteria of the combination technique. A technique which inherently uses the error terms has been introduced which is the so-called optimized combination technique. This is based on the fact that the combination technique gives an exact result in the case of a projection into a sparse grid space if their partial projections commute. They have analysed the performance of the combination technique in a projection framework and used the C/S decomposition. Based on that analysis, modified optimal combination coefficients are derived and substantially expand the applicability and performance of the combination technique.[30]

Most recently in the error analysis of combination technique, it has been shown that it requires derivation of a specific multivariate error expansions on Cartesian grids and for linear difference schemes through an error correction technique. By this, an error formulae will be derived to use for analysis of the convergence. Note that its dependence on dimension and smoothness in case of linear elliptic and parabolic problems has been on the focus. Finally, introduction of a new framework to analyze error bounds for general difference schemes in arbitrary dimensions is given. [39]

2.3. Works on Parallel Environment

Coarse grain parallelism, which basically is a kind of decomposition of tasks for the combination method makes it perfectly suited for MIMD parallel computers and distributed processing on workstation networks. There have been early studies parallel for the solution of elliptic partial differential equations on MIMD structured computers and parallel sparse grid preconditioning or solution of partial differential equations on Workstation networks [5, 6].

The concept of parallelization can be applied to different parts of solutions; for instance, the parallelization of the basic iterative method, the parallelization of the preconditioning step and so on. However, most types of preconditioners are not parallelizable that efficiently. They require modifications of the numerical algorithms resulting in slower convergence rates. A well parallelizable preconditioner using combination technique has been introduced by Griebel. They have compared their method with preconditioners inefficiently parallelizable and with preconditioner like classical multigrid which are well parallelizable but do not possess the natural parallel characteristic of the combination method. In contrast to these techniques, the parallelism in the combination technique is more explicit. Similar to the domain decomposition approach, the combination method possesses a simple parallelization potential, because all the sub-problems are independent [5].

Further parallel experiments on MIMD-machines and networks, however, have shown that it is insufficient to achieve substantially better efficiency rates for combination method. Therefore, the idea of complex or advanced load balancing strategy is necessary to exploit the benefits of massive parallel systems equipped with quick communication hardware [5].

Till that point of time, no comparison has been done with usual sparse grid methods, simply because the parallelization of an sparse grid code usually is non-trivial and requires a substantial effort on coding. Early works in this regard in comparison with combination technique are investigated by Zumbusch [23]. A parallel version of a finite difference discretization of partial differential on arbitrary, adaptively refined sparse grids is proposed. The efficient parallelisation is based on a dynamic load-balancing approach with space-filling curves. with applications can be in higher-dimensional problems such as financial engineering, in quantum physics, in statistical physics and in general relativity. Presented issue there is that usually hierarchies of refined grids in neighbour nodes may reside on different processors so it needs to be managed. One solution can be creation and updating of appropriate ghost nodes on a communication operation. The space-filling curve is simply a unique mapping of nodes to processors so it immediately shows which processor needs to be communicated with [23]. Relevantly, a load model for linear initial value runs with GENE is introduced for effective load balancing for the combination technique in [40].

Another interesting concept is the idea of using GPGPUs also known as multi-core programming. In multi dimensional option pricing problems of computational finance, the sparse grid combination technique can be a practical tool to solve arising PDEs. Using Hierarchization leads to linear systems smaller in size compared to standard finite element or finite difference discretization methods. Excessive demands of memory for direct methods which challenges the iterative methods suggests the usage of massive parallelism of general purpose Graphics Processing Units (GPGPU)s. It also requires proper data structures and efficient implementation of iterative solvers. Performance analysis and the scalability of combination technique based solvers on the NVIDIAs CUDA platform compared to CPUs for certain applications shows promising results because of locality and linearity properties.[32]

Advanced idea of hierarchization as preprocessing step to facilitate the communication needed for the combination technique has been presented in [36]. The derived Parallel hierarchization algorithm outperforms the baseline drastically and achieves good performance. The algorithm needs iterative hierarchization and dehierarchization. (For further details please check [36])

Lastly, in discussion of fault tolerance methods, it is known that extreme scale computing usually leads to the increase in probability of soft and hard faults. Parallel fault tolerant algorithms with modification of sparse grid combination method is in focus to solve partial differential equations in the presence of faults. It modifies combination formula to accommodate the loss of few component grids. A prototype implementation within a MapReduce framework using the dynamic process features and asynchronous message passing of MPI is presented.[38].

2.4. Works on Fluid Mechanics, Heat Transfer and other PDEs

Arguably, one of the most important application areas for combination technique can be in solution for partial differential equations with high dimension, high computation complexity or high memory demands regrading high order number of grid points. As first demonstration of the advantages of combination approach investigation on modal problem has been done in two dimensional cases [7]. Modal problems are as follows:

1. Smooth Solution
2. Singular Solution
3. Boundary layer (solution of combination technique is equal to full grid, in general holds for solution only depends on one direction or boundary layer)

4. Distorted quadrilateral and triangular elements

5. Nonlinear heat transfer

In case of nonlinear heat transfer, the combination technique produces good solutions. The error quotient is major factor to check here. It has been shown that the efficiency, the degrees of freedom, the run time and the achieved accuracy for the combination technique is far better than the full grid approach. Also note that additional Newton iterations seems to be a remedy the approach even further [7].

First studies of the sparse grid combination technique as an efficient method for the solution of fluid dynamics problems has been done in [14]. It shows the numerical experiments for the application of the combination method to CFD problems, e.g. two-dimensional laminar flow problems with moderate Reynolds numbers. The research is based on the fact that implementation of the combination technique can be based on any black-box solver. Usually, fluid dynamics problems have to be solved on rather complex domains; thus, a reasonable approach is to decompose the domain into blocks to handle the problem. Obviously, the combination technique works on such block structured domains as well as complicated domains like a graded grids. Since the sparse grid methods are highly economical on storage requirements, given that they produce a fairly accurate solution, the usage is highly recommended [14].

In [20], promising numerical results are presented for the combination technique applied to a constant coefficient advection equation for four test cases of. Their work differs from [20] in that it also presents error estimates [22].

1. Horizontal advection

2. Diagonal advection

3. Time dependent advection

4. The Molenkamp-Crowley test case

Similar to heat transfer problem explained above, in [24] a significant progress in the numerical simulation of systems of partial differential equations in the advection diffusion reaction equations of large-scale transport problems in the modelling of pollution of the atmosphere, surface water and ground water has been achieved. Since these models are three dimensional and modelling transport and chemical exchange over long time periods requires very efficient algorithms computational capacity is a restricting factor. For example in simulation of global air pollution, huge numbers of grid points is needed, each of which many calculations must be carried out in. The application of sparse grid combination techniques might be a solution. However, in their research, the authors only con-

sidered pure advection and left the diffusion and reaction processes for future research[24].

More recently, a convection diffusion problems on the conventional unit square has been investigated and observed using a sparse grid Galerkin finite element method in[31]. Lastly, application to probabilistic equations like the stochastic collocation method based is on the horizon. For instance, an anisotropic sparse grid solution is an important tool to solve partial differential equations with random input data[41].

2.5. Works on Numerical Integration

Based on original idea of Smolyak there has been some similar work in numerical integrations. For example, in [18], Gerstner and Griebel review and compare existing algorithms for the numerical integration with the the ones of multivariate functions over multi-dimensional cubes using several variants of the sparse grid method.[18, 1]

Multivariate integrals arise in many application fields, such as statistical mechanics, computational finance and discretization of partial differential and integral equations or the numerical computation of path integrals. The so-called curse of dimension are also in play in these conventional numerical algorithms for computation of integrals there. So the rectifying remedy of sparse grid combination method is used in [18].

2.6. Works on Regression Problem

In the context of regression or basically fitting the function to given values, there has been some works. In [25], Hegland compares the iterative algorithm for multidimensional sparse grid regression with penalty and showed improved performance compared to iterative methods based on the combination technique.
The combination technique approach shows instabilities in some situations and is not guaranteed to converge specially with higher discretization levels. As stated before, the optimized combination technique can repair these instabilities, based on the fact that combination coefficients also depend on the function to be reconstructed, thus a nonlinear solution. It has been shown that the computational complexity of the optimized method still scales in linear manner to the number of data.[27]

In [33], there has been investigation in theory and experiment of the reason why combination technique solution for regularized least squares fitting is not as effective as it is in the case of elliptic partial differential equations. Their argument is that this is due to the irregular and random data distribution, and dependency of number of data to the grid resolution. They note that overfitting can arise when the mesh size goes to zero. So they conclude an optimal combination coefficients can prevent the amplification of the sam-

pling noise present.

2.7. Works on Data Mining and Machine Learning

Data is currently produced with a huge rate. The issues in data processing sciences are mainly the increasing amount of data recorded and also the increasing complexity of these data. Application of analysis of image, multimedia, or spatial data are some examples of it.. An important task in data modelling is to develop prediction models or functional relationships between their selected features. The so-called curse of dimensionality arises when one wants to identify such predictive models. In [26], Hegland discusses, how to choose the function spaces with an iterative approach to increase complexity of functions. His approach is to use adaptive complexity management closely related to the Analysis of Variance, also known as ANOVA decomposition. As the sparse grid approximations is good framework in this regard, it has been combined with regression trees and multivariate splines to analyze the complexity of solution

Similar to the idea above, the generative dimensionality reduction methods in machine learning have been investigated in [42]. Bohn, Garcke and Griebel propose a framework to build a mapping from a lower dimensional space problem to higher dimensional data space and vice versa to achieve a dimension adaptive sparse grid reduction method. The reason to do so in data analysis is because some directions play more important role than the others and it is reasonable to refine the underlying discretization space only in these directions based on the value of reconstruction error in that dimension.

2.8. Works on Eigenvalue Problems

In computational physics, the concept of eigenvalue problem arises for example for the Born-Oppenheimer approximation of the stationary Schrödinger equation for atoms. A discrete eigenvalue problem is based on finite element discretization of the problem. In [29], Garcke proposes to use optimized combination technique for the solution of this problem but applying it directly to eigenvalue problems is not possible. The remedy is to use partial solutions as ansatz functions and reconstruct the projection of the optimized combination technique as a Galerkin-approach[29].

Similarly, in the context of hot fusion plasmas there is a five-dimensional gyro-kinetic equations. As five dimensional problem in plasma needs a lot grid points and it grows exponentially because of the high dimensionality, i.e. a factor of five. However, as shown

before, the combination technique can be applied to this eigen-value problem for the gyrokinetic code GENE[37].

2.9. Works on Adaptive Methods

This section is of major importance since it is closely related to the work of this thesis, which is also based on adaptive methods. First major study in this regard has been done by Griebel [19]. Griebel has investigated how to use standard operations between two functions, i.e. addition or subtraction, scalar multiplication, and division. This is done by transformation of values to nodal basis and accumulation in nodal basis and then returning to hierarchical basis representation. By this setup, we can estimate the error indicators and perform adaptive multilevel treatment of PDEs. First hand usage of hash table benefits in the case of adaptive multilevel treatment are also another important part of the research[19].

Another important research has been in the context of machine learning and regression by Garcke [28]. Note that this work has been categorized with adaptive methods, rather than in regression section simply because of the importance of their work in correspondence with the research at hand. It has been proposed how to use hierarchy of generalized sparse grids and choose the partial functions with adaptive iterative procedure. By doing so, one can pick out features insignificant to the prediction and thus the adaptivity.

3. Methods

This chapter discusses the mathematical techniques used in this thesis. The main theories discussed here are firstly, concepts of nodal and hierarchical basis for a function representation in multidimensional spaces and consequently based on hierarchical bases theories of sparse grids and combination techniques. Respectively, these main methods construct an image about mathematical reasoning behind methods used in this thesis.

3.1. Concept of nodal basis

First idea about nodal basis comes form the finite element method and it's splitting of function into a space. The function representation in that space can be done based on nodal basis which are set of functions existing in the space. For the mathematical representation of the nodal basis, the hierarchical basis and the sparse grids, work of Garcke [35] has been used. Following same notation, let's consider a multi-index $\underline{l} = (l_1, \ldots, l_d) \in \mathbb{N}^d$ and an anisotropic grid $\Omega_{\underline{l}}$ with mesh size $h_{\underline{l}} := (h_{l_1}, \ldots, h_{l_d}) = 2^{-l} := \left(2^{-l_1}, \ldots, 2^{-l_d}\right)$. $\Omega_{\underline{l}}$ contains equidistant but different h_{l_t} in coordinate direction $t, t = 1, \ldots, d$. Thus grid Ω consists of points

$$x_{l,j} := (x_{l_1,j_1}, \ldots, x_{l_d,j_d}) \tag{3.1}$$

where $x_{l_t,j_t} := j_t \cdot 2^{-l_t}$ and $j = 0, \ldots, 2^{l_t}$. An associated space $V_{\underline{l}}$ of piecewise n-linear function can be defined as

$$V_{\underline{l}} := span\left\{\phi_{\underline{l},\underline{j}} | j_t = 0, \ldots, 2^{l_t}, t = 1, \ldots, d\right\} = span\left\{\phi_{\underline{l},\underline{j}} | 0 \leq \underline{j} \leq 2^{\underline{l}}\right\} \tag{3.2}$$

$$\phi_{\underline{l},\underline{j}}(\underline{x}) := \prod_{t=1}^{d} \phi_{l_t,j_t}(x_t) \tag{3.3}$$

The one dimensional functions $\phi_{l,j}$ have support

$$[x_{l,j} - h_l, x_{l,j} + h_l] \cap [0,1] = [(j-1)h_l, (j+1)h_l] \cap [0,1] \tag{3.4}$$

These one dimensional functions are defined by :

$$\phi_{l,j}(x) = \begin{cases} 1 - |x/h_l - j|\,, & x \in [(j-1)h_l, (j+1)h_l] \cap [0,1] \\ 0, & \text{otherwise} \end{cases} \tag{3.5}$$

This representation is basically a representation on nodal basis.

3.2. Concept of hierarchical basis

The conversion of the function coefficients on a grid from standard nodal basis to the representation in hierarchical basis is carried out by a constant operation count per grid point. The transformation to the hierarchical basis representation is done by the algorithm described thoroughly in [7] which gives us further insight into the hierarchical basis. So, after defining the multi-indices, space, grids and one dimensional basis functions, one needs to define hierarchical difference space $W_{\underline{l}}$

$$W_{\underline{l}} := V_{\underline{l}} \setminus \bigoplus_{t=1}^{d} V_{\underline{l}} - \underline{e}_t \tag{3.6}$$

where $\underline{e}_t$ is the t-th unit vector. From the definition of the index set

$$\phi_{l,j}(x) = \begin{cases} j_t = 1, \ldots, 2^{l_t} - 1, & j_t \text{ odd }, t = 1, \ldots, d, \text{ if } l_t > 0 \\ j_t = 0, 1, , & t = 1, \ldots, d \text{ if } l_t = 0 \end{cases} \tag{3.7}$$

It leads to

$$W_{\underline{l}} = span \left\{ \phi_{\underline{l},\underline{j}} | \underline{j} \in \ \mathrm{B}_{\underline{l}} \right\} \tag{3.8}$$

The hierarchical difference spaces leads us to multilevel subspace decomposition. We can write the spaces as the direct sum of the subspaces

$$V_n := \bigoplus_{l_1=0}^{n} \cdots \bigoplus_{l_n=0}^{n} W_{\underline{l}} = \bigoplus_{|\underline{l}|_\infty \leq n} W_{\underline{l}} \tag{3.9}$$

where

$$|\underline{l}|_\infty := \ \max_{1 \leq t \leq n} l_t \text{ and } \ |\underline{l}|_1 := \sum_{t=1}^{n} l_t \tag{3.10}$$

The set of functions

$$\left\{ phi_{\underline{l},\underline{j}} | \underline{j} \in \ \mathrm{B}_{\underline{l}} \right\}_{\underline{l}=\underline{0}}^{\underline{n}} \tag{3.11}$$

is hierarchical basis of V_n. Therefore, every function $f \in V_n$ can be represented as

$$f(\underline{x}) = \sum_{|\underline{l}|_\infty \leq n} \sum_{|\underline{j}| \in \mathrm{B}_{\underline{l}}} \alpha_{\underline{l},\underline{j}} \cdot \phi_{\underline{l},\underline{j}}(x) = \sum_{|\underline{l}|_\infty \leq n} f_{\underline{l}}(\underline{x}), \quad \text{with } \ f_{\underline{l}} \in W_{\underline{l}} \tag{3.12}$$

where $\alpha_{\underline{l},\underline{j}} \in \mathbb{R}$ are the coefficients of the representation in the hierarchical tensor product basis.

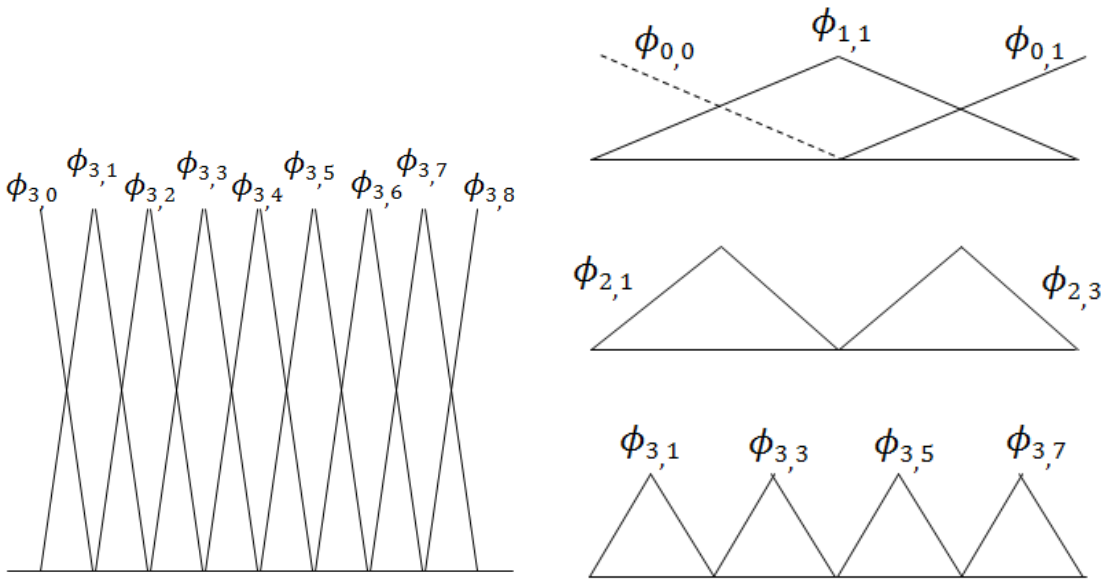
Furthermore,

$$V := \lim_{n \to \infty} \bigoplus_{|\underline{k}| \leq n} W_{\underline{k}} \tag{3.13}$$

The generalized d-dimensional hierarchization operator can be written as

$$\alpha_{\underline{l},\underline{j}} = \left(\prod_{t=1}^{n} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}_{l_t, j_t} \right) f \tag{3.14}$$

The hierarchical basis simplifies the calculations with the functions that are defined in different grids. As an example, one can simply add the single coefficients for calculation the addition of two functions that are defined the different grids, because basis functions associated with the same grid points are exactly same[7].
Note that, with a cursory glance, the hierarchical matrices seems complicated however one needs to factorize the matrices into the general nodal basis stiffness matrix and sparse matrices. It is not required to assemble the discretization matrix.[3]. Comparison of nodal and hierarchical basis for one dimensional level three has been shown in figure. 3.1.



(a) Nodal basis

(b) Hierarchical basis

Figure 3.1.: Nodal and hierarchical basis of level $n = 3$.

3.3. Sparse Grids based on hierarchical subspace splitting

Efficient discretization techniques are imperative for majority of problems in numerical mathematics such as interpolation, point approximation etc. Introduced by Zenger in [4] and based on hierarchical tensor product approximation spaces, sparse grids provide an efficient approach to improve the ratio of invested storage and computing time to the achieved accuracy.[17]

Following [4] finite element space is the direct sum of the subspaces where the error propagation is equal or larger than some prescribed tolerance. This leads to an approximation space that corresponds to a sparse grid in place of a full grid.
Hierarchical basis functions with small support and small contribution to function representation are not included in discrete space level n .This type of grids are classisified as sparse grids.

$$V_n^s := \bigoplus_{|\underline{l}|_1 \leq n} W_{\underline{l}} \tag{3.15}$$

Using (3.9), we get

$$f_n^s(\underline{x}) = \sum_{|\underline{l}|_1 \leq n} \sum_{\underline{j} \in B_{\underline{l}}} \alpha_{\underline{l},\underline{j}} \phi_{\underline{l},\underline{j}}(\underline{x}) = \sum_{|\underline{l}|_1 \leq n} f_{\underline{l}}(\underline{x}), \quad \text{with } f_{\underline{l}} \in W_{\underline{l}} \tag{3.16}$$

The sparse grids can also be formulated as follows:

$$V_{0,n}^s := \bigoplus_{|\underline{l}|_1 \leq n+d-1} W_{\underline{l}}, \quad l_t > 0 \tag{3.17}$$

The approximation error can defined as

$$\| f - f_{0,n}^s \| \leq \sum_{|\underline{l}|_1 \leq n+d-1} \| f_{\underline{l}} \| \tag{3.18}$$

where as the interpolation error is defined as

$$\| f - f_n^s \|_2 = \mathcal{O}\left(h_n^2 \log(h_n^{-1})^{d-1}\right) \tag{3.19}$$

A simple visual represention of result from a python script called "exahd-teachlet" given by adviosrs to the thesis resulted in figure 3.2.

3.4. Combination technique

To discretize a function f on particular combination of anisotropic grids $\Omega_{\underline{l}}$ with uniform mesh size $h_t = 2^{l_t}$ in the t-th coordinate direction. different grids have different mesh sizes for different directions. all grids can be considered with

$$|\underline{l}| := l_1 + ... + l_d = n - q, \;\; q = 0, ..., d-1, \;\; l_t \geq 0 \tag{3.20}$$

Using finite element approach with piece-wise linear function in each dimensions, one gets the nodal basis as discrete partial functions from different grids using the combination formula

$$f_n^c(\underline{x}) := \sum_{q=0}^{d-1} (-1)^q \begin{pmatrix} d-1 \\ q \end{pmatrix} \sum_{|\underline{l}|_1 = n-q} f_{\underline{l}}(\underline{x}) \tag{3.21}$$

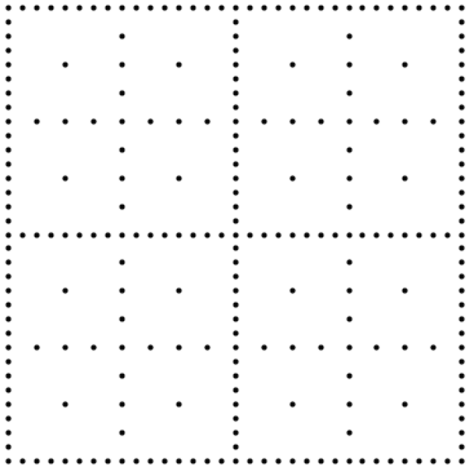


Figure 3.2.: Classical sparse grid for l=4

The function $f_n^c(\underline{x})$ is a part of the sparse grid scpace V_n .

$$|f - f_n^c| = \mathcal{O}\left(h_n^2 \cdot \log(h_n^{d-1})\right) \tag{3.22}$$

One can also consider ANOVA representation in form the combination method, where the function can be represented as

$$f(\underline{x}) = \sum_{\{j_1,\ldots,j_q\}\subset\{1,\ldots,d\}} c_{j_1,\ldots,j_q} f_{j_1,\ldots,j_q}\left(x_{j_1},\ldots,x_{j_q}\right) \tag{3.23}$$

where each f depends only depends only on a subset of size q of the dimensions. It is also possible that there exist different refinement levels for each dimension. If dimension q is significantly smaller than d, then the complexity is only dependent on q, which is also called superposition dimension. An advantage of such case is beneficial where one can choose grids of their own and not follow the grid sequence.. This is known as the dimension adaptive procedure. Here an index set I is expressed such that one needs to fulfill an admissibility condition

$$\underline{k} \in \mathrm{I} \text{ and } \underline{j} \leq \underline{k} \quad \Longrightarrow \quad \underline{j} \in \mathrm{I} \tag{3.24}$$

It means that index $\underline{k} \in I$ if all smaller grids belong to I. The combination coefficients for dimension adaptive combination technique depends only upon the index set I.

$$f_I^c(\underline{x}) := \sum_{\underline{k}\in I} \left(\sum_{\underline{z}=\underline{0}}^{1} (-1)^{|\underline{z}|1} \cdot \chi^I(\underline{k}+\underline{z}) \right) f_k(x) \tag{3.25}$$

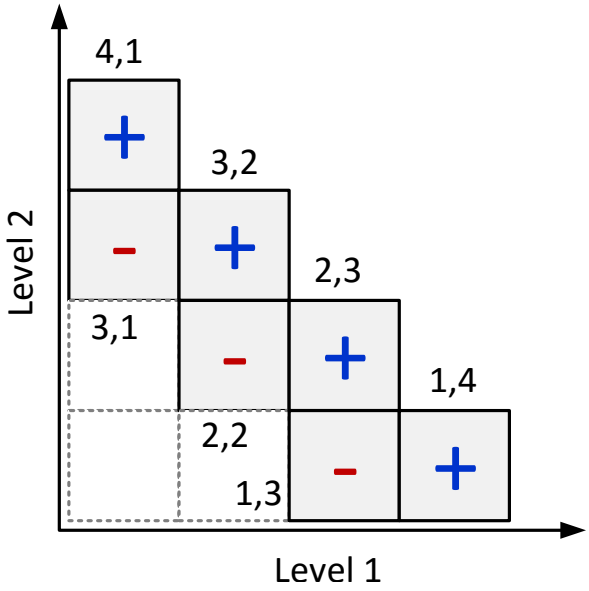


Figure 3.3.: Classical sparse grid combination method for l=4

where χ^I is the characteristic function of I and is expressed as follows

$$\chi^I(\underline{k}) = \begin{cases} 1 & \text{if } k \in I \\ 0 & otherwise \end{cases} \tag{3.26}$$

Another motivation for sparse grids is to represent any sparse grid function as a linear combination of its interpolants on the regular full grids. This requires us to seek a certain linear combination of full grid spaces to create a sparse grid space. This approach do away with the need for data structures more complicated than a few arrays to store a sparse grid function and requires only regular data structures.[7]
The crux of the combination technique is similar to the multilevel splitting of the finite element spaces which changes the nodal bases of the space with hierarchical bases. [3] The basis for combination techniques comes from the liner combination of independent solution of many problems with reduced size. [5]. Figure 3.3 shows an example of this multilevel combination which is basically addition and subtraction in this special case.

In total combination technique involves $O(h_n^{-1}log(h_n^{-1}))$ unknowns in contrast to $O(h_n^{-2})$ unknowns for the conventional full grid approach however combination solution is nearly as accurate as the standard solution. it can be proven that error is only slightly worse than for the associated full grid.[6, 7]. To some extent the combination technique even works for non-smooth solutions. Now, h_i^2 and h_j^2 in have to be replaced by h_i^α , h_j^β with appropriate

α and β, but because of the properties of the combination technique the leading error terms still cancel. However, for the problems with severe singularities, the appropriate combination of adaptively refined grids is recommended.[6]
It is seen that in some situations the combination technique leads to the cancellation of certain error terms in the asymptotic error expansion[7]
Lastly, one should note that the error analysis of this method has been relatively explained in [35] and the references there within. For further details please check the chapters of related works under "Works on Error Analysis and Convergence" presented earlier in the thesis. The discussion of these analysis is out of scope for this thesis but perhaps relatively important in future works.

Part II.

Implementation, Results and Conclusion

4. Current Implementation

Since the idea of main concept for the current implementation is based on trees, more specifically quad trees, we present the relevant general ideas of tree at first and explain how they are related to this implementation. The theory presented in this section is based on a book by Goodrich et al. unless otherwise stated[44].

4.1. General ideas about tree structure

Productivity experts claim that breakthroughs come by thinking ”nonlinearly”. Tree structures have been an enormous leap in data organization since they allow the implementation of algorithms more rapidly than linear data structures such as lists, vectors and sequences.

In categorizing trees as ”nonlinear” with respect to organizational relationship, the main feature that must be highlighted is more sophisticated relationships than elementary ”before” and ”after” relationships between objects in sequences. A tree follows a **hierarchical** architecture, with some objects being ”above” and some ”below” others.

A **tree** is an abstract data type. In a tree, each element has a **parent** element. It can have zero or more **children** elements.

Formal Tree Definition

A **tree T** can be defined as a set of **nodes** storing elements in a **parent-child** relationship and has the properties as defined below:

- A nonempty **T** has a special node known as the **root** of **T** which has no parent.
- Every other node ν of **T** has a unique parent node w and every node with parent w is a **child** of w.

If a parent has two nodes which are its children, then these children are **siblings**. If a node ν does not have children then it is **external** while it is **internal** if it has one or more children. External nodes can also be termed as **leaves**.

Figure 4.1 shows an example of a tree representing computer files organized in the form of nested directories.

The relation of this to our implementation is that the external nodes are the nodes which doesn’t require more refining. **Ordered Trees**

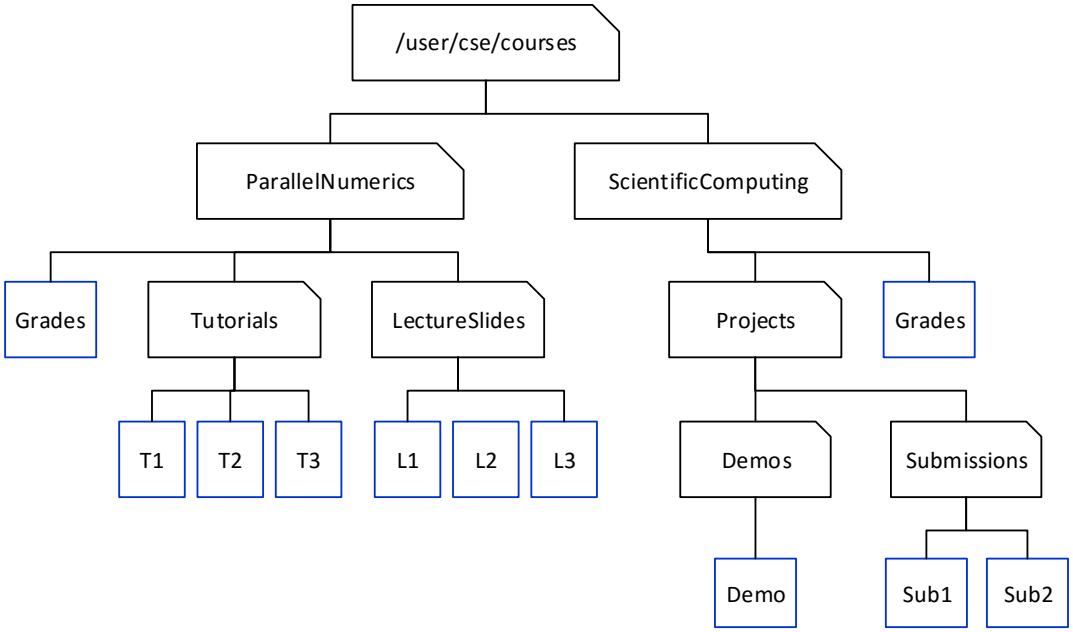


Figure 4.1.: Tree representing a portion of a file system

A tree having a linear ordering for the children of each node is termed as **ordered**. The children of a node can be identified as first, second and so on. In ordered trees, linear order relationship between siblings is illustrated by listing them in a sequence.

For implementation purposes ”position” and ”node” can be used interchangeably for trees.

Later it can be observed how the nodes are exactly a full grid solution or projection of combination technique solution on to full grid one.

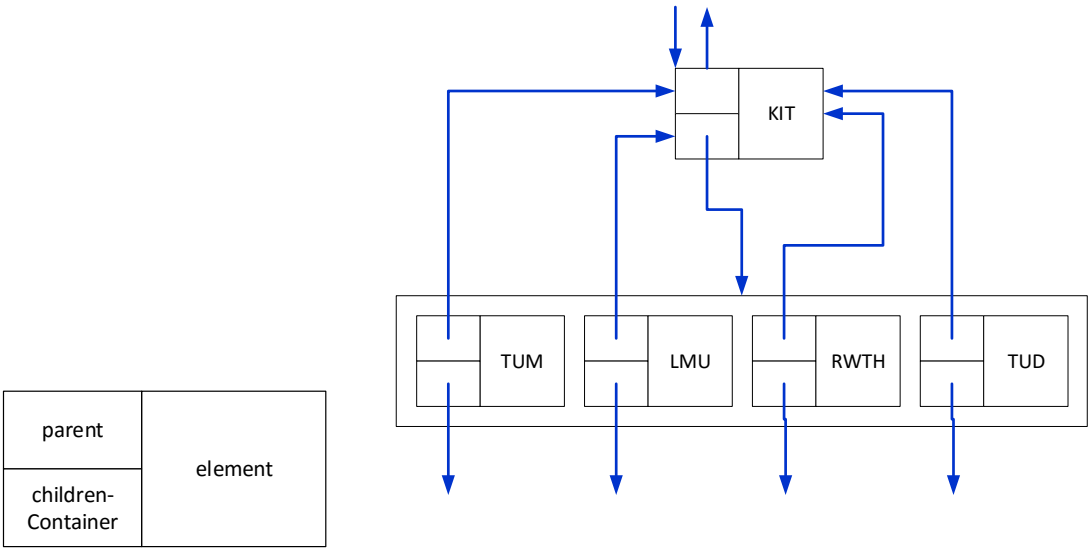
A Linked Structure for General Trees

A **linked structure** can be used to comprehend a tree **T**. Each node of **T** can be depicted as a position object p with the below mentioned fields: reference to the node’s element, link to node’s parent and some kind of collection to store links into node’s children. Figure 4.2 shows a representation of a node of a tree and a schematic representation of the data structure associated with a node and its children.

The tree implemented in this thesis is definitely a linked structure tree since by means of pointers the parent node and child node are linked in both ways.

Preorder Traversal

A traversal of a tree **T** is a method of acquiring all the nodes of **T**. Preorder traversal which is a basic traversal scheme for trees has been represented here.



(a) (b)

Figure 4.2.: The linked structure for a general tree: (a) the node structure; (b) the portion of the data structure associated with a node and its children.

In a preorder traversal of **T**, the root of **T** is called initially and further the subtrees rooted at its children are traversed repeatedly. In an ordered tree, the subtrees are traversed according to the order of the children. The definitive action associated with the "visit" of a node depends on the application of this traversal.

Algorithm 1 preorder(T, p)

```
1: procedure PREORDER(T, P)
2:     perform the visitaction for node p
3:     for each child do p of q
4:         recursively traverse the subtree rooted at q by calling preorder (T,q)
```

This algorithm is suitable for creating a linear ordering of the nodes of a tree in which parents always come before their children. The preorder traversal is an effective way to access all the nodes of a tree.

Figure 4.3 shows an example of the preorder traversal of the tree associated with a research paper. The document is sequentially read from beginning to the end.

So again, in this thesis the accessing order is exactly like a preorder traversal and there is a linear ordering of children so that we always know the recursive algorithms imple-

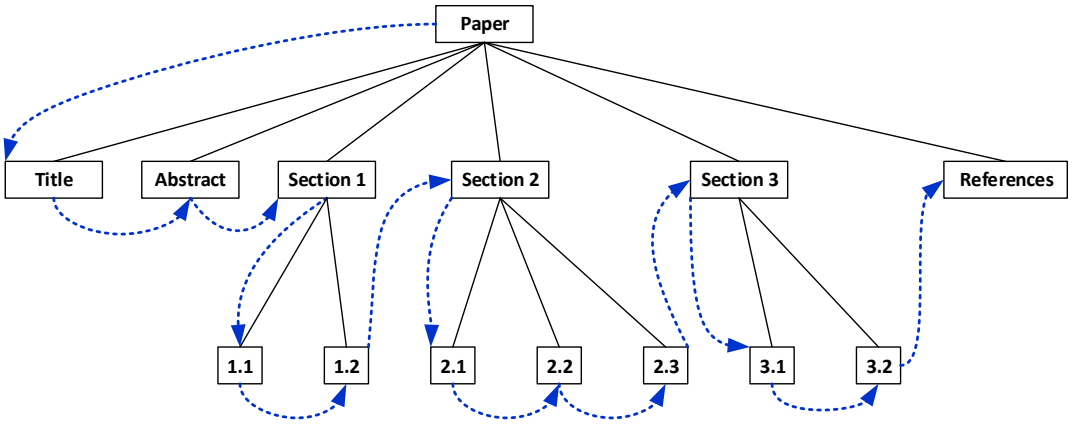


Figure 4.3.: Preorder traversal of an ordered tree, where the children of each node are ordered from left to right.

mented for different task are done

Quad Trees

A **quad tree** is an ordered tree in which each node has at most four children

1. Every node has at most four children.

2. Each child node can be labeled as south-west(SW), south-east(SE), north-west(NW) or north-east(NE).

3. The ordering of a node is implemented as SW→SE→NW→NE.

A quad tree is termed to be **proper** if each node has either zero or four children.

A Recursive Quad Tree Definition

A recursive quad tree can be defined as a quad tree which

- is either empty

- consists of a node which is **root** and four further quad trees which are called as south-west(SW), south-east(SE), north-west(NW) or north-east(NE) subtrees.

Figure 4.4 shows a representation of a node in a linked data structure for representing a quad tree.

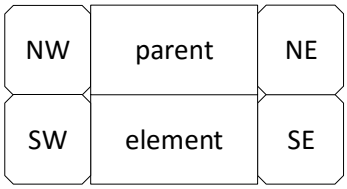


Figure 4.4.: A node in a linked data structure for representing a quad tree.

4.2. Main scheme of the implementation

To present the idea of main scheme which the work of this thesis is based upon an example of data structure and how the solution tree looks like is given in figure 4.5. The main reason here is that based on this visual element every aspect of the underlying refinement scheme can be observed. First, the root of this tree is the initial combination solution projected to a full grid. A total error after comparing the result to the normal full grid in the node will be evaluated and if it is higher than a predefined threshold then that node will be refined. Certainly for the root we require that it always gets refined.
One should note that under the refinement, a node basically will be internal and produces four children namely SW,SE,NW,NE thus the naming quad tree. Basically a new node is the same as the initial node in that it will be produced with same level vector using a combination technique so it can be imagined exactly as another node of the tree which can also be refined if needed. Application to higher dimensions should seem straightforward after this.

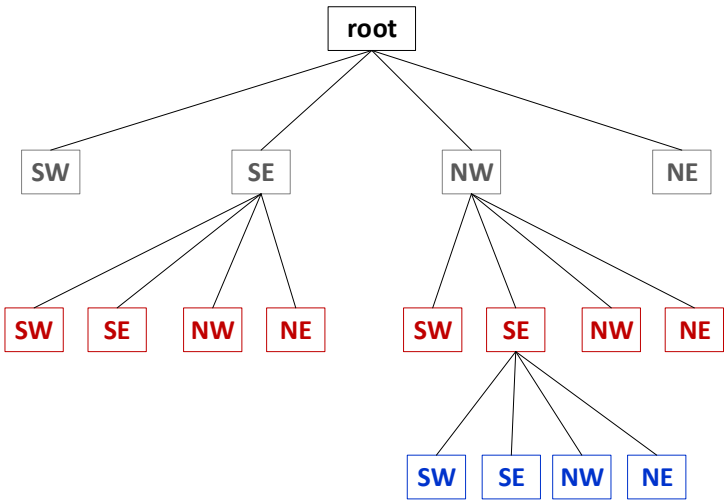


Figure 4.5.: Quad tree representation

Important discussion again was that we stick to the same pattern in our lower nodes. This can be seen exactly in the figure 4.6.

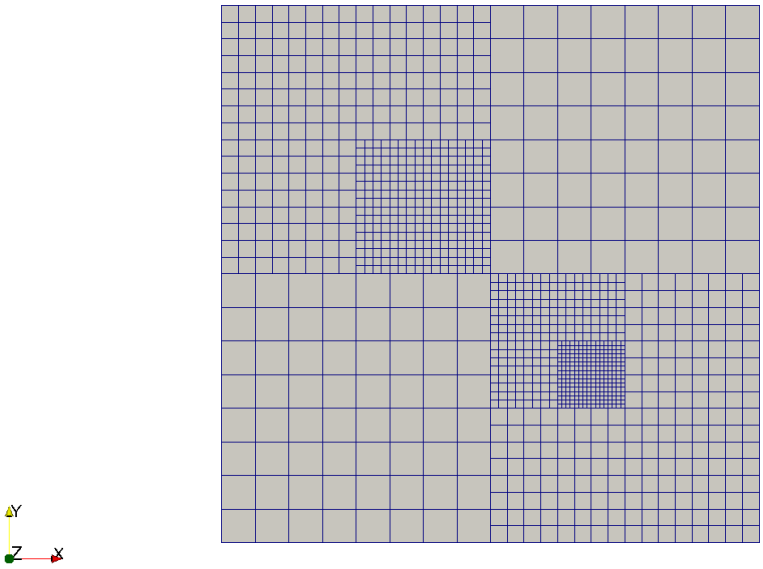


Figure 4.6.: Example of main scheme

5. Results

In this chapter the result of the current implementation will be presented and discussed in details to have an insight of the important observable issues. Trying to tackle the different schemes and naturally compare it to a full grid method is the primary objective here.

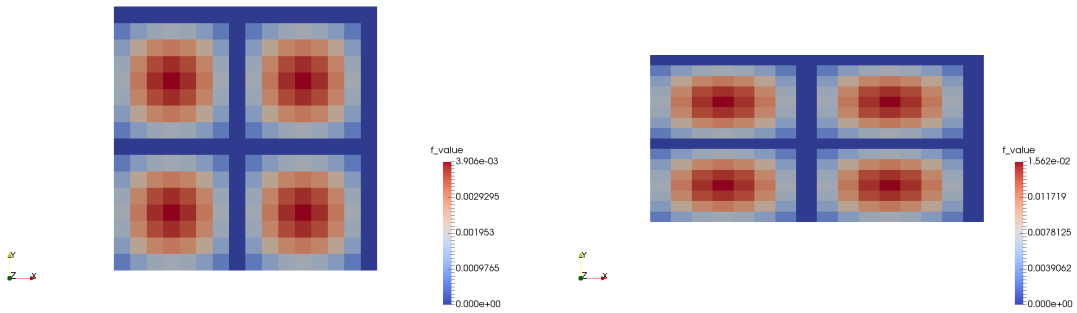
In addition, not only the solution at hand is the base for higher dimension, it can also be a base for different applications, such as regression and data fitting problem in data mining, preconditioner for multigrid-like methods, partial differential equation solver in elliptic and parabolic problems, and more importantly in computational mechanics, chemistry or physics. However, by realizing this hierarchy of complexity of problems, it can be observed that interpolation problem is the fundamental method which is the task discussed here. The author acknowledges the drawbacks of method mainly in regard to convergence, but also addresses the reasons which could resolve a part of those problems for the current implementation.

As discussed earlier in chapter 4, the objective here is to project the result of the combination technique to the full grid. This has already been done by the fact that the underlying idea of the current implementation is to first project the function values of different level vector grids all to be combined with into a corresponding full grid and then combine them pointwise. This way some advantages and some disadvantage compared to other combination schemes presented in the literature can be observed.

Firstly, the obvious disadvantage would be the fact that by projection of all grids to a full grid requires a lot of extra memory consumption and this is exactly against the idea of rectifying curse of dimensionality. However, a solution is described here to address this problem, which will be better observed after the examination of the results of the adaptive refinement strategy. Assuming the method works for the adaptive refinement, it gives the rise to idea that it is required to use a coarse grid and use adaptive refinement in the areas needed. This way, total error of same order will be achieved with less storage and operations in comparison to full grid which needs high resolution in the first place. In the worst case scenario, the full grid resolution will be achieved and the solution structure requires exactly the number of grids in combination technique scheme multiplied by the storage required for the normal full grid. Since high valued level vector is not used in this case, it is not a significant weakness.

In contrast, the advantage that can be achieved in this method is that while the required tasks performed separately for each of the grids in the implemented version of combination method is same as normal combination technique, the extra effort is just from a projection. Therefore, using a very efficient projection method, there are not many operations added to the solution which ultimately means less computational effort compared to solution on normal full grid method. Another advantage of this, as discussed earlier, can be possibility to start with a coarse grain solution and adaptively refine to areas needed. A detailed investigation of using lower resolution schemes for lower subtree problems can possibly show better result for storage space and memory usage.

Note that conventionally we are using the unit square domain as shown in figure 5.1a. The reason for this is that mathematically, the division of square domain is easier and straightforward. However, our implementation can work with any rectangular domain as shown in an example in figure 5.1b.



(a) Square domain (b) Rectangular domain

Figure 5.1.: (a) Used square domain for the adaptive combination technique computation, (b) Sample rectangular domain

5.1. Verification of components

In any case, as for any scientific approach, we first need to verify that our implementation works as expected. This has been done by comparison of the solution to normal full grid problem. We present here different test cases to ensure the functionality under a variety of cases.

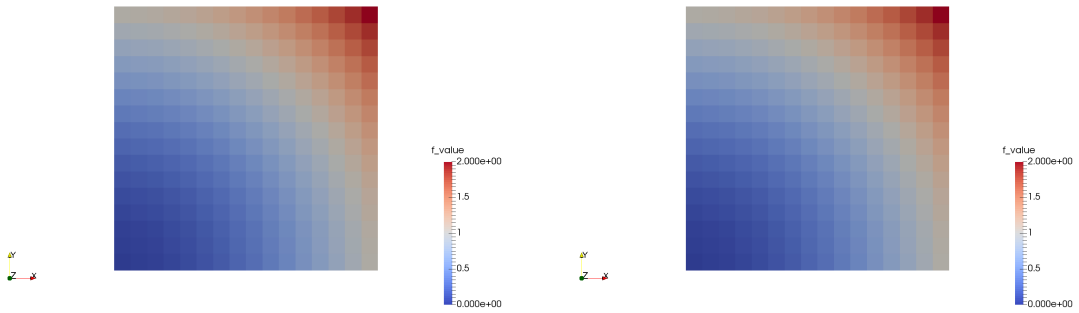
1. $f(x) = x^2 + y^2$
2. $f(x) = x^2 \cdot y^2$

3. $f(x) = \sqrt[2]{x} \cdot \sqrt[2]{y}$

4. $f(x) = 16 \cdot x(1-x)y(1-y)$

In each case we try to compare the error of combination technique, given the default scheme of $\overrightarrow{l} = (4, 4)$.

In every case, the figures showing the difference are compiled in figure 5.6 to better observe and compare the extent of the spread of error in the domain. For special test case 1, the results are shown in figure 5.2. Since the bilinear interpolation is used in this case, figure shows that the combination technique gives the perfect solution. The reason for this is the nature of problem. Since bilinear interpolation is performed in one direction first and then in the other direction, it can be observe that interpolation of a function which is not of terms $x^even \cdot y^even$ gives no error. Figures 5.3-5.5 show the results for cases 2-4



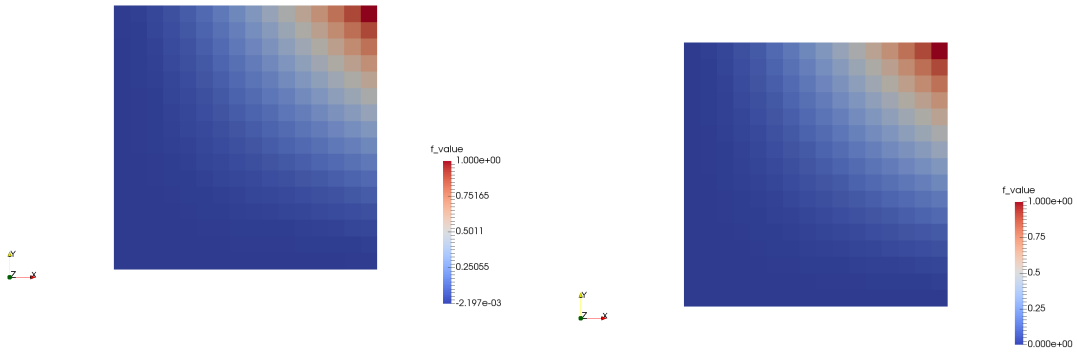
(a) Combination (b) Verification

Figure 5.2.: Results for (a) combination and (b) verification of components for case 1

The absolute values of the general error for cases 1 to 4 are compiled in the table **??**. The general error is 0 for case 1, as already discussed. Case 4 shows a much higher absolute error as the function has a multiplicative factor of 16, leading to higher function values.

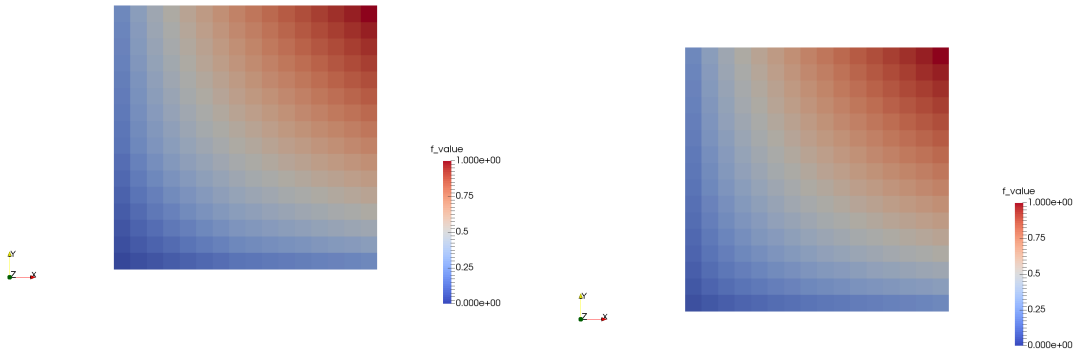
Table 5.1.: The resulting general error for cases 1-4 for verification of components

Test Case Number	Function	General Error
1	$f(x) = x^2 + y^2$	0
2	$f(x) = x^2 \cdot y^2$	0.4306
3	$f(x) = \sqrt[2]{x} \cdot \sqrt[2]{y}$	0.9224
4	$f(x) = 16 \cdot x(1-x)y(1-y)$	6.8906



(a) Combination (b) Verification

Figure 5.3.: Results for (a) combination and (b) verification of components for case 2



(a) Combination (b) Verification

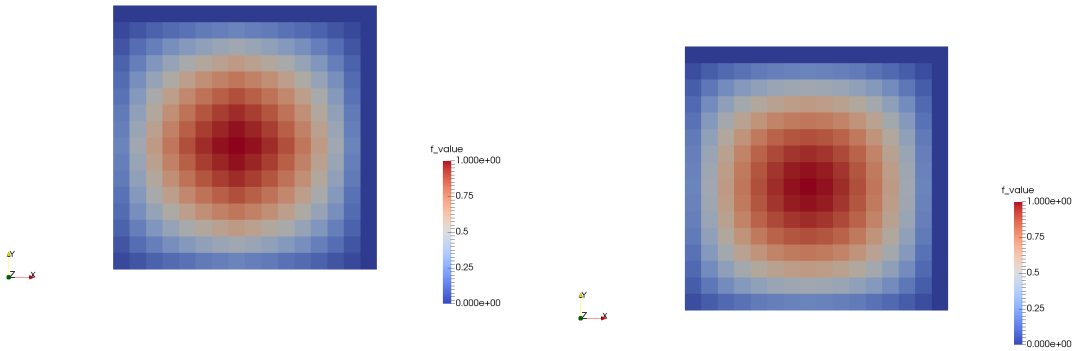
Figure 5.4.: Results for (a) combination and (b) verification of components for case 3

5.2. Results of local refinement (spatial adaptivity)

Based on the the fact that the current method has been verified, next two different cases are checked for local refinement – one simply to ensure the proper functionality of the implementation of recursive tree, and second for the test cases presented above.

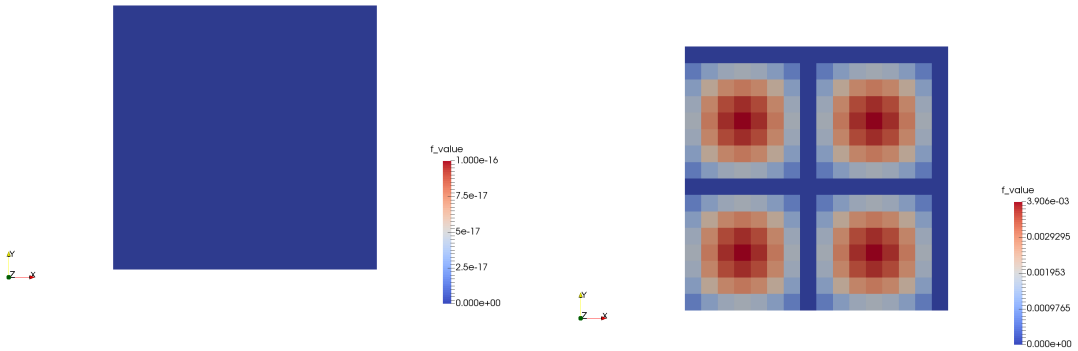
5.2.1. Error indicator based on predefined error function

As explained earlier, the objective here is to come up with a valid test case for local refinement. Based on author's background in mesh generation, there is a technique specially for unstructured grid generation with background predefined function [15] and adaptive grid generation algorithm[34]. Motivated by ideas presented we will enforce a predefined error to the problem by defining an error function which is higher than our threshold for certain

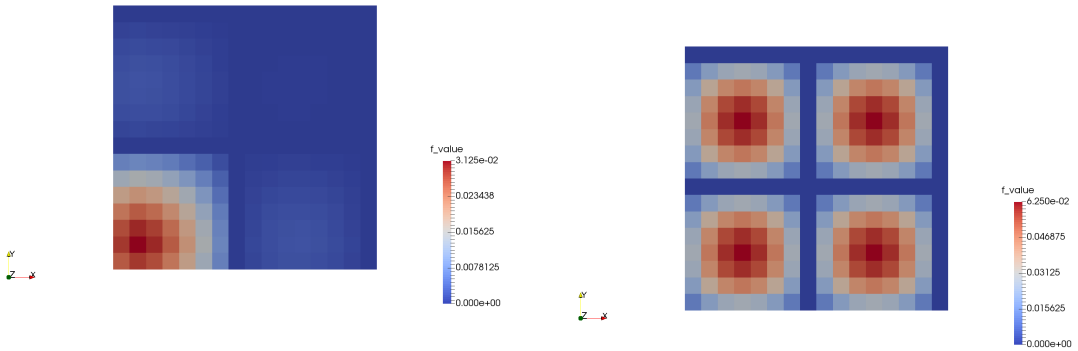


(a) Combination (b) Verification

Figure 5.5.: Results for (a) combination and (b) verification of components for case 4



(a) Case 1 (b) Case 2



(c) Case 3 (d) Case 4

Figure 5.6.: Results for combination techniques for cases 1-4

regions. This way we can predict the expected working of the local refinement exactly. For better clarity, the solution tree structure is presented in figure 5.7 and the predefined function with it's image on the unit square problem is presented in figures 5.8 and 5.9.

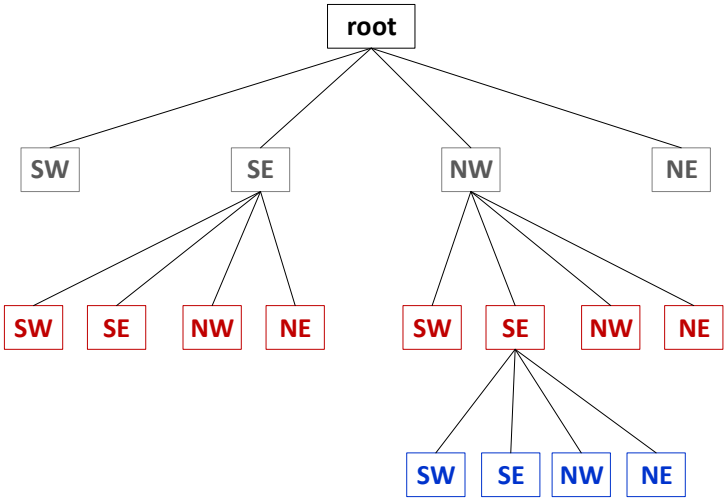


Figure 5.7.: Quad tree representation

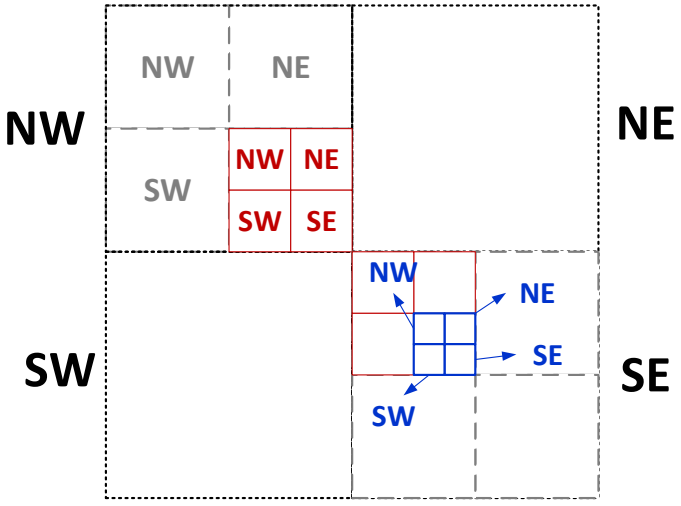


Figure 5.8.: Visualisation of the quad tree

As seen in figure 5.10, the solution to this case matches the expectations. Based on this result, the final evaluation is performed in the next section.

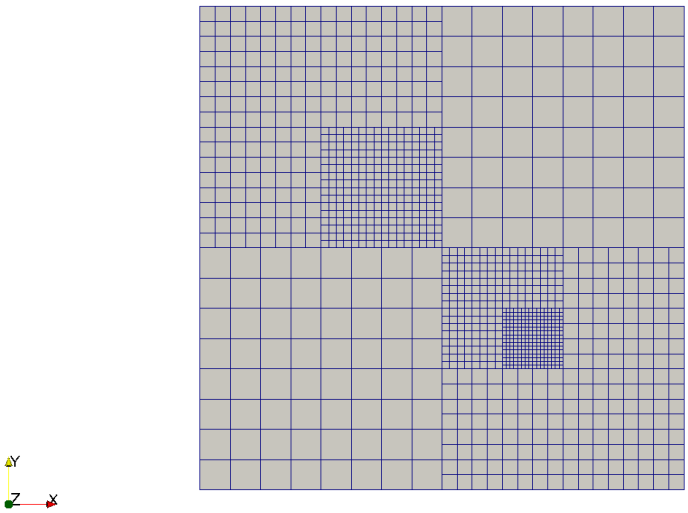


Figure 5.9.: Meshing for a predefined refinement using the shown quad tree

5.2.2. Error indicator based on solution of combination technique

After verification of the code in both adaptive and non-adaptive cases, the next focus is on further investigation into the actual problem at hand, which is performing the adaptation in correspondence with the local error of solution in each node or subtree. For that, an error indicator has been introduced which compares the solution to the full grid method. The primary reason for defining some extra test cases in the verification part was that to be able to observe how the local refinement works with different cases, otherwise there is much similarity between case 2, 3 and 4. The results are visualized in figures 5.11, 5.12 and 5.13 for cases 2,3 and 4, respectively.

It is important to point out the necessity of visualizing the error in the solution because from the solution images, the difference is hard to distinguish with human eye.

5.3. Error and Accuracy

In the last section the error is visualized in unit square simply because it doesn't make sense to just give one total error value for a special schemes. The difference here is to have multiple refinement levels performed in a loop/recursive way and then check how the total error acts. Based on arguments in literature a logarithmic behavior is expected. Key factors in the implemented code which are at play are threshold for the error and the level

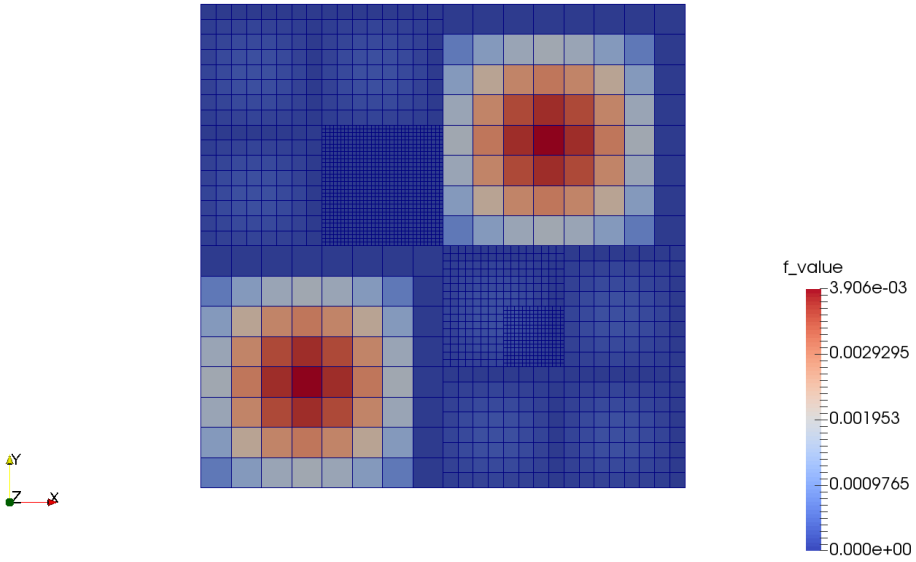


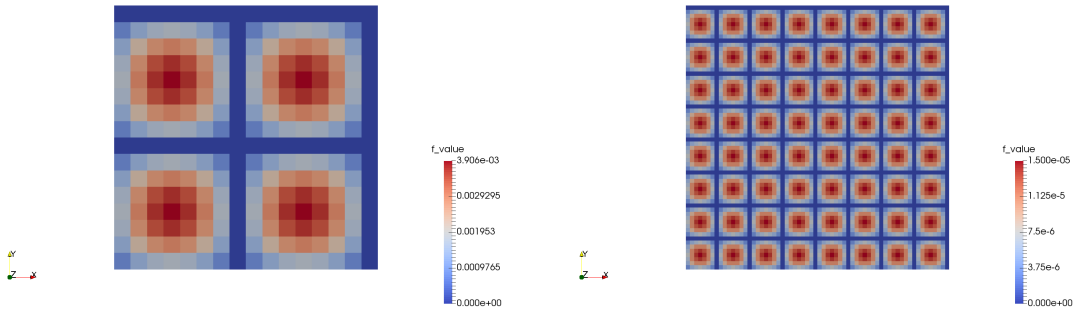
Figure 5.10.: Predefined error function user for function evaluation

of refinement. Based on them some other schemes imagined here are as follow:

1. Threshold independent, refinement level changing.
2. Threshold changing, refinement level constant.

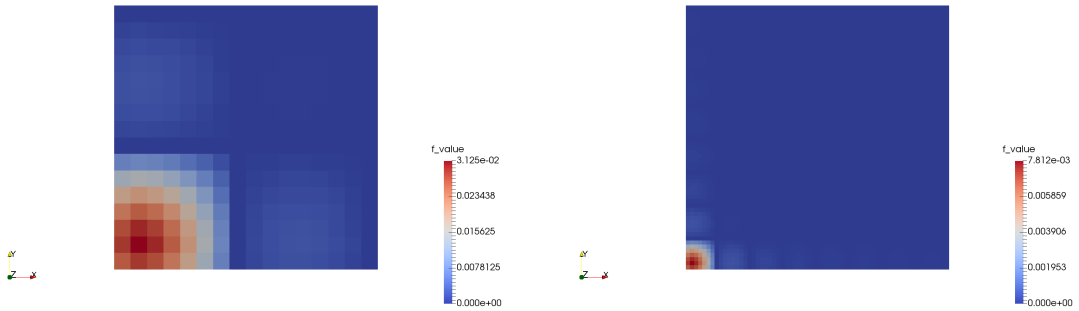
The reason for these two scheme is that the results should be separated and based on the factors at play. Figures 5.14 and 5.15 show the resulting error graphs for cases 2 to 4. The difference between the two cases is that one makes the height of the tree to be higher in some local part and the other one ensures that in each level we reach an expected local error.

As expected, the logarithmic error in both cases decreases with an increase in the level of mesh refinement and becomes constant after one point. It highlights the fact that the mesh independence is achieved in both cases and the results are free from the effect of the mesh refinement. Comparison of figures 5.14 and 5.15 for case 2 shows much higher error in the threshold-independent approach and a significantly lower error in the refinement-constant approach. The reason is due to the specific definition of the function, in which only a small subsection is relevant for mesh-refinement. As the threshold-independent approach is unable to focus on a specific subsection, the resulting error is higher by a factor of 10 compared to the refinement-constant approach.



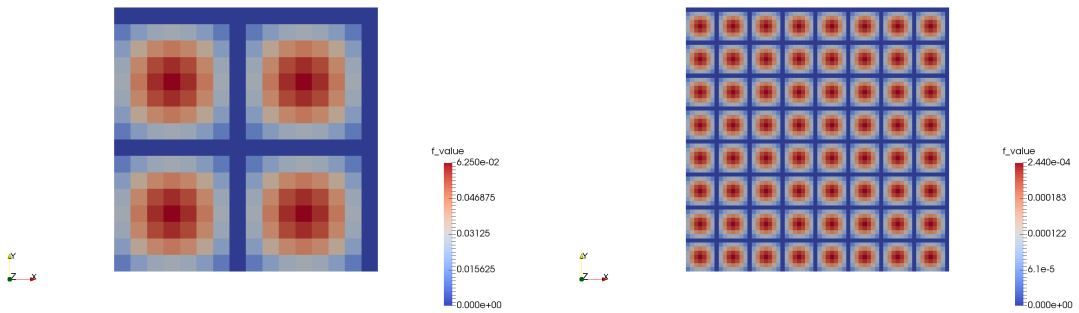
(a) case 2 before refinement (b) case 2 after refinement

Figure 5.11.: Comparison of the error values for case 2 (a) before and (b) after refinement



(a) case 3 before refinement (b) case 3 after refinement

Figure 5.12.: Comparison of the error values for case 3 (a) before and (b) after refinement



(a) case 4 before refinement (b) case 4 after refinement

Figure 5.13.: Comparison of the error values for case 4 (a) before and (b) after refinement

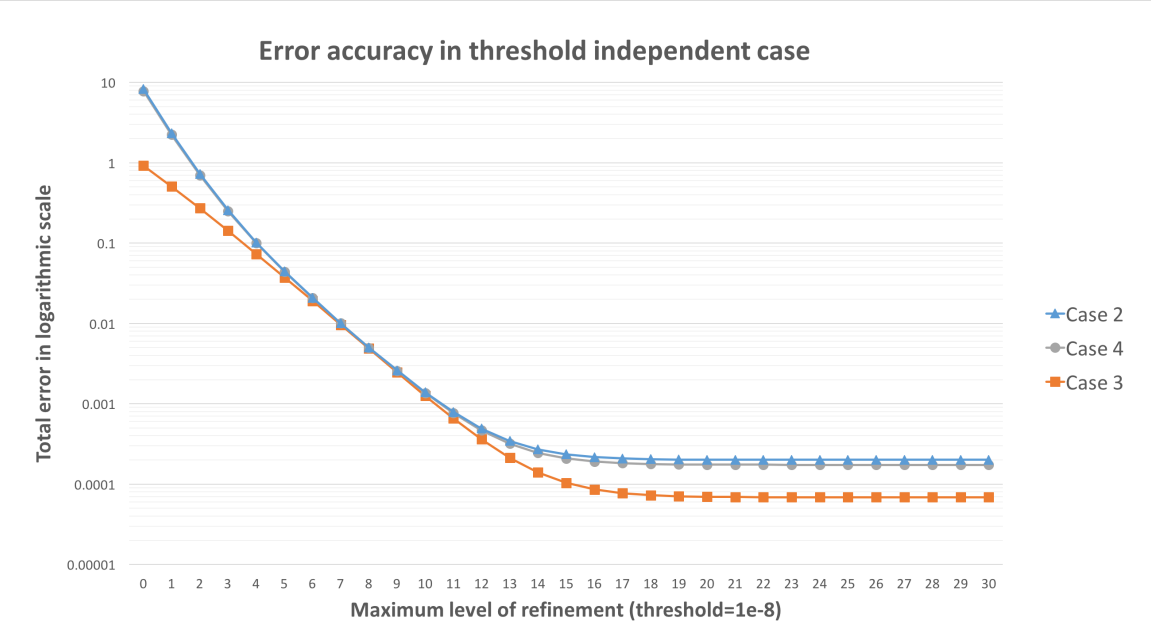


Figure 5.14.: Logarithmic visualization of the error values for the threshold independent, refinement level changing approach for cases 2 to 4

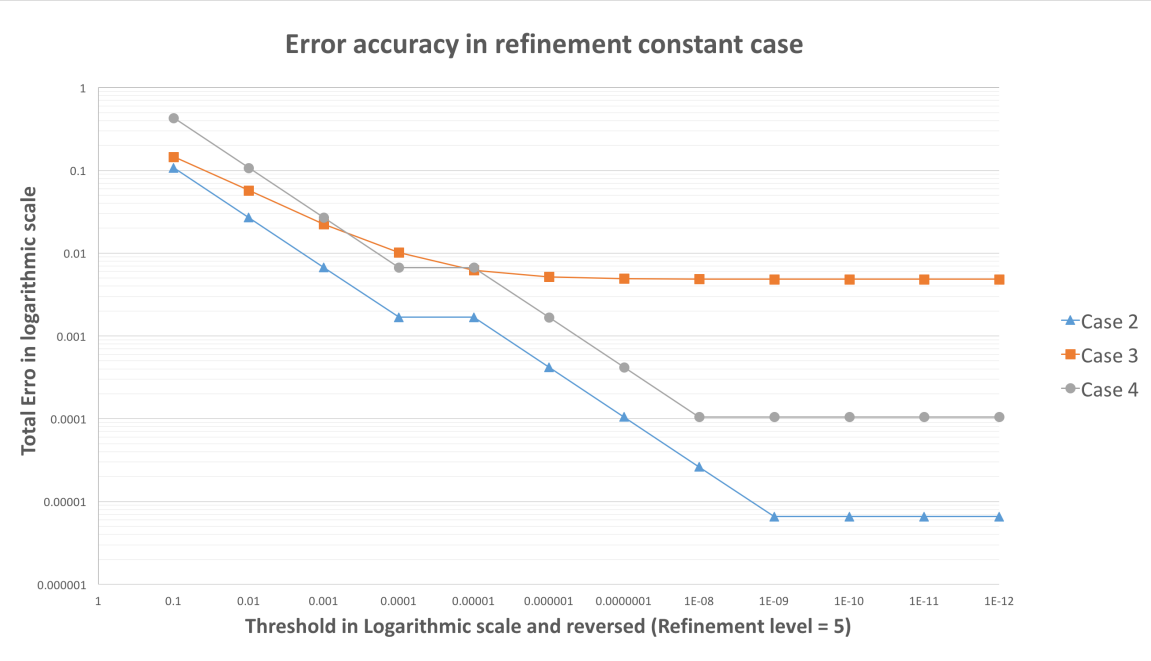


Figure 5.15.: Logarithmic visualization of the error values for the threshold changing, refinement level constant approach for cases 2 to 4

6. Conclusion

6.1. Conclusion

This thesis investigated a new adaptive algorithmic approach using sparse grids and combination technique for solving interpolation types of problems. It is advantageous in terms of its robustness less floating point operations(FLOPs). The approach is also suitable for parallelization inherently separation of the problem to independant subproblems. Focused on reaching same order of grid points and memory use as conventional combination methods the approach tries to achieve better accuracy by faster adapting to region with higher order of error by starting from much coarser grid.

Several test functions as listed below verify that our implementation works as expected compared to high resolution full grid without adaptation and thus validated to ensure the functionality under a variety of cases.

1. $f(x) = x^2 + y^2$
2. $f(x) = x^2 \cdot y^2$
3. $f(x) = \sqrt[2]{x} \cdot \sqrt[2]{y}$
4. $f(x) = 16 \cdot x(1-x)y(1-y)$

The default scheme of $\overrightarrow{l} = (4, 4)$ for all the nodes of the tree including the root is assumed. In case the test function is not of hyperbolic form with respect to two dimension the bilinear interpolation used in combination technique gives the perfect solution. Simulations are run to observe the error for the local mesh refinement for two independent cases. The error accuracy is observed to be better than full grid mesh refinement. Test cases are also prepared in order to observe the mesh refinement when a predefined error function is enforced to the problem. Besides the predefined error, a simulation based error indicator for local grid has been investigated. The error accuracy is observed to be atleast as effective as the full grid mesh refinement. Under changing levels of refinement in a loop the logarithmic behavior of error as expected form the literature is observed for the solution suggested.
To examine the independent effect of the two key factors, threshold for the error and the level of refinement, two approaches are evaluated.

1. Threshold independent, refinement level changing.
2. Threshold changing, refinement level constant.

The logarithmic error in both cases with an increase of that factor and thus level of refinement becomes saturated after one point since we there is no further refinement. Two different cases are different because one makes the height of the tree to be higher in some local part and the other one ensures that in each level we reach an expected local error. Test case 2 shows much higher error in the threshold-independent approach than the refinement-constant approach due to the definition of the function, in which error in concentrated in one subsection.
Overall, the conclusion is that the approach works as expected under different circumstances and while it has been validated for both adaptive and non-adaptive versions it shows the application of this method will be promising for future research and maybe a baseline for prospective scientist. Although It had worked for the cases presented but the author believes certain functions with higher derivatives or with singularities can bring some problems as one first need to detect the errors on the grid position. Therefore, If the variation of the function is inside the blocks basically this method will not work.

6.2. Suggestions for future works

As described earlier the interpolation problem is just a baseline for the adaptive mesh refinement combination technique. However, usually problems which has been tackled extensively in relevant literature e.g. regression and partial differential equation problem require higher complexity. This is becasue the numerical operations there are much more complex as they can be even non-linear in format. Thus definition of error indicator in such problems might be tricky but once we can achieve that we can investigate whether the current method can be applied there.

Appendix

A. Appendix

A.1. Source Code

Listing A.1: Source code for adaptive combination technique

```
1  #ifndef COMBINATION_H
2  #define COMBINATION_H
3
4  #include "Fullgrid.h"
5  #include <stdio.h>
6
7  class Combination {
8  private:
9          /*private member variable and function definitions. */
10
11 protected:
12         /*protected member variable and function definitions. */
13
14
15 public:
16         /*public member variable and function definitions. */
17         Fullgrid _grid_l1 , _grid_l2 , _grid_lmin;
18         Fullgrid _grid_l1_interpolated , _grid_l2_interpolated ,
               _grid_lmin_interpolated;
19         Fullgrid _grid_combination;
20         int level_max[DIMENSION],level_min[DIMENSION];
21         int _Npoints[DIMENSION];
22
23         /** Constructor for the Combination */
24         Combination(int* level_1,int* level_2,int* _level_max,
               int* _level_min, Domain domain)
25         :_grid_l1(level_1, domain),_grid_l2(level_2, domain),
               _grid_lmin(_level_min, domain)
```

```
26          ,_grid_l1_interpolated(_level_max, domain),
                _grid_l2_interpolated(_level_max, domain)
27          ,_grid_lmin_interpolated(_level_max, domain),
                _grid_combination(_level_max, domain){
28
29                  for ( int i = 0; i < DIMENSION; ++i ) {
30                          level_max[i]=_level_max[i];
31                          level_min[i]=_level_min[i];
32                  }
33                  //evaluate case function for each grids needed to
                        be combined
34                  _grid_l1.evaluate(domain);
35                  _grid_l2.evaluate(domain);
36                  _grid_lmin.evaluate(domain);
37
38          }
39
40          /** Destructor Deallocates the data
41           * TASK
42           *  */
43          virtual ~Combination() {
44                  //delete[] ;
45          }
46
47          void GetCombination(int* level_1,int* level_2, Domain
                domain){
48                  /*apply the method iterate*/
49                  for ( int i = 0; i < DIMENSION; ++i ) {
50                          _Npoints[i]=(pow(2,level_max[i]))+1;
                                //2^l+1
51                  }
52                  direct_injection(level_1,_grid_l1,domain,&
                        _grid_l1_interpolated);
53                  direct_injection(level_2,_grid_l2,domain,&
                        _grid_l2_interpolated);
54                  direct_injection(level_min,_grid_lmin,domain,&
                        _grid_lmin_interpolated);
55
56                  interpolation(level_1,&_grid_l1_interpolated,
                        domain);
```

```
57                interpolation(level_2,&_grid_l2_interpolated,
                      domain);
58                interpolation(level_min,&_grid_lmin_interpolated,
                       domain);
59
60                for ( int i = 0; i < _Npoints[0]; ++i ) {
61                        for ( int j = 0; j < _Npoints[1]; ++j ) {
62                                _grid_combination.function_values
                                    [i][j]= _grid_l1_interpolated.
                                    function_values[i][j] +
                                    _grid_l2_interpolated.
                                    function_values[i][j] -
                                    _grid_lmin_interpolated.
                                    function_values[i][j];
63                        }
64                }
65
66        }
67
68        void interpolation(int* level, Fullgrid*
              interpolated_grid, Domain domain){
69
70                int lower_local_index[DIMENSION],
                      upper_local_index[DIMENSION];
71                //interploation in X direction
72                for ( int i = 0; i < _Npoints[0]-1; ++i ) {
73                        for ( int j = 0; j < _Npoints[1]; ++j ) {
74                                lower_local_index[0]=(int) floor(
                                     i/( pow( 2,level_max[0]-level
                                    [0] ) ) ) ;
75                                upper_local_index[0]=
                                    lower_local_index[0]+1;
76                                interpolated_grid->
                                    function_values[i][j]= ((
                                    double) ( upper_local_index[0]
                                     * ( pow( 2,level_max[0]-level
                                    [0] ) ) - i))/((double) ( pow(
                                     2,level_max[0]-level[0] ) ))
                                    * interpolated_grid->
                                    function_values[(int) (
```

```
                                    lower_local_index[0]* pow( 2,
                                    level_max[0]-level[0] ) )][j]+
                                     ((double) ( i -
                                    lower_local_index[0] * ( pow(
                                    2,level_max[0]-level[0] ) ) ))
                                    /((double) ( pow( 2,level_max
                                    [0]-level[0] ) )) *
                                    interpolated_grid->
                                    function_values[(int) (
                                    upper_local_index[0]*pow( 2,
                                    level_max[0]-level[0] ))][j];
77                          }
78                  }
79                  //interploation in Y direction
80                  for ( int i = 0; i < _Npoints[0]; ++i ) {
81                          for ( int j = 0; j < _Npoints[1]-1; ++j )
                                {
82                                  lower_local_index[1]=(int) floor(
                                     j/( pow( 2,level_max[1]-level
                                    [1] ) ) );
83                                  upper_local_index[1]=
                                    lower_local_index[1]+1;
84                                  interpolated_grid->
                                    function_values[i][j]= ((
                                    double) ( upper_local_index[1]
                                     * ( pow( 2,level_max[1]-level
                                    [1] ) ) - j))/((double) ( pow(
                                     2,level_max[1]-level[1] ) ))
                                    * interpolated_grid->
                                    function_values[i][(int) (
                                    lower_local_index[1]*pow( 2,
                                    level_max[1]-level[1] ))]+ ((
                                    double) ( j -
                                    lower_local_index[1] * ( pow(
                                    2,level_max[1]-level[1] ) ) ))
                                    /((double) ( pow( 2,level_max
                                    [1]-level[1] ) )) *
                                    interpolated_grid->
                                    function_values[i][(int) (
                                    upper_local_index[1]*pow( 2,
```

```
                                        level_max[1]-level[1] ))];
85                          }
86                  }
87
88          }
89
90          void direct_injection(int* level, Fullgrid _grid, Domain
               domain, Fullgrid* _projected_grid){
91
92                  // Initialize to zero
93                  for ( int i = 0; i < _Npoints[0]; ++i ) {
94                          for ( int j = 0; j < _Npoints[1]; ++j ) {
95                                  _projected_grid->function_values[
                                       i][j]= 0;
96                          }
97                  }
98
99                  //for all local values we directly project to
                       global
100                 int _Npoints_local[DIMENSION];
101                 for ( int i = 0; i < DIMENSION; ++i ) {
102                         _Npoints_local[i]=(pow(2,level[i]))+1;
                               //2^l+1
103                 }
104
105                 for ( int i = 0; i < _Npoints_local[0]; ++i ) {
106                         for ( int j = 0; j < _Npoints_local[1];
                               ++j ) {
107                                 _projected_grid->function_values[
                                       (int) (i*pow( 2,level_max[0]-
                                       level[0] ))][ (int) (j*pow( 2,
                                       level_max[1]-level[1] )) ]=
                                       _grid.function_values[i][j];
108                         }
109                 }
110         }
111
112 };
113
114 #endif
```

Listing A.2: Source code for Full grid method

```
1  #ifndef FULLGRID_H
2  #define FULLGRID_H
3
4  #include "Definitions.h"
5  /*here Full grid structure will be build*/
6
7  class Fullgrid {
8  public:
9          /*public member variable and function definitions. */
10         int _Npoints[DIMENSION];
11         double _mesh_size[DIMENSION];
12         double** function_values;
13         double *position_d1, *position_d2;
14
15         /** Default constructor for the fullgrid
16          */
17         Fullgrid(int* level, Domain _domain) {
18                 for ( int i = 0; i < DIMENSION; ++i ) {
19                         _Npoints[i]=(pow(2,level[i]))+1;
                               //2^l+1
20                         _mesh_size[i]= (_domain.end_point[i] -
                               _domain.start_point[i]) / (pow(2,level
                               [i]) );     //2^-l
21                 }
22                 position_d1 = new double[ _Npoints[0] ];
23                 position_d2 = new double[ _Npoints[1] ];
24                 //Assign first dimension
25                 function_values = new double*[_Npoints[0]];
26                 //Assign second dimension
27                 for(int i = 0; i <_Npoints[0] ; i++){
28                         function_values[i] = new double[_Npoints
                               [1]];
29                 }
30         }
31
32         //copy constructor
33         Fullgrid(const Fullgrid& _copyfromfullgrid) {
```

```
34                  for ( int i = 0; i < DIMENSION; ++i ) {
35                          _Npoints[i]=_copyfromfullgrid._Npoints[i
                                ];        //2^l+1
36                          _mesh_size[i]= _copyfromfullgrid.
                                _mesh_size[i];
37                  }
38                  position_d1 = new double[ _Npoints[0] ];
39                  position_d2 = new double[ _Npoints[1] ];
40                  function_values = new double*[_Npoints[0]];
41                  for(int i = 0; i <_Npoints[0] ; i++){
42                          function_values[i] = new double[_Npoints
                                [1]];
43                  }
44                  for ( int j_1 = 0; j_1 < _Npoints[0]; ++j_1 ) {
45                          position_d1[j_1]=_copyfromfullgrid.
                                position_d1[j_1];
46                  }
47                  for ( int j_2 = 0; j_2 < _Npoints[1]; ++j_2 ) {
48                          position_d2[j_2]=_copyfromfullgrid.
                                position_d2[j_2];
49                  }
50                  //              **(_array) = new double[
                        index_size_global]();  ///Zero-initialize (
                        somehow doesn't work here)
51                  for ( int i = 0; i < _Npoints[0]; ++i ) {
52                          for ( int j = 0; j < _Npoints[1]; ++j ) {
53                                  function_values[i][j]=
                                        _copyfromfullgrid.
                                        function_values[i][j];  //X^2*
                                        Y^2 - TASK (implement new
                                        function to do here)
54                          }
55                  }
56
57          }
58
59          //Assignment operator
60          Fullgrid& operator=(const Fullgrid& _equalfullgrid){
61                  if (this!=&_equalfullgrid){
62                          for ( int i = 0; i < DIMENSION; ++i ) {
```

```
63                          _Npoints[i]=_equalfullgrid.
                                _Npoints[i]; //2^l+1
64                          _mesh_size[i]= _equalfullgrid.
                                _mesh_size[i];
65                      }
66                      position_d1 = new double[ _Npoints[0] ];
67                      position_d2 = new double[ _Npoints[1] ];
68                      function_values = new double*[_Npoints
                            [0]];
69                      //Assign second dimension
70                      for(int i = 0; i <_Npoints[0] ; i++){
71                          function_values[i] = new double[
                                _Npoints[1]];
72                      }
73
74                      for ( int j_1 = 0; j_1 < _Npoints[0]; ++
                            j_1 ) {
75                          position_d1[j_1]=_equalfullgrid.
                                position_d1[j_1];
76                      }
77                      for ( int j_2 = 0; j_2 < _Npoints[1]; ++
                            j_2 ) {
78                          position_d2[j_2]=_equalfullgrid.
                                position_d2[j_2];
79                      }
80                      //              **(_array) = new double[
                            index_size_global]();  ///Zero-
                            initialize (somehow doesn't work here)
81                      for ( int i = 0; i < _Npoints[0]; ++i ) {
82                          for ( int j = 0; j < _Npoints[1];
                                ++j ) {
83                              function_values[i][j]=
                                    _equalfullgrid.
                                    function_values[i][j];
                                      //X^2*Y^2 - TASK (
                                    implement new function
                                     to do here)
84                          }
85                      }
86                  }
```

```
87                 return *this;
88         }
89
90         /** Destructor Deallocates the data
91          * TASK
92          */
93         virtual ~Fullgrid() {
94                 delete[] function_values;
95                 delete[] position_d1;
96                 delete[] position_d2;
97         }
98
99         /*evaluates function value for TestFunction defined in
               definitions*/
100        void evaluate(Domain _domain){
101
102                for ( int j_1 = 0; j_1 < _Npoints[0]; ++j_1 ) {
103                        position_d1[j_1]=_domain.start_point[0]+
                           j_1*_mesh_size[0];
104                }
105                for ( int j_2 = 0; j_2 < _Npoints[1]; ++j_2 ) {
106                        position_d2[j_2]=_domain.start_point[1]+
                           j_2*_mesh_size[1];
107                }
108                //              **(_array) = new double[
                   index_size_global]();  ///Zero-initialize (
                   somehow doesn't work here)
109                for ( int i = 0; i < _Npoints[0]; ++i ) {
110                        for ( int j = 0; j < _Npoints[1]; ++j ) {
111                                function_values[i][j]=
                                   TestFunction(position_d1[i],
                                   position_d2[j]);  //X^2*Y^2 -
                                   TASK (implement new function
                                   to do here)
112                        }
113                }
114        }
115
116
117
```

```
118         /** notice: by introducing domain it may need to change
                accordingly by adding level of domain
119         by putting level_max it will give the index for full grid
120          */
121         /** here a multidimensional indices which is stored in 1D
                array called "index" is mapped to a value to use in
                grid
122          *  TASK
123          */
124         /**
125         int Map2arrayIndex_local(int * index, int*level)  {
126                 int multiplier=1,mapped_index=0;
127                 for loop over dimensions to get to index
128                 for ( int i = 0; i < DIMENSION; ++i ) {
129                         _Npoints[i]=(pow(2,level[i]))+1;
                                //2^l+1
130                         if (i>0){
131                                 multiplier*=_Npoints[i-1];
132                         }
133                         mapped_index+=index[i]*multiplier;
134                 }
135                 return mapped_index;
136         }
137          */
138 private:
139
140 };
141
142 #endif
```

Listing A.3: Source code for quad trees

```
1 #ifndef QUADTREE_H
2 #define QUADTREE_H
3
4 #include "Definitions.h"
5 #include "Combination.h"
6 #include "Visual.h"
7 #include <math.h>
8
9 class Node{
```

```
10 public:
11         bool refined_factor; //whether the node should be refined
               or not 0=no refinement 1=to be refined
12         double total_error_in_node; //total error calculated in
              the subtree
13         double total_error_in_node_using_predefined_function; //
14         int node_level;
15         string name;
16         Domain domain;
17
18
19         Fullgrid* beforecombi_fullgrid;
20         Fullgrid* aftercombi_fullgrid;
21         Fullgrid* difference_grid;
22
23         Node* parent_node;
24         Node* NW_child_node;
25         Node* NE_child_node;
26         Node* SE_child_node;
27         Node* SW_child_node;
28
29
30         //constructor
31         Node(): parent_node(NULL),NW_child_node(NULL),
              NE_child_node(NULL),SE_child_node(NULL),SW_child_node(
              NULL){};
32
33 };
34
35
36 #endif
```

Listing A.4: Source code for solution

```
1 #ifndef SOLUTION_H
2 #define SOLUTION_H
3
4 #include "QuadTree.h"
5 #include "Definitions.h"
6 #include "Combination.h"
7 #include "Visual.h"
```

```
 8 #include <math.h>
 9 #include <iostream>
10 using namespace std;
11
12 #include <sys/types.h>
13 #include <sys/stat.h>
14 #include <unistd.h>
15
16 bool solve(Node* current_node);
17
18 void expand_Quad(Node* current_node);
19
20 double evaluate_predefinedError(Domain new_domain);
21
22 void AdaptiveRefinement(Node* current_node);
23
24 bool solve(Node* current_node){
25
26         current_node->beforecombi_fullgrid      = new Fullgrid(
              level_max, current_node->domain);
27         current_node->aftercombi_fullgrid       = new Fullgrid(
              level_max, current_node->domain);
28         current_node->difference_grid           = new Fullgrid(
              level_max, current_node->domain);
29
30         current_node->beforecombi_fullgrid->evaluate(current_node
              ->domain);
31
32         Combination* _combinationSolution= new Combination(
              level_1,level_2,level_max,level_min,current_node->
              domain);
33         _combinationSolution->GetCombination(level_1,level_2,
              current_node->domain);
34         current_node->aftercombi_fullgrid=&_combinationSolution->
              _grid_combination;
35
36         //************************Calculation of difference
              value between sparse grid combination and normal full
              grid and computing total error of the node
37         current_node->total_error_in_node=0;
```

```
38          for ( int i = 0; i < _Npoints[0]; ++i ) {
39                  for ( int j = 0; j < _Npoints[1]; ++j ) {
40                          current_node->difference_grid->
                               function_values[i][j]= current_node->
                               beforecombi_fullgrid->function_values[
                               i][j] - current_node->
                               aftercombi_fullgrid->function_values[i
                               ][j];
41                          if (current_node->difference_grid->
                               function_values[i][j]<0){
42                                  current_node->total_error_in_node
                                       -=current_node->
                                       difference_grid->
                                       function_values[i][j];//if it'
                                       s negative then make it
                                       positive and add (sum +
                                       absolute(x)) = sum - x
43                          }
44                          else{
45                                  current_node->total_error_in_node
                                       +=current_node->
                                       difference_grid->
                                       function_values[i][j];//if it'
                                       s positive then just add (sum
                                       + absolute(x)) = sum + x
46                          }
47
48                  }
49          }
50          //*************************
51          if (USE_PREDEFINED_ERROR==0){ //if we use predefined then
                 we don't use the error calculated
52                  if (current_node->total_error_in_node>
                       TRESHOLD_ERROR && current_node->node_level<
                       MAX_ADAPTATION_LEVEL){    //if error is not
                       satisfied or we didn't go too far then make a
                       new refinement
53                          current_node->refined_factor=1;
54                  }
55                  else {
```

```
56                      current_node->refined_factor=0;
57              }
58      }
59      else if (USE_PREDEFINED_ERROR==1){
60              current_node->
                    total_error_in_node_using_predefined_function
                    = evaluate_predefinedError(current_node->
                    domain);
61              if (current_node->
                    total_error_in_node_using_predefined_function>
                    TRESHOLD_ERROR && current_node->node_level<
                    MAX_ADAPTATION_LEVEL){    //if error is not
                    satisfied or we didn't go too far then make a
                    new refinement
62                      current_node->refined_factor=1;
63              }
64              else {
65                      current_node->refined_factor=0;
66              }
67      }
68
69
70      //if only we need refining then we draw that branch
71
72
73      if (current_node->refined_factor==0){
74              if (WRITE_VTK==1){
75                      struct stat st_results = {0};
76                      if (stat("Results", &st_results) == -1) {
77                              mkdir("Results", 0700);
78                      }
79                      //vtk with subdirectory for no
                            combination
80                      struct stat st_nocombi = {0};
81                      if (stat("Results/no_combination", &
                            st_nocombi) == -1) {
82                              mkdir("Results/no_combination",
                                    0700);
83                      }
84                      string fullgrid_name="./Results/
```

```
                    no_combination/noCombi" + std::
                    to_string(current_node->node_level) +
                     current_node->name;
85                  const char *cfullgrid_name =
                    fullgrid_name.c_str();
86                  write_vtkFile(cfullgrid_name,current_node
                    ->domain,current_node->
                    beforecombi_fullgrid);
87                  //vtk with subdirectory for combination
88                  struct stat st_combi = {0};
89                  if (stat("Results/combination", &st_combi
                    ) == -1) {
90                        mkdir("Results/combination",
                          0700);
91                  }
92                  string combigrid_name="./Results/
                    combination/Combi" + std::to_string(
                    current_node->node_level) +
                    current_node->name;
93                  const char *ccombigrid_name =
                    combigrid_name.c_str();
94                  write_vtkFile(ccombigrid_name,
                    current_node->domain,current_node->
                    aftercombi_fullgrid);
95                  //vtk with subdirectory for no
                    combination
96                  struct stat st_diff = {0};
97                  if (stat("Results/difference_grid", &
                    st_diff) == -1) {
98                        mkdir("Results/difference_grid",
                          0700);
99                  }
100                 string diffgrid_name="./Results/
                    difference_grid/difference" + std::
                    to_string(current_node->node_level) +
                    current_node->name;
101                 const char *cdiffgrid_name =
                    diffgrid_name.c_str();
102                 write_vtkFile(cdiffgrid_name,current_node
                    ->domain,current_node->difference_grid
```

```
                                );
103                     }
104             }
105
106             return current_node->refined_factor;
107     }
108
109     void expand_Quad(Node* current_node){
110
111             //making this node parent of all children
112             current_node->SW_child_node = new Node();
113             current_node->SE_child_node = new Node();
114             current_node->NW_child_node = new Node();
115             current_node->NE_child_node = new Node();
116             current_node->SW_child_node->parent_node=current_node;
117             current_node->SE_child_node->parent_node=current_node;
118             current_node->NW_child_node->parent_node=current_node;
119             current_node->NE_child_node->parent_node=current_node;
120             //naming scheme
121             current_node->SW_child_node->name=current_node->name+"SW"
                    ;
122             current_node->SE_child_node->name=current_node->name+"SE"
                    ;
123             current_node->NW_child_node->name=current_node->name+"NW"
                    ;
124             current_node->NE_child_node->name=current_node->name+"NE"
                    ;
125             //Node level is added one time
126             current_node->SW_child_node->node_level=current_node->
                    node_level+1;
127             current_node->SE_child_node->node_level=current_node->
                    node_level+1;
128             current_node->NW_child_node->node_level=current_node->
                    node_level+1;
129             current_node->NE_child_node->node_level=current_node->
                    node_level+1;
130             //Setting new domains for all four children
131             //South West child
132             current_node->SW_child_node->domain.start_point[0] =
                    current_node->domain.start_point[0];
```

```
133         current_node->SW_child_node->domain.start_point[1] =
                current_node->domain.start_point[1];
134         current_node->SW_child_node->domain.end_point[0]   = (
                current_node->domain.start_point[0]+current_node->
                domain.end_point[0])/2;
135         current_node->SW_child_node->domain.end_point[1]   = (
                current_node->domain.start_point[1]+current_node->
                domain.end_point[1])/2;
136         //South East child
137         current_node->SE_child_node->domain.start_point[0] = (
                current_node->domain.start_point[0]+current_node->
                domain.end_point[0])/2;
138         current_node->SE_child_node->domain.start_point[1] =
                current_node->domain.start_point[1];
139         current_node->SE_child_node->domain.end_point[0]   =
                current_node->domain.end_point[0];
140         current_node->SE_child_node->domain.end_point[1]   = (
                current_node->domain.start_point[1]+current_node->
                domain.end_point[1])/2;
141         //North West Child
142         current_node->NW_child_node->domain.start_point[0] =
                current_node->domain.start_point[0];
143         current_node->NW_child_node->domain.start_point[1] = (
                current_node->domain.start_point[1]+current_node->
                domain.end_point[1])/2;
144         current_node->NW_child_node->domain.end_point[0]   = (
                current_node->domain.start_point[0]+current_node->
                domain.end_point[0])/2;
145         current_node->NW_child_node->domain.end_point[1]   =
                current_node->domain.end_point[1];
146         //North East Child
147         current_node->NE_child_node->domain.start_point[0] = (
                current_node->domain.start_point[0]+current_node->
                domain.end_point[0])/2;
148         current_node->NE_child_node->domain.start_point[1] = (
                current_node->domain.start_point[1]+current_node->
                domain.end_point[1])/2;
149         current_node->NE_child_node->domain.end_point[0]   =
                current_node->domain.end_point[0];
150         current_node->NE_child_node->domain.end_point[1]   =
```

```
            current_node->domain.end_point[1];
151 }
152
153 double evaluate_predefinedError(Domain new_domain){
154         double _error=0;
155         //calculation of positions might be for no reason after
               evaluate function has been called once
156         int Ncell[DIMENSION];
157         double mesh_size_domain[DIMENSION];
158         double *cellposition_d1, *cellposition_d2;
159
160         for ( int i = 0; i < DIMENSION; ++i ) {
161                 Ncell[i]=(pow(2,level_max[i])); //2^l cell
162                 mesh_size_domain[i]= (new_domain.end_point[i] -
                       new_domain.start_point[i]) / (pow(2,level_max[
                       i]) );     //2^-l
163         }
164         cellposition_d1 = new double[ Ncell[0] ];
165         cellposition_d2 = new double[ Ncell[1] ];
166
167         for ( int j_1 = 0; j_1 < Ncell[0]; ++j_1 ) {
168                 cellposition_d1[j_1]=new_domain.start_point[0]+
                       j_1*mesh_size_domain[0]+mesh_size_domain
                       [0]/2.0;
169         }
170         for ( int j_2 = 0; j_2 < Ncell[1]; ++j_2 ) {
171                 cellposition_d2[j_2]=new_domain.start_point[1]+
                       j_2*mesh_size_domain[1]+mesh_size_domain
                       [1]/2.0;
172         }
173         for ( int i = 0; i < _Npoints[0]; ++i ) {
174                 for ( int j = 0; j < _Npoints[1]; ++j ) {
175                         _error+= ErrorFunction(cellposition_d1[i
                               ],cellposition_d2[j]);  //X^2*Y^2 -
                               TASK (implement new function to do
                               here)
176                 }
177         }
178         return _error;
179 }
```

```
180
181 void AdaptiveRefinement(Node* current_node){
182
183         current_node->refined_factor=solve(current_node);
184
185         if (current_node->refined_factor==1)
186         {
187                 expand_Quad(current_node);
188
189                 AdaptiveRefinement(current_node->SW_child_node);
190                 AdaptiveRefinement(current_node->SE_child_node);
191                 AdaptiveRefinement(current_node->NW_child_node);
192                 AdaptiveRefinement(current_node->NE_child_node);
193         }
194
195 }
196
197 double compute_error(Node* current_node){
198         static int j=0;
199
200         double current_error=0;
201         if (current_node->refined_factor==0){
202                 current_error+=current_node->total_error_in_node;
203                 j++;
204                 //cout<<"current one we are at j= "<< j <<" with
                        name= "<<current_node->name<<endl;
205                 //cout<<"total error in node was=" <<
                        current_node->total_error_in_node<<endl;
206                 const char *temp = current_node->name.c_str();
207                 if (USE_PREDEFINED_ERROR==1){printf ("total error
                        predefined for %s = %f\n",temp,current_node
                        ->
                        total_error_in_node_using_predefined_function)
                        ; }
208
209         }
210         else if (current_node->refined_factor==1)
211         {
212                 current_error+=compute_error(current_node->
                        SW_child_node);
```

```
213                     current_error+=compute_error(current_node->
                            SE_child_node);
214                     current_error+=compute_error(current_node->
                            NW_child_node);
215                     current_error+=compute_error(current_node->
                            NE_child_node);
216             }
217
218             return current_error;
219     }
220
221     #endif
```

Bibliography

[1] S Smolyak. Quadrature and Interpolation Formulas for Tensor Products of Certain Classes of Functions. *Soviet Mathematics, Doklady*, 4:240–243, 1963.

[2] William J. Gordon. Blending-Function Methods of Bivariate and Multivariate Interpolation and Approximation. *SIAM Journal on Numerical Analysis*, 8(1):158–177, mar 1971.

[3] Harry Yserentant. On the multi-level splitting of finite element spaces. *Numerische Mathematik*, 49(4):379–412, jul 1986.

[4] C. Zenger. Sparse grids. *Notes on Numerical Fluid Mechanics - Parallel Algorithms for Partial Differential Equations (W. Hackbusch, ed.)*, 31, 1990.

[5] M Griebel, W Huber, U. Rüde, and T. Störtkuhl. The combination technique for parallel sparse-grid-preconditioning or -solution of PDE's on workstation networks. In Luc Bougé, Michel Cosnard, Yves Robert, and Denis Trystram, editors, *Parallel Processing: CONPAR 92‚ÄîVAPP V*, pages 217–228. Springer Berlin Heidelberg, 1992.

[6] Michael Griebel. The Combination Technique for the Sparse Grid Solution of PDE's on Multiprocessor Machines. *Parallel Processing Letters*, 02(01):61–70, mar 1992.

[7] Michael Griebel, Michael Schneider, and Christoph Zenger. A combination technique for the solution of sparse grid problems. *Iterative methods in Linear Algebra*, pages 263–281, 1992.

[8] U Rüde. Extrapolation and Related Techniques for Solving Elliptic Equations. In *BERICHT I-9135, INSTITUT FUR INFORMATIK, TU MUNCHEN*, 1992.

[9] M. Griebel. Sparse grid multilevel methods, their parallelization and their application to CFD. In J. Häser, editor, *Parallel Computational Fluid Dynamics 92*, pages 161–174. Elsevier, New Brunswick, 1993.

[10] Christoph Pflaum. Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation. *Institut fur Informatik: Technische Universitat Munchen*, pages 1–35, 1993.

[11] V.N. Temlyakov. On Approximate Recovery of Functions with Bounded Mixed Derivative. *Journal of Complexity*, 9(1):41–59, mar 1993.

[12] H.-J. Bungartz, M. Griebel, D. Röschke, and C. Zenger. Pointwise convergence of the combination technique for Laplace's equation. *East-West J. Numer. Math*, 2:21–45, 1994.

[13] U. Rüde. Multilevel, Extrapolation, and Sparse Grid Methods. In *Multigrid Methods IV*, number 1, pages 281–294. Birkhäuser Basel, Basel, 1994.

[14] Michael Griebel and Veronika Thurner. The efficient solution of fluid dynamics problems by the combination technique. *International Journal of Numerical Methods for Heat & Fluid Flow*, 5(3):251–269, mar 1995.

[15] William D. Henshaw. Automatic grid generation. *Acta Numerica*, 5:121, jan 1996.

[16] Christoph Pflaum. Convergence of the Combination Technique for Second-Order Elliptic Differential Equations. *SIAM Journal on Numerical Analysis*, 34(6):2431–2455, 1997.

[17] Hans-Joachim Bungartz and Thomas Dornseifer. Sparse Grids: Recent Developments for Elliptic Partial Differential Equations. In *Multigrid Methods V (Lecture Notes in Computational Science and Engineering Volume 3)*, pages 45–70. Springer Berlin Heidelberg, 1998.

[18] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids Thomas. *Numerical Algorithms*, 18(3/4):209–232, 1998.

[19] Michael Griebel. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing*, 61(2):151–179, jun 1998.

[20] Michael Griebel and Gerhard Zumbusch. Adaptive Sparse Grids for Hyperbolic Conservation Laws. In *Hyperbolic Problems: Theory, Numerics, Applications*, pages 411–422. Birkhäuser Basel, Basel, 1999.

[21] C Pflaum and A Zhou. Error analysis of the combination technique. *Numerische Mathematik*, 84(2):327–350, dec 1999.

[22] Boris Lastdrager, Barry Koren, and Jan Verwer. The Sparse-Grid Combination Technique Applied to Time-Dependent Advection Problems. In *Multigrid Methods VI*, volume 14, pages 143–149. sep 2000.

[23] Gerhard Zumbusch. Parallel Adaptively Refined Sparse Grids. In *Multigrid Methods VI*, pages 285–292. Springer Berlin Heidelberg, 2000.

[24] Boris Lastdrager, Barry Koren, and Jan Verwer. The sparse-grid combination technique applied to time-dependent advection problems. *Applied Numerical Mathematics*, 38(4):377–401, sep 2001.

[25] Markus Hegland. Additive Sparse Grid Fitting. In *In: Proceedings of the Fifth International Conference on Curves and Surfaces, Saint-Malo*, number February, pages 209–218, 2002.

[26] M Hegland. Adaptive sparse grids. *ANZIAM Journal*, 44(April):C335–C353, apr 2003.

[27] Jochen Garcke. Regression with the optimised combination technique. In *Proceedings of the 23rd international conference on Machine learning - ICML '06*, pages 321–328, New York, New York, USA, 2006. ACM Press.

[28] Jochen Garcke. A dimension adaptive sparse grid combination technique for machine learning. In Wayne Read, Jay W. Larson, and A. J. Roberts, editors, *Proceedings of the 13th Biennial Computational Techniques and Applications Conference, CTAC-2006*, volume 48, pages C725—-C740. 2007.

[29] Jochen Garcke. An optimised sparse grid combination technique for eigenproblems. *PAMM*, 7(1):1022301–1022302, dec 2007.

[30] Markus Hegland, Jochen Garcke, and Vivien Challis. The combination technique and some generalisations. *Linear Algebra and its Applications*, 420(2-3):249–275, jan 2007.

[31] Sebastian Franz, Fang Liu, Hans-görg Roos, Martin Stynes, and Aihui Zhou. The combination technique for a two-dimensional convection-diffusion problem with exponential layers. *Applications of Mathematics*, 54(3):203–223, jun 2009.

[32] Abhijeet Gaikwad and Ioane Muni Toke. GPU based sparse grid technique for solving multidimensional options pricing PDEs. In *Proceedings of the 2nd Workshop on High Performance Computational Finance - WHPCF '09*, pages 1–9, New York, New York, USA, 2009. ACM Press.

[33] Jochen Garcke and Markus Hegland. Fitting multidimensional data using gradient penalties and the sparse grid combination technique. *Computing*, 84(1-2):1–25, apr 2009.

[34] Mohamed S Ebeida, Roger L Davis, and Roland W Freund. A new fast hybrid adaptive grid generation technique for arbitrary two-dimensional domains. *Library*, (April):305–329, 2010.

[35] Jochen Garcke. Sparse Grids in a Nutshell. In Jochen Garcke and Michael Griebel, editors, *Sparse Grids and Applications*, volume 88 of *Lecture Notes in Computational Science and Engineering*, pages 57–80. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[36] Philipp Hupp. Hierarchization for the Sparse Grid Combination Technique. *CoRR Computing Research Repository*, aug 2013.

[37] Christoph Kowitz and Markus Hegland. The Sparse Grid Combination Technique for Computing Eigenvalues in Linear Gyrokinetics. *Procedia Computer Science*, 18:449–458, 2013.

[38] J.W. Larson, M Hegland, B Harding, S Roberts, L Stals, A.P. Rendell, P. Strazdins, M.M. Ali, C. Kowitz, R. Nobes, J. Southern, N. Wilson, M. Li, and Y. Oishi. Fault-Tolerant Grid-Based Solvers: Combining Concepts from Sparse Grids and MapReduce. *Procedia Computer Science*, 18:130–139, 2013.

[39] Christoph Reisinger. Analysis of linear difference schemes in the sparse grid combination technique. *IMA Journal of Numerical Analysis*, 33(2):544–581, apr 2013.

[40] Mario Heene, Christoph Kowitz, and Dirk Pflüger. Load Balancing for Massively Parallel Computations with the Sparse Grid Combination Technique. In *Advances in Parallel Computing (Volume 25: Parallel Computing: Accelerating Computational Science and Engineering (CSE)*, pages 574 – 583. 2014.

[41] J Erhel, Z Mghazli, and M Oumouni. An adaptive sparse grid method for elliptic PDEs with stochastic coefficients. *Computer Methods in Applied Mechanics and Engineering*, 297:392–407, dec 2015.

[42] Bastian Bohn, Jochen Garcke, and Michael Griebel. A sparse grid based method for generative dimensionality reduction of high-dimensional data. *Journal of Computational Physics*, 309:1–17, 2016.

[43] Martin Buhmann, F.-J. Delvos, and Walter Schempp. *Boolean Methods in Interpolation and Approximation*, volume 74. Longman Scientific and Technical, oct 1990.

[44] Michael Goodrich, Roberto Tamassia, and David Mount. *DATA STRUCTURES AND ALOGORITHMS IN C++*. John Wiley & Sons, 2007.