Tensor B-Spline Numerical Method for PDEs: a High Performance Approach

Inauguraldissertation

zur

Erlangung der Würde eines Doktors der Philosophie

vorgelegt der

Philosophisch-Naturwissenschaftlichen Fakultät

der Universität Basel

von

Dmytro Shulga

aus Ukraine

Basel, 2020

Abstract

Solutions of Partial Differential Equations (PDEs) form the basis of many mathematical models in physics and medicine. In this work, a novel Tensor B-spline methodology for numerical solutions of linear second-order PDEs is proposed. The methodology applies the B-spline signal processing framework and computational tensor algebra in order to construct high-performance numerical solvers for PDEs. The method allows high-order approximations, is mesh-free, matrix-free and computationally and memory efficient.

The first chapter introduces the main ideas of the Tensor B-spline method, depicts the main contributions of the thesis and outlines the thesis structure.

The second chapter provides an introduction to PDEs, reviews the numerical methods for solving PDEs, introduces splines and signal processing techniques with B-splines, and describes tensors and the computational tensor algebra.

The third chapter describes the principles of the Tensor B-spline methodology. The main aspects are 1) discretization of the PDE variational formulation via B-spline representation of the solution, the coefficients, and the source term, 2) introduction to the tensor B-spline kernels, 3) application of tensors and computational tensor algebra to the discretized variational formulation of the PDE, 4) tensor-based analysis of the problem structure, 5) derivation of the efficient computational techniques, and 6) efficient boundary processing and numerical integration procedures.

The fourth chapter describes 1) different computational strategies of the Tensor B-spline solver and an evaluation of their performance, 2) the application of the method to the forward problem of the Optical Diffusion Tomography and an extensive comparison with the state-of-the-art Finite Element Method on synthetic and real medical data, 3) high-performance multicore CPU- and GPU-based implementations, and 4) the solution of large-scale problems on hardware with limited memory resources.

Acknowledgments

I thank very much Prof. Patrick Hunziker for an opportunity to do my PhD study in his research group, for his extraordinary view on scientific and engineering problems that we were solving together, his important advices, helpful criticism and patience. I thank very much Dr. Oleksii Morozov for the vast amount of knowledge and inspiration he gave me, for his steady help, needful criticism, and friendship. I thank Prof. Volker Roth for being my faculty representative, for his advices, comments and help. I thank Prof. Michael Unser who kindly agreed to be a co-referee of the thesis committee. I thank Dr. Felix Friedrich for teaching me advanced programming skills. I thank Dr. Gabor Kosa for his comments and advises. I thank everyone I worked with in Patrick Hunziker's research group: Georgette B. Salieb-Beugelaar, Marc Wolf, Daniel Gonçalves, Maurice Mutro Nigo, Kegang Liu, Deborah Minnig, Xueya Wang, Timothée Martiel, Roman Lehner, and Kai Tisljar. I thank Elena Volosyuk for her help and support. I thank Susann Widmer for her help with English writing. I thank very much my mother Rymma, father Igor, my cousin Anna and my aunt Ludmila for their love and support.

I express my gratitude to the Swiss National Science Foundation for funding my work by Grant "Optical tomography: novel spline algorithms and application in mice and man, SNF Nr. 160178".

Contents

Notation

$\mathbb{N}$	set of natural numbers
$\mathbb{N}_0$	set of natural numbers including zero
$\mathbb{Z}$	set of integer numbers
$\mathbb{R}$	Euclidean space
$\mathbb{R}^d$	$d-$dimensional Euclidean space, $d \in \mathbb{N}$
$\mathbb{H}$	Hilbert space
$\mathbb{L}^2$	Lebesgue space of square integrable functions
$\mathbb{W}^{1,2}$	Sobolev space of $\mathbb{L}^2$-functions with weak $\mathbb{L}^2$-derivatives up to the 1-st order
a	scalar
$\mathbf{a} \in \mathbb{R}^N$	real vector of size N
$\mathbf{A} \in \mathbb{R}^{M\times N}$	real matrix of size $M \times N$
$\mathcal{A}_n^m \in \mathbb{R}^{M\times N}$	real tensor of size $M \times N$
$\mathcal{A}_{\mathbf{n}}^{\mathbf{m}} \in \mathbb{R}^{M_1\times..\times M_k\times N_1\times..\times N_l}$	real tensor of size $M_1 \times .. \times M_K \times N_1 \times .. \times N_L$, $K \in \mathbb{N}$, $L \in \mathbb{N}$
$f(x)$	univariate scalar function, $f : \mathbb{R} \to \mathbb{R}$
$g(\mathbf{x})$	multivariate scalar function, $g : \mathbb{R}^d \to \mathbb{R}$
$\mathbf{w}(\mathbf{x}) = (w_1(\mathbf{x}), ..., w_m(\mathbf{x}))$	multivariate vector function, $\mathbf{w} : \mathbb{R}^d \to \mathbb{R}^m$
$\frac{df}{dx}$	derivative of a univariate scalar function $f(x)$
$\frac{\partial g}{\partial x_i}$	partial derivative of a multivariate scalar function $g(\mathbf{x})$
$\nabla g(\mathbf{x}) = \left(\frac{\partial g}{\partial x_1}, \quad \cdots, \quad \frac{\partial g}{\partial x_d}\right)$	gradient of a multivariate scalar function $g(\mathbf{x})$
$\nabla^2 g(\mathbf{x}) = \nabla \cdot \nabla g(\mathbf{x}) = \frac{\partial^2 g}{\partial x_1^2} + \ldots + \frac{\partial^2 g}{\partial x_d^2}$	Laplacian of a multivariate scalar function $g(\mathbf{x})$
$\nabla \cdot \mathbf{u}(\mathbf{x}) = \frac{\partial u_1}{\partial x_1} + \ldots + \frac{\partial u_d}{\partial x_d}$	divergence of a multivariate vector function $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^d$
$\beta^n(x)$	univariate B-spline of degree $n \in \mathbb{N}_0$, $x \in \mathbb{R}$
$\beta^n_{k,h}(x) = \beta^n(x/h - k)$	univariate B-spline of degree n, attached to the k-th node ($k \in \mathbb{Z}$) of a regular grid with step size $h \in \mathbb{R}$
$\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x}) = \beta^n(\mathbf{x}/\mathbf{h} - \mathbf{k})$	multivariate B-spline of degree n, attached to the $\mathbf{k}$-th node ($\mathbf{k} \in \mathbb{Z}^d$) of a multidimensional regular grid with step size $\mathbf{h} \in \mathbb{R}^d$
$\mathbf{a}_1 \cdot \mathbf{a}_2$	scalar product of two vectors
$\langle f_1, f_2 \rangle$	scalar product of two functions
$*$	convolution

Chapter 1

Introduction

Partial Differential Equations (PDEs) play a significant role in many scientific, engineering, and medical applications. They describe physical phenomena in electrostatics, electrodynamics, optics, heat transfer, fluid dynamics, diffusion, solid mechanics, etc.

Various applications that require solutions of PDEs in a volume domain are of practical importance in particular for applications in medicine. Applications include modelling of light propagation (e.g. near-infrared light propagation in the tissue – the forward problem of Optical Diffusion Tomography (ODT) [1]), modelling of the electrical potential distribution (e.g. the forward problem of Electrical Impedance Tomography [2]), modelling of the current flow (e.g. the forward problem of Electrical Capacitance Tomography [3]), modelling of the electrical potential distribution (e.g. the forward problem of electroencephalography [4]), modelling of the heat transfer in the solid objects and living tissues (e.g. solving a bioheat equation [5]), fluid simulation (e.g. weather prediction [6]), and many others.

In most real-life cases, it is too difficult or impossible to obtain an analytical solution of a PDE. Therefore numerous numerical methods are employed to achieve an approximate solution with sufficient precision. Often finding a numerical solution of a PDE is a computationally intensive task. Therefore the development of fast and accurate PDE solvers is an essential field of research.

1.1 Tensor B-spline Methodology for PDEs

1.1.1 Motivation

This work was motivated by the practical need to solve large PDEs for biomedical applications and the intuition that the characteristics of tensor- and B-spline algorithms may fit well to the data structure and algorithmic requirements encountered in solving PDEs. In particular, we hoped that such a method:

1. Benefits from a signal processing framework that allows accurate transition from the continuous nature of physical problems to discrete formulations suited for computing, ideally for each, the coefficients, source and solution of a PDE.

2. Allows to employ high-degree basis functions easier in comparison to the state-of-the-art high-order Finite Element Methods. This high-degree basis should be more computationally efficient than one in FEM.

3. Does not require domain mesh construction, which is a difficult and expensive procedure, but rather is based on a simple regular grid. At the same time the method should allow to handle arbitrarily-shaped domains.

4. Allows to solve large-scale problems (up to billion unknowns) on the off-the-shelf computer workstations, is scalable to high-performance computing architectures.

5. Can solve PDE of different type, and can be incorporated in the state-of-the-art medical imaging frameworks as an alternative to FEM solvers.

1.1.2 B-Splines

B-splines are a perfect candidate for a powerful signal processing instrument that can be incorporated in the numerical solver of PDEs. B-splines have been used for computer graphics and computer-aided design for a long time, and they have been applied to signal and image processing and reconstruction as well. B-splines have been successfully applied to a wide range of applications including signal interpolation, approximation, smoothing, resampling [7], multidimensional reconstruction [8], multiresolution processing [7], parametric contour representation [9], etc. B-splines provide high-quality representation of an underlying signal, accurate differentiation and integration, and multi-scale transformations via a set of efficient filtering-based techniques [10, 11].

1.1.3 Tensors

Tensors are the instrument that can help to exploit the structure of a discrete PDE formulation represented with tensor-product B-splines. Tensors are widely used in physics and increasingly used in signal processing for operations on multidimensional data and data analysis [12]. Tensor decompositions are an important tool for revealing the hidden components in the data [13], machine learning [14], etc. Compared to matrices, such tensors allow us to represent multidimensional structures more compactly and naturally [13]. To some extent, tensors can help to overcome the computational difficulties of large-scale problems. The computational tensor algebra framework [15, 16] helps to preserve data structure and coherence and allows us to derive efficient solving algorithms to solve multidimensional tensor problems. The use of the tensor structure of a multidimensional signal reconstruction problem formulated in terms of B-splines enables the computability of very large problems on standard hardware [8].

1.1.4 Tensor B-spline Method

In this thesis, an accurate and efficient Tensor B-spline numerical method for PDEs is developed. The method can be applied to various PDE types in the context of different applications.

Fig. 1.1 highlights the main properties that make Tensor B-spline method a promising candidate for numerical solutions of PDEs. B-splines naturally link continuous and discrete domains [7] (Fig. 1.1 (a)), while providing excellent approximations of coefficients, sources, and solutions. The combination of B-splines and tensor algebra preserves the intrinsic structure of the problem and enforces both sparsity, separability and decomposed tensor structure in a very natural way (Fig. 1.1 (b)). At the same time, it makes it easy to design highly efficient parallel and matrix-free algorithms. As a result, highly accurate and efficient solutions can be obtained (Fig. 1.1 (c)).

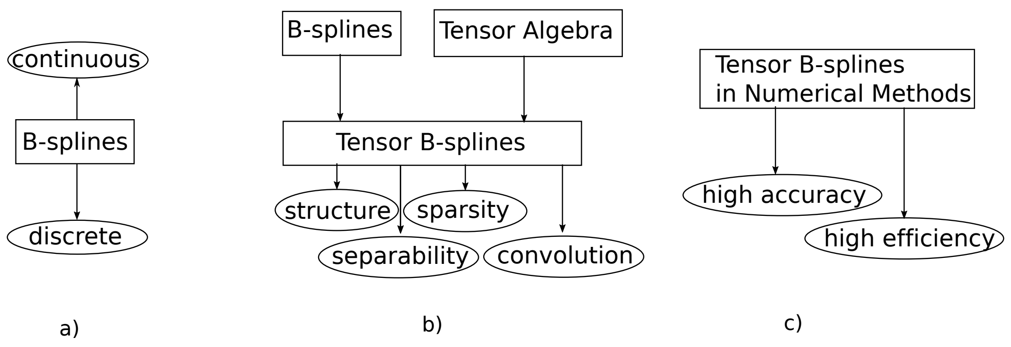


Figure 1.1: Diagram of Tensor B-spline PDE solver main features.

The Tensor B-spline method described in this thesis has many appealing properties of a "generic" numerical PDE solver: 1) it provides us with accurate and flexible discretizations of PDE coefficients, sources, and solution, 2) it allows for efficient integration strategies, 3) it makes it relatively simple to develop fast and memory-efficient algorithms, 4) the mathematical elegance of computational tensor algebra [15] leads to natural and transparent models.

Tensor B-spline method can provide many advantages compared to the widely used FEM. FEM uses Lagrange polynomials that tend to oscillate, relies on mesh that difficult to generate, uses sparse matrices that are inefficient for high-performance computations, consumes significant amount of memory for large-scale problems or when high-order basis is used. Tensor B-spline method provides accurate high-degree B-spline representations, efficient tensor-based algorithms, requires no meshing of a domain. The Tensor B-spline methodology allows convergence in a small number of iterations e.g., using the conjugate gradient (CG) or the multigrid (MG) solver, and can be highly memory-efficient thanks to in-place computations enabled by a tensor structure of the problem. Tensor B-spline method overcomes the integration difficulties on the domain boundary via the use of the Divergence Theorem and B-spline properties.

The benefits of combining tensors and B-splines have been shown for solving diffusion PDEs in Optical Diffusion Tomography [17, 18] resulting in a generic numerical method for PDEs [19].

1.2 Contributions of the Thesis

The main contributions of this thesis are:

1. A novel Tensor B-spline numerical method for PDEs. We show how an application of B-spline functions and algorithms together with tensors and computational tensor algebra results in efficient numerical solutions of PDEs. The method allows high-order approximations, is mesh-free, supports domains with complex boundaries and provides efficient matrix-free parallel algorithms for numerical solutions of PDEs.

2. We propose a pervasive usage of a B-spline signal processing framework for numerical solutions of PDEs in order to benefit from efficient representations and filter-based processing of PDE coefficients, source and solution. We proposed an efficient integration method in the proximity of the PDE boundaries. The integration method is based on the Divergence Theorem and the properties of B-splines.

3. The decomposed tensor structure of the B-spline-based PDE formulation is shown. The tensor B-spline kernels are introduces, and an elegant method of optimization of computations via tensor algebra approach is presented.

4. We apply the Tensor B-spline method to the forward problem of the ODT, perform an extensive comparison with the state-of-the-art FEM on synthetic and real medical data

and show the method advantages. We show how high-performance multicore CPU- and GPU-based implementations of the Tensor B-spline solver can be constructed, and large-scale solutions with limited memory resources can be obtained.

The results were published in three papers:

1. D. Shulga, O. Morozov, and P. Hunziker “A Tensor B-spline Approach for Solving the Diffusion PDE with Application to Optical Diffusion Tomography”, IEEE Transactions on Medical Imaging 36 (4), 972-982, 2017.
2. D. Shulga, O. Morozov, and P. Hunziker “Solving 3-D PDEs by Tensor B-Spline Methodology: A High Performance Approach Applied to Optical Diffusion Tomography”, IEEE Transactions on Medical Imaging 37 (9), 2115-2125, 2018.
3. D. Shulga, O. Morozov, V. Roth, F. Friedrich, and P. Hunziker “Tensor B-Spline Numerical Methods for PDEs: a High-Performance Alternative to FEM”, preprint arXiv: 1904.03057, 2019.

1.3 Thesis structure

The thesis consists of four chapters. In the second chapter, the thesis background is presented with an introduction to PDEs, review of the numerical methods of PDEs, introduction to splines and B-spline signal processing, and finally, an introduction to tensors and computational tensor algebra.

The third chapter presents the Tensor B-spline numerical method for PDEs. This chapter describes the method’s motivation, begins with a PDE weak formulation, applies B-spline basis functions, and constructs a tensor-based formulation of the problem. Then, the analysis of the formulation is presented with examples of the tensor structures in two- and three-dimensions. Then an efficient integration method is given, as well as the application of different boundary conditions. The chapter is concluded with the method summary and discussion.

The fourth chapter presents the Tensor B-spline method implementation, evaluation, and comparison with the state-of-the-art FEM. Efficient computational strategies are presented. An extensive comparison with FEM is performed using an example of the Diffusion PDE in the context of the ODT forward problem solution. The results of the comparison are presented for two- and three-dimensional cases. A high-performance evaluation of large-scale system operators and a large-scale solution on a heterogeneous workstation are presented. The chapter is concluded with a discussion.

The thesis is concluded with a summary.

Chapter 2

Background

2.1 Introduction

In this chapter, we provide an introduction to Partial Differential Equations (PDEs), describe some commonly used PDEs and different boundary conditions. Then we review state-of-the-art numerical methods for PDEs and discuss their advantages and disadvantages. We overview splines, their properties, B-spline interpolation, and approximation via digital filtering. We give an introduction to tensors and computational tensor algebra. Finally, we discuss the modern numerical methods for PDEs and the potential of B-splines and tensors in application to the numerical solutions of PDEs.

2.2 Partial Differential Equations

Partial Differential Equations (PDEs) play an important role in many disciplines, including physics, engineering, biology, medicine, etc. PDEs describe physical phenomena in electrostatics, electrodynamics, optics, heat transfer, fluid dynamics, diffusion, elasticity, sound propagation.

The unknown function in a PDE usually represents a physical quantity, often of spatially continuous nature, and the derivatives represent its rates of change. A PDE for a real multivariate function $\varphi(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$

$$\xi\left(\mathbf{x}, \varphi, \frac{\partial \varphi}{\partial x_1}, ..., \frac{\partial \varphi}{\partial x_d}, \frac{\partial^2 \varphi}{\partial x_1 \partial x_1}, ..., \frac{\partial^2 \varphi}{\partial x_1 \partial x_d}, ...\right) = 0, \ \mathbf{x} \in \Omega, \tag{2.1}$$

links the function $\varphi(\mathbf{x})$ and its partial derivatives in some domain $\Omega \subset \mathbb{R}^d$ (Fig. 3.12). In order to obtain a unique solution of the PDE (2.1), boundary conditions (BC) have to be provided to define the function behavior on the domain boundary $\partial\Omega$.

In cases when the PDE unknown function depends on time t, equation 2.1 also includes time derivatives $\partial\varphi/\partial t$, $\partial^2\varphi/\partial t^2$, ...

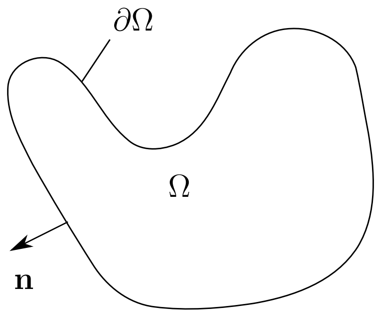


Figure 2.1: An example of a two-dimensional domain $\Omega \subset \mathbb{R}^2$ with a smooth boundary $\partial\Omega$; $\mathbf{n} \in \mathbb{R}^2$ is an outward normal to the domain boundary.

A PDE is called linear if $\xi(\ldots)$ is a linear function of $\varphi(\mathbf{x})$ and its derivatives. This work considers linear second-order equations in one-, two-, and three- dimensions ($d = 1, 2, 3$). The function $\varphi(\mathbf{x})$ is assumed to be twice continuously differentiable in the domain Ω. The boundary $\partial\Omega$ is assumed to be Lipschitz-continuous [20].

2.2.1 Boundary Conditions and Initial Values

A PDE governs a family of possible solutions; a particular solution is defined by the auxiliary conditions like boundary conditions and initial values. A PDE can be coupled with boundary conditions of different types. For $\mathbf{x} \in \partial\Omega$, the following boundary conditions can be applied [21]:

1. Dirichlet BC $\varphi(\mathbf{x}) = g(\mathbf{x})$ (non-homogeneous) and $\varphi(\mathbf{x}) = 0$ (homogeneous), specifies the value of the function on the boundary;

2. Neumann BC $\nabla\varphi(\mathbf{x}) \cdot \mathbf{n} = g(\mathbf{x})$, specifies the value of the normal derivative $\nabla\varphi(\mathbf{x}) \cdot \mathbf{n}$ on the boundary;

3. Robin BC $\alpha(\mathbf{x})(\nabla\varphi(\mathbf{x}) \cdot \mathbf{n}) + \beta(\mathbf{x})\varphi(\mathbf{x}) = g(\mathbf{x})$, specifies the combination of the function value and the normal derivative on the boundary;

4. Cauchy BC $\varphi(\mathbf{x}) = a(\mathbf{x})$, $\nabla\varphi(\mathbf{x}) \cdot \mathbf{n} = b(\mathbf{x})$ specifies separately the values of the function and its normal derivative on the boundary;

5. Mixed BC specifies different boundary conditions on disjoint parts of the boundary.

For time-dependent problems, the auxiliary conditions consist of the initial value of the function on the time boundary ($t = 0$) combined with any of the boundary conditions listed above.

2.2.2 Green's Function

The impulse response of the PDE (2.1) with subject to some specified boundary conditions is called the Green's function [22]. The Green's function $G(\mathbf{x}, \mathbf{s})$, $\mathbf{x}, \mathbf{s} \in \mathbb{R}^d$ of a linear differential operator $L = L(\mathbf{x})$ acting on the collection of distributions over $\Omega \in \mathbb{R}^d$, is the solution (also called the fundamental solution for L) of

$$LG(\mathbf{x}, \mathbf{s}) = \delta(\mathbf{x} - \mathbf{s}), \tag{2.2}$$

where $\delta(\mathbf{x})$ is the Dirac delta function. If the Green's function $G(\mathbf{x}, \mathbf{s})$ can be determined for a given operator L, then the solution of the equation $L\varphi(\mathbf{x}) = g(\mathbf{x})$ can be determined as the convolution of $G(\mathbf{x}, \mathbf{s})$ with the right hand side $g(\mathbf{x})$:

$$\varphi(\mathbf{x}) = G * g = \int_\Omega G(\mathbf{x}, \mathbf{s}) g(\mathbf{s}) d\mathbf{s}. \tag{2.3}$$

2.2.3 Classification of Second-Order Linear PDEs

Let us consider the general form of a second-order linear PDE in two dimensions, $\mathbf{x} \in \mathbb{R}^2$:

$$a(\mathbf{x})\frac{\partial^2\varphi}{\partial x_1^2} + b(\mathbf{x})\frac{\partial^2\varphi}{\partial x_1 \partial x_2} + c(\mathbf{x})\frac{\partial^2\varphi}{\partial x_2^2} + d(\mathbf{x})\frac{\partial\varphi}{\partial x_1} + e(\mathbf{x})\frac{\partial\varphi}{\partial x_2} + f(\mathbf{x})\varphi(\mathbf{x}) = g(\mathbf{x}). \tag{2.4}$$

The PDE properties depend on the relative magnitudes of the coefficients $a(\mathbf{x}), b(\mathbf{x})$, and $c(\mathbf{x})$. Depending on the sign of the discriminant $b(\mathbf{x})^2 - 4a(\mathbf{x})c(\mathbf{x})$ [21]:

1. if $b(\mathbf{x})^2 - 4a(\mathbf{x})c(\mathbf{x}) < 0$, the PDE is called elliptic,
2. if $b(\mathbf{x})^2 - 4a(\mathbf{x})c(\mathbf{x}) = 0$, the PDE is called parabolic,
3. if $b(\mathbf{x})^2 - 4a(\mathbf{x})c(\mathbf{x}) > 0$, the PDE is called hyperbolic.

A given PDE may be of one type at a specific point $\mathbf{x}'$, and of another type at some other point $\mathbf{x}''$.

From a physical viewpoint, elliptic, parabolic, and hyperbolic PDEs represent a steady state or equilibrium processes, time-dependent diffusion processes, and wave propagation correspondingly. Elliptic equations describe systems in their minimal energy state. Parabolic equations describe evolutionary phenomena that lead to a steady state described by an elliptic equation. Hyperbolic equations often model the transport of some physical quantity, e.g. mass transfer in fluids [21].

2.2.4 Examples of Second Order Linear PDEs

Here we review second order linear PDEs that describe some frequent physical problems. These include wave equation, diffusion equation, heat equation, Poisson and Laplace equations, and convection-diffusion equation.

Wave Equation

For a time-dependent pressure field $\varphi(\mathbf{x}, t)$, the wave equation [21] is defined as

$$\frac{\partial^2\varphi(\mathbf{x}, t)}{\partial t^2} = c^2\nabla^2\varphi(\mathbf{x}, t) + f(\mathbf{x}, t), \ \mathbf{x} \in \Omega, \tag{2.5}$$

where c is the speed of the wave propagation in the media, and $f(\mathbf{x}, t)$ is the force term. The wave equation is a hyperbolic equation that arises in the problems of vibrations, electrostatics, electromagnetics, fluid dynamics, acoustics, etc.

Diffusion Equation

The linear diffusion equation [21] has the following form

$$\frac{\partial\varphi(\mathbf{x}, t)}{\partial t} = \nabla \cdot [D(\mathbf{x})\nabla\varphi(\mathbf{x}, t)], \ \mathbf{x} \in \Omega, \tag{2.6}$$

where $\varphi(\mathbf{x}, t)$ is the density of the diffusing material, and $D(\mathbf{x})$ is the diffusion coefficient. The diffusion equation applies to problems in mass diffusion, momentum diffusion, heat diffusion, etc. In physics, the Diffusion PDE describes the behavior of the collective motion of micro-particles in a material, in optics, it describes the light propagation in a turbid media.

Heat Equation

The linear heat equation [23], which is a special case of the diffusion equation, is defined as

$$\rho c_p \frac{\partial \varphi(\mathbf{x},t)}{\partial t} - \nabla \cdot [k(\mathbf{x})\nabla\varphi(\mathbf{x})] = g(\mathbf{x}),\ \mathbf{x} \in \Omega, \tag{2.7}$$

where $\varphi(\mathbf{x},t)$ is the temperature, $k(\mathbf{x})$ is the thermal conductivity, ρ is the density, c_p is the heat capacity, and $g(\mathbf{x})$ is the rate at which the energy is generated per unit volume of the medium. In a steady state ($\partial\varphi(\mathbf{x},t)/\partial t = 0$), (2.7) reduces to

$$-\nabla \cdot [k(\mathbf{x})\nabla\varphi(\mathbf{x})] = g(\mathbf{x}),\ \mathbf{x} \in \Omega. \tag{2.8}$$

Poisson and Laplace Equations

A special case of the diffusion equation is the Laplace equation [21]

$$\nabla^2\varphi(\mathbf{x}) = 0,\ \mathbf{x} \in \Omega. \tag{2.9}$$

The Laplace equation applies to steady-state problems in ideal fluid flow, mass diffusion, heat transfer, electrostatics, etc.

Another particular case of the diffusion equation is the Poisson equation [21]

$$\nabla^2\varphi(\mathbf{x}) = q(\mathbf{x}),\ \mathbf{x} \in \Omega, \tag{2.10}$$

which is an inhomogeneous form of the Laplace equation.

Convection-diffusion equation

The convection-diffusion equation [21] is a combination of the diffusion and convection equations. The convection-diffusion equation is defined as

$$\frac{\partial \varphi(\mathbf{x},t)}{\partial t} = \nabla \cdot [D(\mathbf{x})\nabla\varphi(\mathbf{x},t)] - \nabla \cdot (\mathbf{v}\varphi(\mathbf{x},t)) + g(\mathbf{x}), \tag{2.11}$$

where $\varphi(\mathbf{x},t)$ specifies the concentration for mass transfer, $D(\mathbf{x})$ is the diffusion coefficient, $\mathbf{v}$ is the velocity field that the quantity is moving with, $g(\mathbf{x})$ describes sources or sinks of the $\varphi(\mathbf{x},t)$.

2.3 Review of Numerical Methods for PDEs

In most real-life cases, especially when a domain boundary has a complex geometry, it is too difficult or impossible to solve a PDE analytically. Therefore, numerous numerical methods are employed in order to get an approximate solution with sufficient precision. For many problems finding a numerical solution of a PDE might be a very computationally intensive task. Each numerical method for PDEs has its own strengths and weaknesses and may be better suited for one problem and less suited for others.

Sobolev Spaces

Consider $\mathbb{V}$ be a vector space over a field $\mathbb{R}$ equipped with a scalar (inner) product. The scalar product defines a norm $||f||_{\mathbb{V}} = (f \cdot f)^{\frac{1}{2}},\ f \in \mathbb{V}$. A space $\mathbb{V}$ is complete (with respect to the given scalar product) if every Cauchy sequence $f_n \in \mathbb{V},\ n \in \mathbb{N}$ converges to an element $f \in \mathbb{V}$, i.e. $\lim_{n\to\infty} ||f_n - f|| = 0$. A complete vector-space $\mathbb{V}$ is called a Hilbert space $\mathbb{H}$.

Consider a bounded domain $\Omega \subset \mathbb{R}^d$. A Lebesgue space $\mathbb{L}^2(\Omega)$ contains all functions $v(x)$, $\mathbf{x} \in \mathbb{R}^d$ such that

$$\int_\Omega v^2(\mathbf{x}) d\mathbf{x} < \infty. \tag{2.12}$$

For $u(\mathbf{x}),\ v(\mathbf{x}) \in \mathbb{L}^2(\Omega)$ the scalar product is defined as

$$\langle u, v \rangle_{\mathbb{L}^2(\Omega)} = \int_\Omega u(\mathbf{x}) v(\mathbf{x}) d\mathbf{x}, \tag{2.13}$$

and the corresponding norm is defined as

$$||u(x)||_{\mathbb{L}^2(\Omega)} = \sqrt{\langle u, u \rangle} = \sqrt{\int_\Omega u^2(\mathbf{x}) d\mathbf{x}}. \tag{2.14}$$

Sobolev space $\mathbb{W}^{1,2}(\Omega)$ contains all functions $u(\mathbf{x})$ such that $u(\mathbf{x}) \in \mathbb{L}^2(\Omega)$ and $\partial u(\mathbf{x})/\partial x_i \in \mathbb{L}^2(\Omega)$. Functions in $\mathbb{W}^{1,2}(\Omega)$ do not have to be differentiable at every point, and can be continuous with piecewise continuous partial derivatives in the domain Ω.

The scalar-product of functions $u(\mathbf{x}),\ v(\mathbf{x}) \in \mathbb{W}^{1,2}(\Omega)$ is defined as

$$\langle u, v \rangle_{\mathbb{W}^{1,2}(\Omega)} = \int_\Omega u(\mathbf{x}) v(\mathbf{x}) d\mathbf{x} + \int_\Omega \nabla u(\mathbf{x}) \cdot \nabla v(\mathbf{x}) d\mathbf{x}, \tag{2.15}$$

and the norm is defined as

$$||u(\mathbf{x})||_{\mathbb{W}^{1,2}(\Omega)} = \sqrt{\int_\Omega u^2(\mathbf{x}) d\mathbf{x} + \int_\Omega |\nabla u(\mathbf{x})|^2 d\mathbf{x}}. \tag{2.16}$$

We assume that the domain boundary $\partial\Omega$ is Lipschitz continuous, meaning that such boundary is continuous and does not have to be differentiable at each point [20]. For example, closed polygonal boundaries satisfy this property.

Smoothness

The smoothness of a function is defined by the number of continuous derivatives it has. A real function $\varphi(x)$, $x \in \mathbb{R}$ is of class C^k if it has continuous derivatives $\varphi'(x)$, $\varphi''(x)$, ... $\varphi^{(k)}(x)$. A function is of class C^∞, or smooth, if it has continuous derivatives of all orders.

2.3.1 Finite Difference Method

The finite difference method (FDM) [24] approximates the differential operator by a finite difference.

Consider a C^∞ real function $\varphi(x)$, $x \in \mathbb{R}$ in the neighbourhood of x_0. For $\Delta x = x - x_0, \Delta x > 0$ the Taylor series of $\varphi(x)$ is defined as

$$\varphi(x_0 + \Delta x) = \varphi(x_0) + \frac{\partial \varphi(x_0)}{\partial x} \Delta x + \frac{1}{2} \frac{\partial^2 \varphi(x_0)}{\partial x^2} (\Delta x)^2 + \dots. \tag{2.17}$$

Let $\varphi(x)$ be a function of class C^2. Using the first two terms in the expansion (2.17), we obtain

$$\varphi(x_0 + \Delta x) = \varphi(x_0) + \frac{\partial \varphi(x_0)}{\partial x} \Delta x + O(\Delta x^2), \tag{2.18}$$

where $O(\Delta x^2)$ designates the approximation error proportional to Δx^2. Thus, an approximation of the first derivative can be obtained as

$$\frac{\partial \varphi}{\partial x}(x_0) \approx \frac{1}{\Delta x} [\varphi(x_0 + \Delta x) - \varphi(x_0)] \tag{2.19}$$

for a sufficiently small Δx. This approximation is known as forward difference. Similarly, a backward and a central difference-based approximation can be obtained:

$$\varphi'(x_0) \approx \frac{1}{\Delta x}\left[\varphi(x_0) - \varphi(x_0 - \Delta x)\right], \tag{2.20}$$

$$\varphi'(x_0) \approx \frac{1}{2\Delta x}\left[\varphi(x_0 + \Delta x) - \varphi(x_0 - \Delta x))\right]. \tag{2.21}$$

An approximation of the second derivative can be determined as

$$\begin{aligned} \varphi''(x_0) \approx \frac{1}{\Delta x}\left(\varphi'(x_0+\Delta x) - \varphi'(x_0 - \Delta x)\right) = \\ \frac{1}{\Delta x}\left(\frac{\varphi(x_0+\Delta x)-\varphi(x_0)}{\Delta x} - \frac{\varphi(x_0)-\varphi(x_0-\Delta x)}{\Delta x}\right) = \\ \frac{1}{\Delta x^2}\left(\varphi(x_0+\Delta x) - 2\varphi(x_0) + \varphi(x_0-\Delta x)\right). \end{aligned} \tag{2.22}$$

We consider the Poisson equation in two dimensions

$$-\nabla^2\varphi(\mathbf{x}) = g(\mathbf{x}),\ \mathbf{x} \in \Omega \subset \mathbb{R}^2, \tag{2.23}$$

with defined boundary conditions on $\partial\Omega$. In the FDM a regular rectangular grid with a grid step $\mathbf{h} = (h_1, h_2)$ for each dimension is constructed as a tensor product of grids $\{(x_1)_i = x_1^0 + (i-1)h_1\}_{i=1}^{I}$ and $\{(x_2)_j = x_2^0 + (j-1)h_2\}_{j=1}^{J}$. The point $\mathbf{x}_0 = (x_1^0, x_2^0)$ is the origin of the grid. An example of an FDM grid is shown in Fig. 2.2. Domain nodes $\Omega^e = ((x_1)_i, (x_2)_j) \in \Omega$ and boundary nodes $\partial\Omega^e = ((x_1)_i, (x_2)_j) \in \partial\Omega$. A continuous function $\varphi(\mathbf{x})$ is sampled at the grid nodes $\varphi((x_1)_i, (x_2)_j) = \varphi_{i,j}$.

Using (2.22), the Laplacian of $\varphi(\mathbf{x})$ can be approximated as

$$\nabla_h^2\hat{\varphi}_{i,j} = \frac{\hat{\varphi}_{i+1,j} - 2\hat{\varphi}_{i,j} + \hat{\varphi}_{i-1,j}}{h_1^2} + \frac{\hat{\varphi}_{i,j+1} - 2\hat{\varphi}_{i,j} + \hat{\varphi}_{i,j-1}}{h_2^2} \tag{2.24}$$

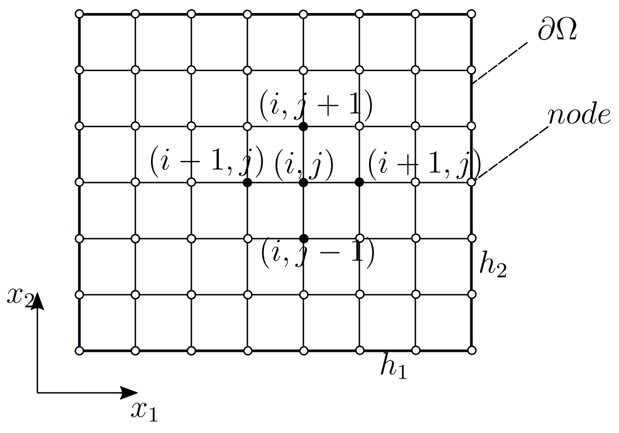


Figure 2.2: An example of a two-dimensional FDM grid for a domain Ω; black nodes depict the stencil corresponding to (2.24).

By setting $h = h_1 = h_2$ we get

$$-\nabla_h^2\hat{\varphi}_{i,j} = \frac{1}{h^2}(4\hat{\varphi}_{i,j} - \hat{\varphi}_{i+1,j} - \hat{\varphi}_{i-1,j} - \hat{\varphi}_{i,j+1} - \hat{\varphi}_{i,j-1}), \tag{2.25}$$

that corresponds to a stencil (Fig. 2.2)

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix}. \tag{2.26}$$

The right-hand side is sampled at node values $g_{i,j} = g((x_1)_i, (x_2)_j)$. The resulting system of equations

$$-\nabla_h^2 \hat{\varphi}_{i,j} = g_{i,j}, \ i = 1, 2, \dots I, \ j = 1, 2, \dots J, \tag{2.27}$$

has to be solved with an application of appropriate boundary conditions. In a matrix-vector form, it is represented as

$$\mathbf{Af} = \mathbf{g}, \tag{2.28}$$

where $\mathbf{A} \in \mathbb{R}^{IJ \times IJ}$, $\mathbf{f} \in \mathbb{R}^{IJ}$, and $\mathbf{g} \in \mathbb{R}^{IJ}$.

In order to apply the Dirichlet BC the right-hand side of (2.28) is modified, and the system of equations is solved for interior nodes only. The application of Neumann BC requires an introduction of ghost points. A detailed description of the application of boundary conditions in FDM can be found in the corresponding literature [24].

The FDM method is simple and easy to program. However, a major drawback of the method is its lack of flexibility: the FDM requires uniform grid, C^2-smoothness of the solution $\varphi(\mathbf{x})$ and it is problematic to impose boundary conditions.

2.3.2 Finite Element Method

The Finite Element Method (FEM) is a very popular and widely used numerical method for solving PDEs [25, 26]. The FEM provides high flexibility in the handling of arbitrarily shaped domains and yields high approximation quality, especially in its *hp*-variant [27].

Consider the Poisson equation with homogeneous Dirichlet boundary conditions:

$$-\nabla^2 \varphi(\mathbf{x}) = g(\mathbf{x}), \ \mathbf{x} \in \Omega, \tag{2.29}$$

$$\varphi(\mathbf{x}) = 0, \ \mathbf{x} \in \partial\Omega \tag{2.30}$$

on a domain $\Omega \subset \mathbb{R}^d$, where $g(\mathbf{x})$ is a known source function. The unknown function $\varphi(\mathbf{x})$ can be approximated by an expansion

$$\varphi(\mathbf{x}) \approx \hat{\varphi}(\mathbf{x}) = \sum_{k \in \mathbb{Z}} c_k \eta_k(\mathbf{x}), \tag{2.31}$$

where $\eta_k(\mathbf{x})$ are linearly-independent basis functions. Let the basis functions satisfy the boundary condition $\eta_k(\mathbf{x}) = 0, \ \mathbf{x} \in \partial\Omega$. The residual $R_\Omega(\mathbf{x}) = \nabla^2 \hat{\varphi}(\mathbf{x}) + g(\mathbf{x})$ is minimized over the domain Ω. According to the method of weighted residuals

$$\int_\Omega \psi_l(\mathbf{x}) R_\Omega(\mathbf{x}) d\mathbf{x} = \int_\Omega \psi_l(\mathbf{x})(\nabla^2 \hat{\varphi}(\mathbf{x}) + g(\mathbf{x})) d\mathbf{x} = 0, \ l \in \mathbb{Z}. \tag{2.32}$$

The Ritz-Galerkin method [25] assigns the weight functions to be equal to basis functions $\psi_l(\mathbf{x}) = \eta_l(\mathbf{x})$, that is equivalent to the least squares method:

$$\int_\Omega \eta_l(\mathbf{x}) \left(\nabla^2 \sum_{k \in \mathbb{R}} c_k \eta_k(\mathbf{x}) + g(\mathbf{x}) \right) d\mathbf{x} = 0, \tag{2.33}$$

$$\sum_{k \in \mathbb{R}} c_k \int_\Omega \eta_l(\mathbf{x}) \nabla^2 \eta_k(\mathbf{x}) d\mathbf{x} = -\int_\Omega g(\mathbf{x}) \eta_l(\mathbf{x}) d\mathbf{x}. \tag{2.34}$$

After integration by parts we get

$$\sum_{k \in \mathbb{R}} c_k \int_\Omega \nabla \eta_k(\mathbf{x}) \nabla \eta_l(\mathbf{x}) d\mathbf{x} = \int_\Omega g(\mathbf{x}) \eta_l(\mathbf{x}) d\mathbf{x}, \tag{2.35}$$

$$\eta_k(\mathbf{x}) = 0, \eta_l(\mathbf{x}) = 0, \ \mathbf{x} \in \partial\Omega. \tag{2.36}$$

The main idea of FEM is to split the domain Ω into a set of non-overlapping sub-domains (called elements) Ω^e and define basis functions $\eta_k(\mathbf{x})$ piecewisely within each element Ω^e. In two-dimensions the elements are usually represented by triangles, in three-dimensions by tetrahedrons. More complex geometrical object are also possible. Consider a domain Ω which is approximated by a finite set $\Upsilon = \{\Omega^e\}_{e=1}^{E}$ of non-overlapping elements (triangles) Ω^e, such that a domain approximation $\widetilde{\Omega} = \cup_{\Omega^e \in \Upsilon} \Omega^e$. The domain boundary $\partial\Omega$ is approximated by a polygon $\widetilde{\partial\Omega} = \cup_{\partial\Omega^e \in \Upsilon' \subset \Upsilon} \partial\Omega^e$. An example of a two-dimensional conforming mesh with nodes at vertices of triangles of a domain Ω is shown in Fig. 2.3.

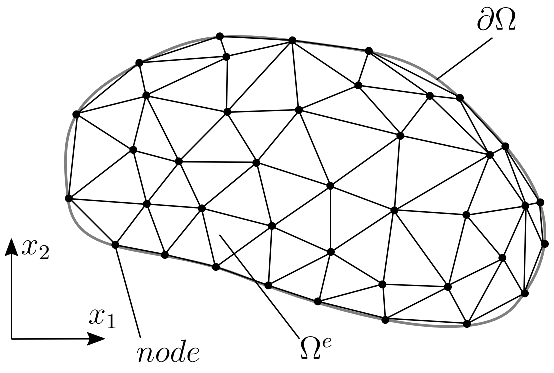


Figure 2.3: An example of a two-dimensional mesh of a domain Ω.

After the domain meshing we obtain:

$$\sum_{k \in \mathbb{Z}} c_k \sum_{e=1}^{E} \int_{\Omega^e} \nabla \eta_k(\mathbf{x}) \nabla \eta_l(\mathbf{x}) d\mathbf{x} = \sum_{e=1}^{E} \int_{\Omega^e} g(\mathbf{x}) \eta_l(\mathbf{x}) d\mathbf{x}, \tag{2.37}$$

$$\eta_k(\mathbf{x}) = 0, \eta_l(\mathbf{x}) = 0, \;\; \mathbf{x} \in \partial\Omega. \tag{2.38}$$

The equation (2.37) boils down to the system of linear equations

$$\mathbf{A}\mathbf{c} = \mathbf{b}, \tag{2.39}$$

where

$$\mathbf{A} = \sum_{e=1}^{E} \int_{\Omega^e} \nabla \eta_k(\mathbf{x}) \nabla \eta_l(\mathbf{x}) d\mathbf{x}, \;\; \mathbf{A} \in \mathbb{R}^{K \times K}, \tag{2.40}$$

$$\mathbf{b} = \sum_{e=1}^{E} \int_{\Omega^e} g(\mathbf{x}) \eta_l(\mathbf{x}) d\mathbf{x}, \;\; \mathbf{b} \in \mathbb{R}^{K}. \tag{2.41}$$

Fig. 2.4 depicts an example of a two-dimensional basis function and an approximation of a function in a FEM basis. Note that the basis function is composed of local basis functions. A Lagrange polynomial of degree p is used as a local basis function, in the linear case ($p = 1$)

$$\eta_i^e = \alpha_i^e + \beta_i^e x + \gamma_i^e y. \tag{2.42}$$

The coefficients α, β, and γ are generally dependent on a mesh geometry and have to be determined.

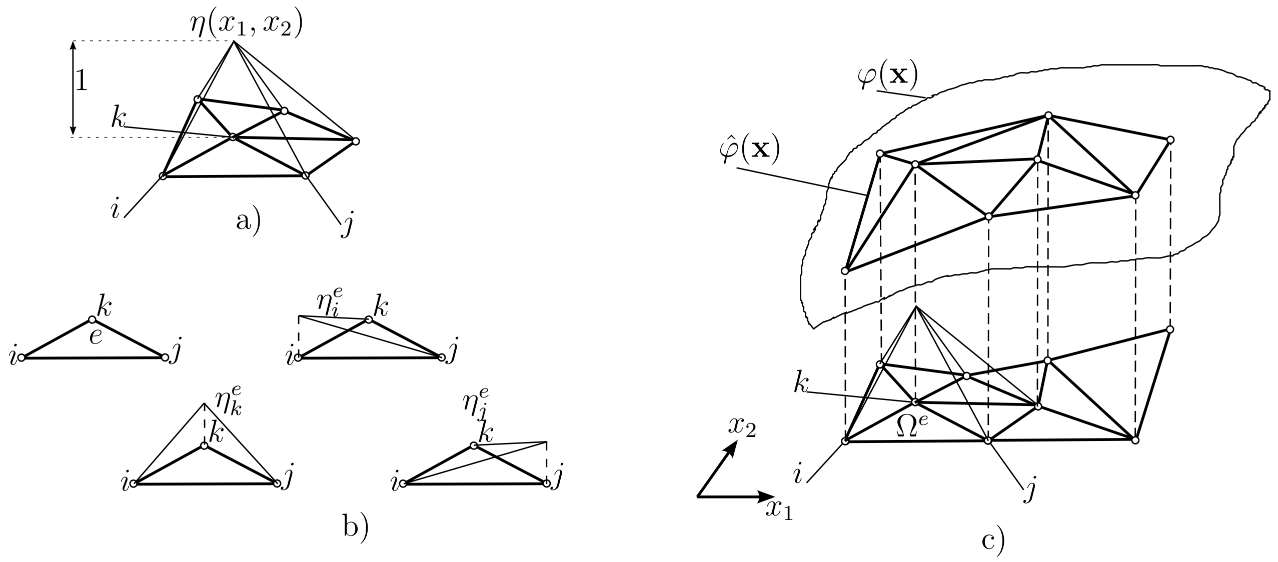


Figure 2.4: a) An example of a FEM basis function $\eta(x_1, x_2)$ in 2-D; b) parts of the $\eta(x_1, x_2)$ are distributed over a mesh element e; c) an approximation $\hat{\varphi}(x_1, x_2)$ of a function $\varphi(x_1, x_2)$ using a FEM basis.

The quality of the approximation can be controlled by the mesh element size or by the degree of the polynomial basis function. The combination of both is used in $hp-$FEM methods [27].

Given a set of $k+1$ data points $(x_0, y_0), ..., (x_j, y_j), ..., (x_k, y_k)$ where no two x_j are the same, the interpolation polynomial in the Lagrange form is a linear combination

$$L(x) = \sum_{j=0}^{k} y_j l_j(x) \tag{2.43}$$

of the Lagrange basis polynomials

$$l_j(x) = \prod_{\substack{0 \le m \le k \\ m \ne j}} \frac{x - x_m}{x_j - x_m}, \; 0 \le j \le k. \tag{2.44}$$

All basis polynomials are zero at $x = x_i$, except $l_i(x)$, for which it holds that $l_i(x_i) = 1$. Figure 2.5 (a) shows an example of non-uniform Lagrange interpolation.

A Lagrange polynomial interpolating cubic B-spline at integer-valued nodes is shown in Figure 2.5 (b).

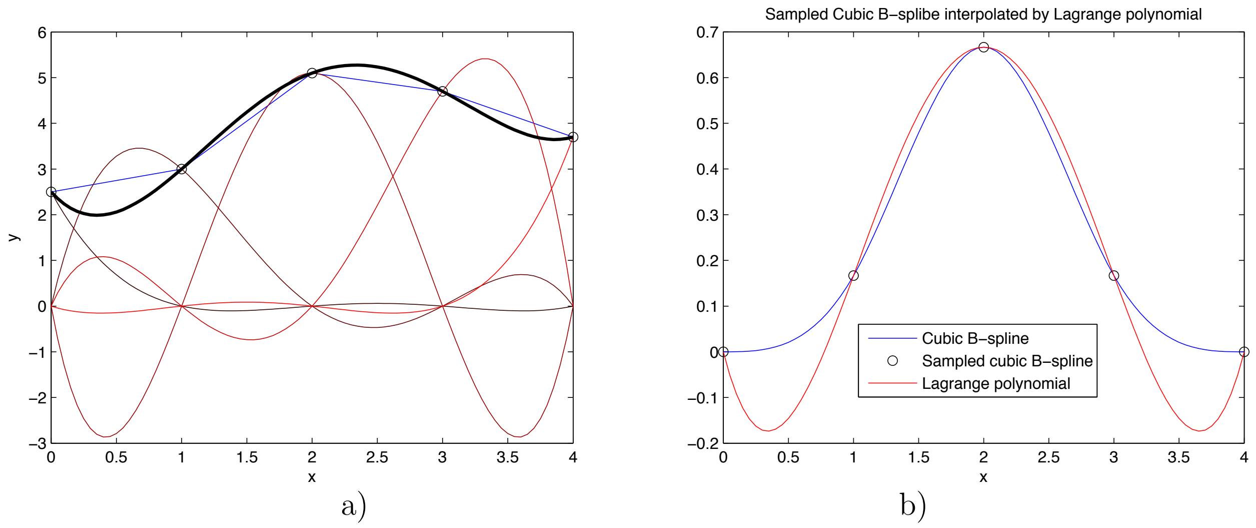


Figure 2.5: (a) Non-uniform Lagrange interpolation, (b) Lagrange polynomial interpolating cubic B-spline at integer-valued nodes.

Despite the flexibility of FEM, solving a problem PDE by FEM may be computationally very demanding on large-scale grids if high accuracy is required because:

1. FEM, especially when applied to 3-D domains of arbitrary shape, typically relies on an unstructured conforming 3-D mesh [28, 29, 30, 31, 25]. The widely-used method to construct a mesh is a Delaunay triangulation [32]. More advanced meshing methods provide better-quality meshes [33, 34]. Generally, meshing is a time-consuming and challenging procedure. Meshing automation defines a critical practical problem in FEM-based approaches, which heavily depends on the specific properties of the domain considered [35]. In FEM, distorted or low-quality meshes can lead to higher errors, but re-meshing is not guaranteed to be feasible in finite time for complex three-dimensional geometries [36].

2. The use of high order polynomial bases can excessively increase the size of the underlying sparse linear system and its density. High-order Lagrange polynomials are tend to oscillate (2.5 (b)).

3. For some tasks like soft-field tomography, separation of bases used to represent the forward solution and parameters [28] increases algorithmic complexity.

4. Approximation of boundaries with linear elements is not accurate in most cases, for precise boundary approximation curved isoparametric elements are typically required. Good surface approximation may be difficult to achieve with a low number of FEM elements [37].

Discontinuous Galerkin FEM

Discontinuous Galerkin (DG) FEM methods have been successfully applied to practical problems involving elliptic PDEs [38, 39]. Such methods combine physical accuracy and flexibility of mesh generation via weakly enforced continuity of discontinuous elements, however at the price of relatively high computational cost [38].

2.3.3 Finite Volume Method

The finite volume method (FVM) [40] is based on a volume integral formulation of the PDE with a finite partitioning set of volumes used to discretize the domain. The FVM applies conservation principles at each finite element volume ensuring global energy conservation in the whole domain. For example, the FVM is commonly used for solving problems of fluid dynamics.

Consider a steady transport equation for some quantity $\varphi(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^d$

$$\nabla \cdot (\mathbf{v}\varphi(\mathbf{x})) = \nabla \cdot (D(\mathbf{x})\nabla\varphi(\mathbf{x})) + g(\mathbf{x}), \ \mathbf{x} \in \Omega, \tag{2.45}$$

with defined boundary conditions on $\partial\Omega$. Integration over the volume Ω produces

$$\int_\Omega \nabla \cdot (\mathbf{v}\varphi(\mathbf{x}) - D(\mathbf{x})\nabla\varphi(\mathbf{x}))\, d\mathbf{x} = \int_\Omega g(\mathbf{x})d\mathbf{x}. \tag{2.46}$$

By applying the divergence theorem, the left-hand side volume integral can be transformed into an integral over the boundary $\partial\Omega$:

$$\int_{\partial\Omega} (\mathbf{v}\varphi(\mathbf{x}) - D(\mathbf{x})\nabla\varphi(\mathbf{x})) \cdot \mathbf{n}dS = \int_\Omega g(\mathbf{x})d\mathbf{x}. \tag{2.47}$$

The domain Ω is approximated by a set of finite number of non-overlapping volume elements V_i such as $\widetilde{\Omega} = \cup_{i=1}^N V_i$, $V_i \cap V_j = \emptyset$, $\forall i \neq j$. An example of the domain discretization is shown in Fig. 2.6. Therefore, the FVM method solves the following system of equations

$$\int_{\partial V_i} (\mathbf{v}\varphi(\mathbf{x}) - D(\mathbf{x})\nabla\varphi(\mathbf{x})) \cdot \mathbf{n}dS = \int_{V_i} g(\mathbf{x})d\mathbf{x}, \ V_i \subset \Omega, \ i = 1, 2, \ldots, N \tag{2.48}$$

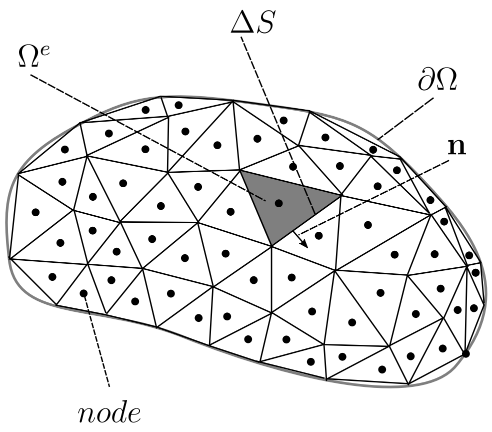


Figure 2.6: An example of a two-dimensional finite volume mesh for a domain Ω.

Like in FEM, the problem (2.48) boils down to solving a system of linear algebraic equations, where the unknown values φ_i represent some approximation of $\varphi(\mathbf{x})$ in a volume element ∂V_i [41]. The FVM requires domain meshing and poses difficulties in achieving high precision of the solution.

2.3.4 Boundary Element Method

In Boundary Element Method (BEM) [42] the boundary $\partial\Omega$ is divided into K elements (Fig. 2.7). The method solves for K singularity solutions which, when superimposed, satisfies the required conditions at the midpoint of each element. The system of K equations to be solved is much smaller than the system needed to solve the same problem using FEM, although the system matrix is no longer sparse.

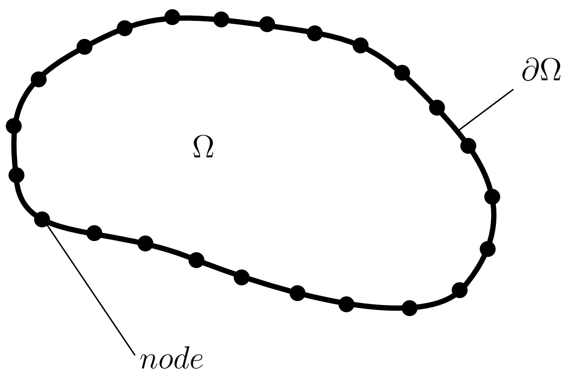


Figure 2.7: An example of a BEM discretization of a two-dimensional domain Ω.

After K solutions are obtained, the solution at any point in Ω can be constructed by Green's functions. The method is very accurate due to semi-analytical nature. It uses fewer dimensions than FEM, therefore, the meshing is more efficient. It is well suited for modeling of thin shell-like structures of materials. However, BEM matrices are dense and non-symmetrical that for large problems poses high computational and memory requirements.

2.3.5 Meshless Methods

The meshless methods [43] do not rely on a finite element or volume mesh as FEM or FVM. The automatic generation of a good quality mesh, especially volumetric, is difficult and consumes significant time. In meshless methods, the approximation is built based on the nodes only. The basis functions are placed at the nodes and the numerical integration is performed without a mesh. An example of a domain discretization is shown in Fig. 2.8. A family of meshless methods includes element-free Galerkin method (EFG), reproducing kernel particle method (RKPM), the partition of unity finite element method (PUFEM), etc.

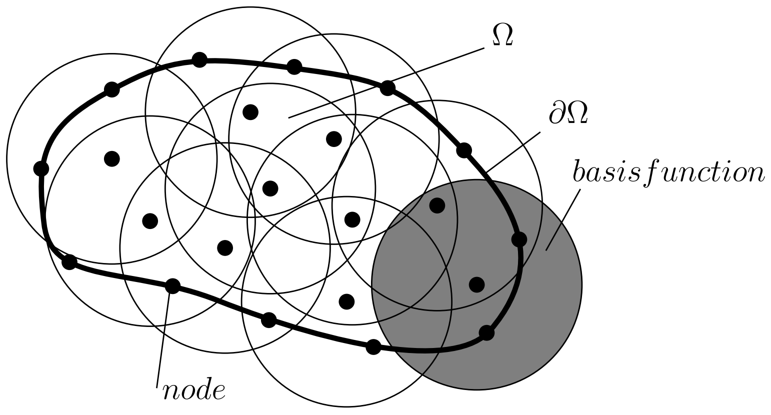


Figure 2.8: Domain discretization in meshless methods.

In most mesh-free methods, complicated non-polynomial interpolation functions are used that make the integration difficult. Another problem is with Dirichlet BC, as the basis functions do not satisfy the requirements on the domain boundary $\partial\Omega$. Meshless methods are computationally less efficient [36] that finite element/volume methods mainly because of complicated basis functions and inefficient integration [44].

2.3.6 Weighted Extended B-spline Method

The use of splines for solving PDEs has been studied by Höllig [20, 45, 46], who showed how tensor-product B-splines can be used in the context of finite-element methods. Weighted Extended B-splines (WEB) [46] complement a regular B-spline discretization with a basis extension to satisfy the Dirichlet boundary condition and improve numerical stability. The method does not require meshing, the basis is constructed on a regular grid.

Consider WEB method in two dimensions. A domain Ω with boundary $\partial\Omega$ is approximated with rectangular grid with grid step $h \in \mathbb{R}$. The grid cells $Q = \mathbf{l}h + [0,1]^2 h,\ \mathbf{l} \in \mathbb{Z}^2$ are partitioned into interior, boundary, and exterior cells, depending on whether $Q \subseteq \Omega$, or $Q \cap \partial\Omega \neq \emptyset$, or $Q \cap \Omega = \emptyset$. Fig. 2.9 (a, b) shows a two-dimensional domain with round boundary, where interior, boundary and exterior cells are depicted by blue, red and gray colors correspondingly.

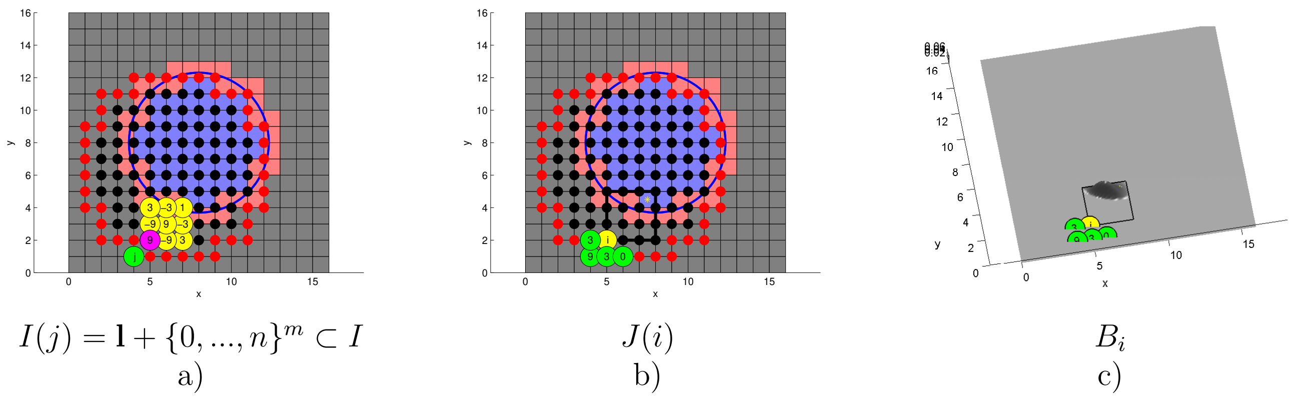


Figure 2.9: Construction of a two-dimensional WEB basis.

A bivariate tensor product B-spline $b^n_{\mathbf{k},h}(\mathbf{x})$ of degree n with grid step h is placed at each grid node, where multidimensional index $\mathbf{k} \in K \subset \mathbb{Z}^2$ is assigned to lower left corner of B-spline. All B-splines $b_{\mathbf{k}}(\mathbf{x})$ are classified into inner B-splines $b_{\mathbf{i}}(\mathbf{x})$, $\mathbf{i} \in I \subset K$, which have at least one interior cell $Q_{\mathbf{i}}$ in their support, and outer B-splines $b_{\mathbf{j}}(\mathbf{x})$, $\mathbf{j} \in J = K \setminus I$ for which support of $b_{\mathbf{j}}(\mathbf{x})$ consists entirely of boundary and exterior cells. Figure 2.9 (a, b) shows inner and outer B-splines distinguished with black and red circles placed at the lower left corner of their supports. For an outer index $\mathbf{j} \in J$ let $I(\mathbf{j}) = \mathbf{l} + \{0, ..., n\}^m \subset I$ be a set of inner indices closest to $\mathbf{j}$, assuming that h is small enough so that such an array exists. We denote by

$$e_{\mathbf{i},\mathbf{j}} = \prod_{v=1}^{m} \prod_{\substack{\mu=0 \\ l_v+\mu \neq i_v}}^{n} \frac{j_v - l_v - \mu}{i_v - l_v - \mu} \tag{2.49}$$

the values of the Lagrange polynomials associated with $I(\mathbf{j})$ and by $J(\mathbf{i})$ the set of all $\mathbf{j}$ with $\mathbf{i} \in I(\mathbf{j})$. Then, the WEB-splines

$$B_{\mathbf{i}} = \frac{w}{w(\mathbf{x_i})} \left[b_{\mathbf{i}}(\mathbf{x}) + \sum_{\mathbf{j} \in J(\mathbf{i})} e_{\mathbf{i},\mathbf{j}} b_{\mathbf{j}}(\mathbf{x}) \right], \ \mathbf{i} \in I, \tag{2.50}$$

form a basis for the WEB-space $w^e\mathbb{B}^n_h(\Omega)$. An example of a WEB-spline placed at the domain boundary is shown in Fig. 2.9 (c).

The weak formulation of a given PDE is discretized using the WEB-basis, and the resulting system of algebraic equations is solved using a sparse matrix approach [46].

2.3.7 Isogeometric Analysis Method

Historically, geometric representations used in Finite Element Analysis (FEA) software and in Computer Aided Design (CAD) software are different. In engineering practice, a design may be developed in a CAD system, then its different geometric description (mesh) is generated to enable FEA. In most cases, a mesh is only an approximation of the CAD model, and often its generation is semi-automated and may occupy up to 80% of overall analysis time [37].

The main idea of isogeometric analysis [37, 47, 48, 49] is to use the same basis for analysis and geometric representation of the model. Standard CAD designs are based on NURBS (Non-Uniform Rational B-splines), and isogeometric analysis employs the same basis functions for FEA. NURBS surfaces are transformed into NURBS solids. CAD design geometry is represented exactly by a coarse mesh of "3-D NURBS elements". However, NURBS-based geometries often suffer from gaps, usually achieving only C^0 continuity across patch boundaries, the geometry refinement leads to an excessive overhead of NURBS control points [47]. NURBS are standard for CAD systems, as they can reproduce conic sections, however, B-splines [48], T-splines [47], etc., can be employed.

2.4 B-splines

The term B-spline is short for "basis spline". A spline is a piecewise polynomial function with segments smoothly connected with each other at nodes (points where the spline segments join). The smoothness of a spline is defined by its degree $n \in \mathbb{N}_0$, which determines the number of continuous derivatives of the spline. A spline can be represented as a weighted sum of shifted B-spline functions. Since their introduction in the late '60s [50, 51], B-splines have found many applications in computer graphics [52], computer-aided design [37], medical imaging [7, 10, 11] and particularly in numerical solutions of PDEs [20, 45, 46]. This work considers splines with uniformly distributed nodes.

2.4.1 B-spline Expansion

A univariate spline function $s(x),\ x \in \mathbb{R}$ can be uniquely represented by an expansion [50]

$$s(x) = \sum_{k\in\mathbb{Z}} c_k \beta^n(x/h - k),\ \ h \in \mathbb{R}, \tag{2.51}$$

that involves basis splines (B-splines) functions $\beta^n(x/h - k)$ of degree n expanded by a grid step h and shifted by an integer k, and the coefficients $c_k \in \mathbb{R}$. The spline (2.51) is uniquely described by the B-spline coefficients c_k.

B-spline functions are obtained from $(n+1)$-fold convolution of a rectangular pulse $\beta^0_+(x)$

$$\beta^n_+(x) = \underbrace{\beta^0_+(x) * \beta^0_+(x) * \ldots * \beta^0_+(x)}_{(n+1)\ \text{times}},\ \ \beta^0_+(x) = \begin{cases} 1, & x \in [0,1) \\ 0, & \textit{otherwise} \end{cases}, \tag{2.52}$$

resulting in a family of functions (shown in Fig. 2.10 (a)) with remarkable properties. Further in the work we consider only centered B-splines $\beta^n(x) = \beta^n_+(x + \frac{n+1}{2})$.

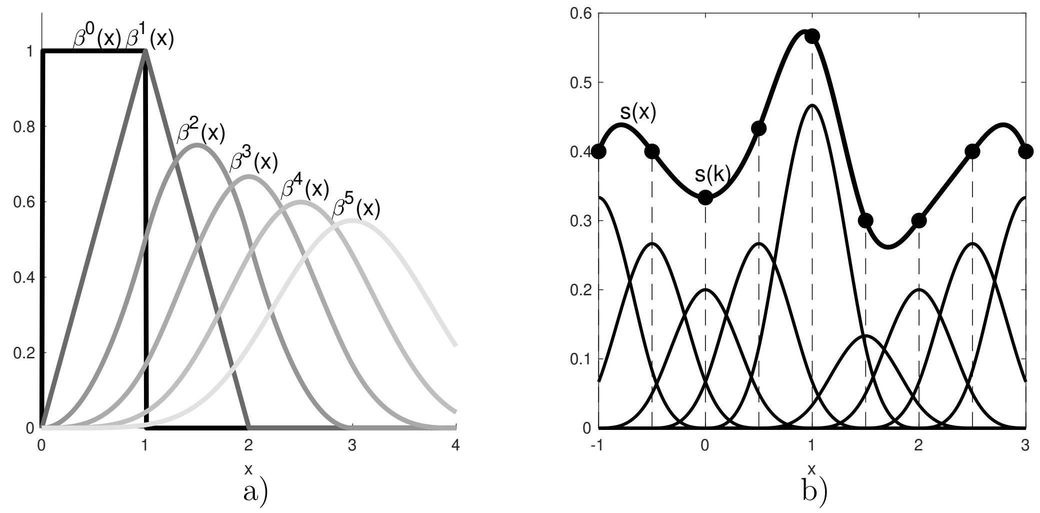


Figure 2.10: (a) A family of univariate B-spline functions $\beta_+^n(x)$ of degrees $n = 0...5$ obtained from $(n+1)$-fold convolution of the rectangular pulse $\beta_+^0(x)$, (b) An example of interpolation of samples $s(k)$ using univariate cubic B-spline functions $\beta^3(x)$, $s(x) = \sum_k c_k \beta^3(x-k)$.

2.4.2 Properties of B-splines

B-spline functions $\beta^n(x)$ are symmetrical and positive in their local support $(-\frac{n+1}{2}, \frac{n+1}{2})$. They are $(n-1)$-times continuously differentiable. The B-spline convolution property (2.52) induces the following important relations:

$$\beta^{m+n+1}(x) = \int_{-\infty}^{+\infty} \beta^m(x-y)\beta^n(y)dy,\ y \in \mathbb{R},\ m \in \mathbb{N}_0, \tag{2.53}$$

$$\beta^{m+n+1}(k-l) = \int_{-\infty}^{+\infty} \beta^m(x-k)\beta^n(x-l)dx,\ l \in \mathbb{Z}. \tag{2.54}$$

The first derivative of B-spline $\beta^n(x)$ can be determined as

$$\frac{\partial \beta^n(x)}{\partial x} = \beta^{n-1}\left(x + \frac{1}{2}\right) - \beta^{n-1}\left(x - \frac{1}{2}\right). \tag{2.55}$$

Similarly, the second derivative is represented as

$$\frac{\partial^2 \beta^n(x)}{\partial x^2} = \beta^{n-2}(x+1) - 2\beta^{n-2}(x) + \beta^{n-2}(x-1). \tag{2.56}$$

Analytical integration of a B-spline can be done as follows

$$\int_{-\infty}^{x} \beta^n(x)dx = \sum_{k=0}^{+\infty} \beta^{n+1}\left(x - \frac{1}{2} - k\right). \tag{2.57}$$

2.4.3 B-spline Interpolation

Often a spline function has to be constructed from a discrete set of data samples $s_k \in \mathbb{R}$, $k \in \mathbb{Z}$ in a way that the spline coincides with input data samples. This scenario is called an interpolation problem and consists in finding the coefficients c_k of expansion (2.51):

$$\sum_{l \in \mathbb{Z}} c_l \beta^n(x-l)|_{x=k} = s_k. \tag{2.58}$$

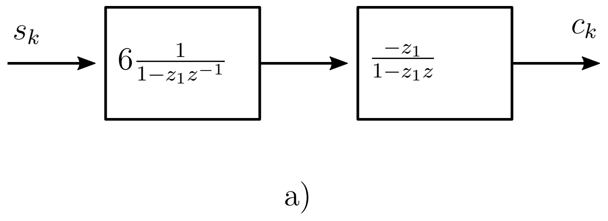
For splines of degree $n = 0$ and $n = 1$ the coefficients are equal to data samples $c_k = s_k$. For degrees $n \geq 2$ the one should solve the system of equations with some boundary conditions. Rather than employ a classical matrix-based approach (that is still commonly used by some authors) this work considers the more efficient digital filtering approach, extensively studied by Unser [7]. The approach of digital filtering is more efficient than matrix-based algorithms [11]. In this approach, the solution is found by inverse filtering

$$c_k = (b_1^n(k))^{-1} * s_k, \tag{2.59}$$

where $b_1^n(k) = \beta^n(x)|_{x=k}$ is a discrete B-spline. The filter kernel $(b_1^n(k))^{-1}$ is called the direct B-spline filter and can be implemented efficiently using a cascade of first-order casual and anti-casual recursive filters. In the case of cubic B-spline

$$(b_3^n(k))^{-1} \overset{z}{\longleftrightarrow} \frac{6}{z + 4 + z^{-1}} = 6\left(\frac{1}{1 - z_1 z^{-1}}\right)\left(\frac{-z_1}{1 - z_1 z}\right), \tag{2.60}$$

where $z_1 = -2 + \sqrt{3}$. The chain of filters is represented graphically in Fig. 2.11 (a). Note that boundary conditions should be specified to define spline function behaviour on the boundary. One could use mirror boundary conditions $s(-k) = s(k)$, $s(N) = s(N-2)$ shown in the Fig. 2.11 (b).



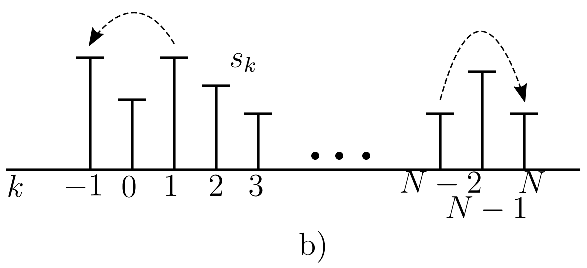


Figure 2.11: a) B-spline inverse transform, b) an example of signal mirror-W boundary condition.

The spline interpolation can be considered in a broader sense as a sampling problem of a band-limited function (an analogy with sinc-based interpolation):

$$s(x) = \sum_{k \in \mathbb{Z}} s_k \zeta^n(x - k), \tag{2.61}$$

where $\zeta^n(x)$ is a cardinal spline of degree n:

$$\zeta^n(x) = \sum_{k \in \mathbb{Z}} (b_1^n)^{-1}(k) \beta^n(x - k). \tag{2.62}$$

The cardinal spline $\zeta^n(x)$ converges to $sinc(x)$ as n goes to infinity. An example of a one-dimensional interpolation using cubic ($n = 3$) B-splines is shown in Fig. 2.10 (b).

2.4.4 B-spline Approximation

Sometimes a spline function has to approximate some continuous function $g(x)$, $x \in \mathbb{R}$. In this scenario the initial function $g(x)$ is approximated by a spline $s_a(x)$ in a way that minimizes $\mathbb{L}^2$-norm

$$\underset{s_a}{\operatorname{argmin}} \, ||g(x) - s_a(x)||_{\mathbb{L}^2}. \tag{2.63}$$

The solution to this problem is given by

$$c_k = \frac{1}{h}\langle g(x), \mathring{\psi}(x/h - k)\rangle, \; c_k \in l_2, \tag{2.64}$$

where $\langle \mathring{\psi}(x-k), \psi(x-l)\rangle = \delta(k-l)$, and $\psi(x) = \sum_{k\in\mathbb{Z}} p_k \beta^n(x-k)$. The coefficients can be efficiently computed via digital filtering using prefiltering and postfiltering filter stages [7]. Once the coefficients c_k are obtained, an approximation spline is constructed as

$$s_a(x) = \sum_{k\in\mathbb{Z}} c_k \psi(x/h - k). \tag{2.65}$$

2.4.5 Multivariate B-splines

A $d-$dimensional B-spline (multivariate B-spline) of degree n is defined as tensor (outer) product of d univariate B-splines

$$\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x}) = \beta^n(x_1/h_1 - k_1)\cdots\beta^n(x_d/h_d - k_d), \; \mathbf{k} \in \mathbb{Z}^d, \; \mathbf{h} \in \mathbb{R}^d \tag{2.66}$$

that results in separability that is an important property of the multivariate B-spline. An example of linear and cubic bivariate B-spline functions in presented in Fig. 2.12.

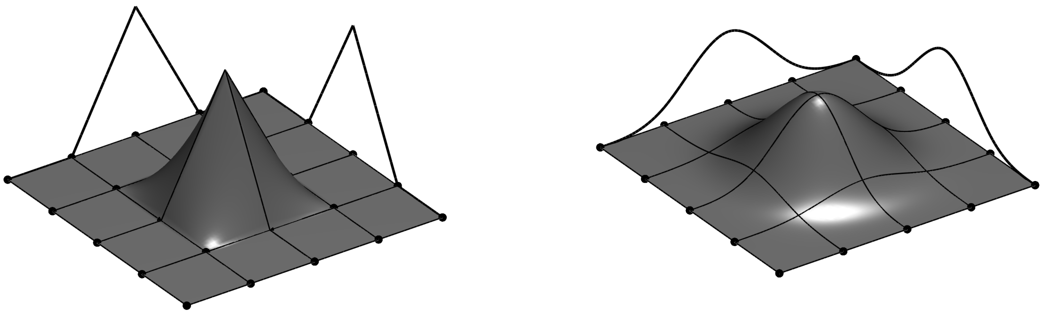


Figure 2.12: Bivariate linear ($n = 1$) and cubic ($n = 3$) B-spline functions $\beta^n(\mathbf{x})$.

The multidimensional signal processing benefits from the B-spline separability, for example in three-dimensional case for $s(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$ the spline representation

$$s(\mathbf{x}) = \sum_{k\in\mathbb{Z}}\sum_{l\in\mathbb{Z}}\sum_{m\in\mathbb{Z}} c_{klm}\beta^n(x_1-k)\beta^n(x_2-l)\beta^n(x_3-m) \tag{2.67}$$

is separable. The coefficients c_{klm} are found by a successive application of the direct B-spline transform (2.59) or $\mathbb{L}^2$-projection (2.64) along each signal dimension.

2.4.6 Applications

B-splines found a wide application for

1. multidimensional signal zooming, rotation, re-sizing, warping[10, 11, 7, 53],
2. sampling and interpolation, signal reconstruction [8],
3. contour and surface representation, active contour models [9],
4. image pyramids [53], multi-resolution,
5. numerical methods for PDEs [20, 54].

2.5 Tensors

Tensors are multidimensional objects that are used in numerous fields, including physics, psychometrics, signal processing, data mining, machine learning, etc. and became a powerful instrument for multidimensional data analysis. Tensors are replacing matrices more and more in many problems that were originally described in terms of matrices because of the limitations of flat-view matrix models [13]. One of the important tools in the tensor analysis is tensor decompositions [12] that play an important role in the data analysis, compression, noise removal, and feature extraction [55], data mining and machine learning [14].

2.5.1 Definitions

A tensor $\mathcal{A} \in \mathbb{R}^{K_1 \times K_2 \times \cdots \times K_N}$ of order N can be represented as an $N-$way array where elements $a_{k_1 k_2 \cdots k_N}$ are indexed by $k_n \in \{1, 2, ..., K_N\}$, $n = 1, 2, ..., N$. From a technical perspective, tensors are straight-forward generalizations of vectors and matrices, that allow natural representation of the functions of more than two indices $k_1, k_2, k_3 ...$, as depicted in Fig. 2.13. At the same time, scalar, vector and matrix can be treated as 0-way, 1-way and 2-way tensor respectively.

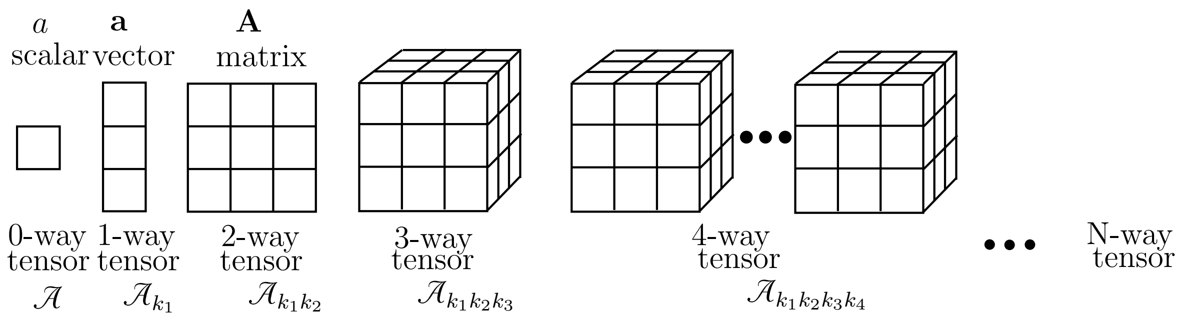


Figure 2.13: Tensors as generalizations of scalars, vectors and matrices.

2.5.2 Tensor Decompositions

Tensor decompositions play an essential role in the multidimensional data analysis. The Canonical Decomposition (CANDECOMP) and Tucker Decomposition are higher-order generalizations of the matrix singular value decomposition (SVD) and principal component analysis (PCA). For example, a third-order tensor $\mathcal{A} \in \mathbb{R}^{K \times L \times M}$ can be factorized using CANDECOMP as the sum of a finite number of rank-one tensors

$$\mathcal{A} = \sum_{r=1}^{R} \mathbf{x}_r \circ \mathbf{y}_r \circ \mathbf{z}_r, \tag{2.68}$$

where $\mathbf{x}_r \in \mathbb{R}^K$, $\mathbf{y}_r \in \mathbb{R}^L$, $\mathbf{z}_r \in \mathbb{R}^M$. The symbol "$\circ$" represents the vector outer product. Fig. 2.14 represents the decomposition graphically.

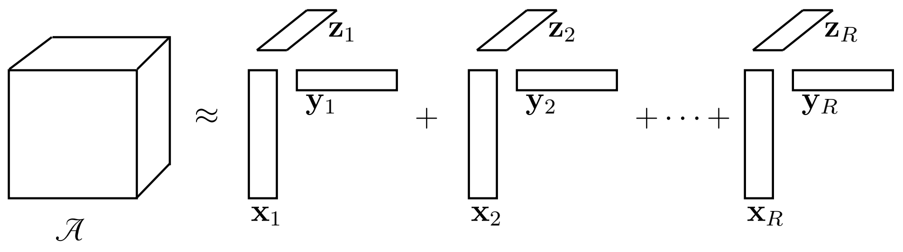


Figure 2.14: Canonical decomposition of a three-way tensor.

For example, a third-order tensor $\mathcal{A} \in \mathbb{R}^{K \times L \times M}$ can be factorized using Tucker decomposition as a core tensor multiplied by a matrix along each mode

$$\mathcal{A} = \mathcal{G} \times_1 \mathbf{X} \times_2 \mathbf{Y} \times_3 \mathbf{Z} = \sum_{p=1}^{P} \sum_{q=1}^{Q} \sum_{r=1}^{R} g_{pqr} \mathbf{x}_p \circ \mathbf{y}_q \circ \mathbf{z}_r, \tag{2.69}$$

where $\mathbf{X} \in \mathbb{R}^{I \times P}$, $\mathbf{Y} \in \mathbb{R}^{J \times Q}$, $\mathbf{Z} \in \mathbb{R}^{K \times R}$. The symbol "$\circ$" represents the vector outer product, $\times_n$ is n-mode product of a tensor with a matrix . Fig. 2.15 represents the decomposition graphically.

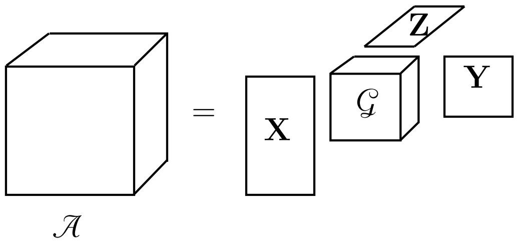


Figure 2.15: Tucker decomposition of a three-way tensor.

A more general form for the decomposition (2.69) called block term decomposition has the following structure

$$\mathcal{B} = \sum_{r=1}^{R} \mathcal{G}_r \times_1 \mathbf{X}_r \times_2 \mathbf{Y}_r \times_3 \mathbf{Z}_r, \tag{2.70}$$

that represents a sum of R Tucker decompositions.

2.5.3 Computational Tensor Algebra

A large number of different tensor products (matrix product, Kronecker product, Khatri-Rao product, etc.) and limitation of classical tensor index order have been addressed by the Computational Tensor Algebra [15]. This framework unifies the concepts from tensor analysis, multilinear algebra, and multidimensional signal processing. It facilitates tensor structure analysis and derivation of efficient computational algorithms.

Tensor Notation

The computational tensor algebra notation is based on the original tensor analysis notation with some modifications for more convenient manipulations with tensor indices.

We consider vector spaces $X \subseteq \mathbb{R}^K$, $Y \subseteq \mathbb{R}^L$, $Z \subseteq \mathbb{R}^M$, $K, L, M \in \mathbb{Z}$. Let a tensor product space $X \times Y \times Z \subseteq \mathbb{R}^{K \times L \times M}$ define a scalar field on a discrete finite three-dimensional grid. The corresponding tensor of this scalar field is denoted as $\mathcal{A}_{xyz}$, that is multidimensional array with indices being in direct relation with corresponding vector spaces. Tensors indices can be either covariant (e.g. $\mathcal{A}_{xyz}$), contravariant (e.g. $\mathcal{A}^{xyz}$), or mixed (e.g. $\mathcal{A}_z^{xy}$).

Computational tensor algebra introduces the commutativity of the tensor product. In analogy to an ordered physical space, computational tensor algebra requires the vector spaces of the tensor product space to have a unique predefined order. This constraint leads to the same result of the tensor products that are evaluated in an arbitrary order because the indices of the resulted tensor have unique positions determined by a predefined order of vector spaces. Computational tensor algebra introduced the generalized tensor product which combines different kinds of tensor products (Kronecker product, Khatri-Rao product, etc.).

2.5.4 Tensor Operations

In this work, a multi-dimensional discrete formulation of a PDE is analysed from a tensor viewpoint instead of the usual matrix-vector approach. Therefore, we depict the relevance and generalization of the tensor operations to the matrix ones.

Scaling

For vectors $\mathbf{u},\ \mathbf{v} \in \mathbb{R}^N$ we define corresponding tensors $\mathcal{U},\ \mathcal{V} \in \mathbb{R}^N$. The scalar $a \in \mathbb{R}$. Vector scaling is defined as $[v_1, v_2, .., v_n] = [au_1, au_2, ..., au_n]$ or $\mathbf{v} = a\mathbf{u}$. In tensor notation is defined as

$$\mathcal{V}^n = a\mathcal{U}^n \tag{2.71}$$

Dot Product and Frobenius Inner Product

The dot product for vectors is defined as $a = \mathbf{u}^T\mathbf{v}$ or $a = \sum_{n=1}^{N} u_n v_n$ and for tensors is defined as

$$a = \mathcal{U}_n\mathcal{V}^n \tag{2.72}$$

For matrices $\mathbf{X} \in \mathbb{R}^{M\times N}$ and $\mathbf{Y} \in \mathbb{R}^{M\times N}$ we define corresponding tensors $\mathcal{X} \in \mathbb{R}^{M\times N}$ and $\mathcal{Y} \in \mathbb{R}^{M\times N}$. The Frobenius inner product takes two matrices and results in a scalar $a = \sum_{mn} x_{mn}y_{mn} = \langle \mathbf{X}, \mathbf{Y}\rangle_F$. In the tensor notation is defined as

$$a = \mathcal{X}^{mn}\mathcal{Y}_{mn} \tag{2.73}$$

Bilinear Form

For matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ we define a corresponding tensor $\mathcal{A} \in \mathbb{R}^{N\times N}$. The bilinear form is defines as $a = \mathbf{u}^T\mathbf{A}\mathbf{v}$ and using tensor notation is defined as

$$a = \mathcal{U}_{n_1}\mathcal{A}^{n_1}_{n_2}\mathcal{V}^{n_2} \tag{2.74}$$

Outer Product

The vector outer product is defines as $\mathbf{A} = \mathbf{u}\mathbf{v}^T$ and for tensors

$$\mathcal{A}^{n_1 n_2} = \mathcal{U}^{n_1}\mathcal{V}^{n_2} \tag{2.75}$$

Matrix-Vector Product

For matrix $\mathbf{B} \in \mathbb{R}^{M\times N}$ and vector $\mathbf{g} \in \mathbb{R}^M$ we define tensors $\mathcal{B} \in \mathcal{R}^{M\times N}$ and $\mathcal{G} \in \mathbb{R}^M$. The matrix-vector product $\mathbf{g} = \mathbf{B}\mathbf{u}$ in a tensor-based notation is defined as

$$\mathcal{G}^m = \mathcal{B}^m_n\mathcal{U}^n \tag{2.76}$$

Matrix-Matrix Product

For matrices $\mathbf{D} \in \mathbb{R}^{M\times L}$ and $\mathbf{C} \in \mathbb{R}^{N\times L}$ we define tensors $\mathcal{D} \in \mathcal{R}^{M\times L}$ and $\mathcal{C} \in \mathbb{R}^{N\times L}$. The matrix-matrix product $d_{ml} = \sum_{n=1}^{N} b_{mn}c_{nl}; m = 1, ..., M; n = 1, ..., N$ or $\mathbf{D} = \mathbf{B}\mathbf{C}$ the tensor notation is

$$\mathcal{D}^m_l = \mathcal{B}^m_n\mathcal{C}^n_l \tag{2.77}$$

Elementwise Product

For a vector $\mathbf{w} \in \mathbb{R}^N$ we define corresponding tensor $\mathcal{W} \in \mathbb{R}^N$. The elementwise product $[w_1, w_2, ...w_n] = [u_1v_1, u_2v_2, ..., u_nv_n]$ or $\mathbf{w} = \mathbf{u} \circ \mathbf{v}$ has tensor notation

$$\mathcal{W}^m = \mathcal{U}^m \mathcal{V}^m \tag{2.78}$$

Tensor Product

For matrices $\mathbf{F} \in \mathbb{R}^{MK \times NL}$ and $\mathbf{E} \in \mathbb{R}^{K \times L}$ we define corresponding tensors $\mathcal{F} \in \mathbb{R}^{M \times N \times K \times L}$ and $\mathcal{E} \in \mathbb{R}^{K \times L}$. The tensor product $\mathbf{F} = \mathbf{B} \otimes \mathbf{E}$, where $\otimes$ is Kroneker product, in tensor notation is defined as

$$\mathcal{F}^{mn}_{kl} = \mathcal{B}^{mn} \mathcal{E}_{kl} \tag{2.79}$$

Generalized Tensor Product

For tensors $\mathcal{H} \in \mathbb{R}^{K \times I \times M \times L}$ and $\mathcal{T} \in \mathbb{R}^{N \times I \times K}$ the generalized tensor product is defined as

$$\mathcal{G}^{ni}_{ml} = \mathcal{H}^{ki}_{ml} \mathcal{T}^{ni}_{k} \tag{2.80}$$

resulting in a tensor $\mathcal{G} \in \mathbb{R}^{N \times I \times M \times L}$. There is no straightforward equivalent in the matrix notation for this operation.

Subtensors

A subset of indices can be fixed in order to get a subtensor. For example, for a tensor $\mathcal{A}_{klm}$ we fix index l to get a subtensor $\tilde{\mathcal{A}}_{km} = \mathcal{A}_{klm} | l \in \mathbb{L}$.

2.5.5 Tensor Structure Analysis

A formulation of a multidimensional problem that is done in terms of computational tensor algebra preserves the original data structure in comparison to the matrix-based formulation. This information helps to derive efficient solving algorithms [16].

The computational tensor algebra framework has been successfully applied to the problem of reconstruction of large sets of irregularly sampled multidimensional data [8]. In this case, the sparse decomposition of the problem was revealed that allowed to implement a highly parallel memory-efficient spline reconstruction algorithm.

In this work, the computational tensor algebra approach is applied to a discreet variational formulation of PDEs, reveals its tensor structure and results in efficient computational algorithms, extending the results published in application to Optical Diffusion Tomography forward problem [17, 18].

Chapter 3

Tensor B-Spline Numerical Method for PDEs

3.1 Main Contributions

This chapter presents a new Tensor B-spline method for numerical solutions of PDEs. The method relies on the use of B-spline functions and algorithms, takes advantage of a tensor structure of a PDE discrete formulation and high-performance matrix-free computations. The B-spline signal processing framework is used pervasively to represent all terms of a PDE. The high-order B-spline representations of PDE coefficients and source are obtained efficiently by means of separable digital filtering techniques. The method's efficient integration procedures result in a multilinear system of equations with a sparse tensor decomposition. This tensor structure yields several specific computational strategies. No meshing is required, domains with arbitrarily-shaped boundaries are supported and efficient integration on the boundary is performed using the Divergence Theorem and properties of B-splines.

3.1.1 The Method Advantages

Common stages of any numerical method for PDE after an error minimization criteria was chosen are: 1) approximation of the PDE unknown, coefficients and source functions; 2) an integration process in order to construct the system of algebraic equation; 3) numerical solving of the resulting algebraic equations. At each stage, the Tensor B-spline method provides some distinctions in comparison with the state-of-the-art.

B-spline Representations via Digital Filtering

The B-spline-based representations of a PDE coefficients an source are obtained efficiently via separable digital filtering algorithms. Although B-splines are used in the WEB method [46], the WEB method does not represent PDE coefficients with B-splines, and spline representations are treated rather in a traditional way using matrices. We propose to use a B-spline signal processing framework [7] for PDEs that allows many useful techniques in a computationally-efficient fashion.

Efficient High-degree Basis

The majority of FEM schemes often use low-degree polynomial elements [28, 30]. The reason for this is the implementation difficulty and high memory consumption of high-degree FEM. B-splines provide a less expensive high-degree basis allowing accurate approximation and fast convergence.

Mesh-free Discretization

The finite element/volume methods inherently rely on a domain meshing. The mesh conforms the domain in the majority of cases. Mesh quality highly affects the accuracy and convergence of an FEM solver [56], but high-quality meshes are challenging to generate [34, 56, 57] and may consume a significant part of the solving time [37]. Moreover, the mesh requires additional storage space. Meshless methods eliminate meshing; however, they have inefficient integration procedures [44]. The B-spline basis is easily constructed on tensor-product grids allowing mesh-free discretizations.

Efficient Numerical Integration

Meshless methods or methods that use non-conforming domain discretization suffer from inefficient numerical integration [44]. The use of B-spline properties and the Divergence Theorem provide efficient and straightforward integration techniques.

Tensor-based Optimizations

The analysis of the structure of the discrete formulation allows free optimization of the system operator computations. The use of computational tensor algebra induces several variants of the tensors "slicing" and ways of contraction along dimensions. This is rather impossible with matrix algebra as it flattens the multidimensional structure and diminishes the space for optimizations.

Efficient Matrix-free Computations

The state-of-the-art numerical methods like FEM [26, 28, 29, 30, 31, 25], WEB [46, 20, 45], meshless methods [36], DG-FEM [38], and others, rely on a sparse matrix framework and sparse matrix-vector multiplication (SpMV) for computations. This classical approach of matricizing inherently multidimensional discrete formulation leaves no room for optimization, requires special storage format and consumes additional storage space. Tensors preserve the multidimensional structure of the problem, and, together with B-splines, result in matrix-free kernel-based efficient computational techniques.

Parallelism and Scalability

Parallelization of the computations in the mesh-based domains might be difficult. Large-scale problem solutions require a large amount of memory and therefore, expensive hardware. B-splines are attached to a regular grid and allow intuitive parallelization techniques that are well-scalable across a different number of CPU cores.

3.2 B-spline Discretization of a PDE

3.2.1 Approximation of the Solution

In most cases, it is either impossible or intractable to obtain an analytic solution of a PDE in a domain Ω with specified boundary conditions on an arbitrarily-shaped domain boundary $\partial\Omega$. Therefore the solution is obtained numerically. For that, the unknown function $\varphi(\mathbf{x}),\ \mathbf{x} \in \mathbb{R}^d$ is approximated by an expansion

$$\varphi(\mathbf{x}) \approx \hat{\varphi}(\mathbf{x}) = \sum_{k \in \mathbb{Z}} c_k \eta_k(\mathbf{x}),\ \ \mathbf{x} \in \Omega. \tag{3.1}$$

From (3.1) it follows that for given basis functions $\eta_k(\mathbf{x})$, the function $\hat{\varphi}(\mathbf{x})$ is fully described by the expansion coefficients c_k. It is highly important to choose appropriate basis functions $\eta_k(\mathbf{x})$, where "appropriate" typically refers to good approximation properties and linear independence.

3.2.2 Variational Formulation of a PDE

The error between the function $\varphi(\mathbf{x})$ and its approximation $\hat{\varphi}(\mathbf{x})$ defines the residual, which is minimized by specific numerical procedures. For instance, the method of weighted residuals [25] requires

$$\int_\Omega \psi_l(\mathbf{x})(\varphi(\mathbf{x}) - \hat{\varphi}(\mathbf{x}))d\mathbf{x} = 0, \tag{3.2}$$

where $\psi_l(\mathbf{x})$, $l \in \mathbb{Z}$ are some weight functions. Substitution of (3.1) into (3.2) results in

$$\sum_{k\in\mathbb{Z}} c_k \int_\Omega \psi_l \eta_k(\mathbf{x})d\mathbf{x} = \int_\Omega \psi_l \varphi(\mathbf{x})d\mathbf{x},\ \ l \in \mathbb{Z}. \tag{3.3}$$

We consider an example of the Diffusion PDE coupled with Robin boundary conditions

$$\begin{aligned} -\nabla \cdot (D(\mathbf{x})\nabla\varphi(\mathbf{x})) + \mu_a(\mathbf{x})\varphi(\mathbf{x}) = q(\mathbf{x}),\ \ \mathbf{x} \in\ \Omega,\\ 2D(\mathbf{x})(\nabla\varphi(\mathbf{x}) \cdot \mathbf{n}) + \varphi(\mathbf{x}) = 0,\ \ \mathbf{x} \in \partial\Omega, \end{aligned} \tag{3.4}$$

where $\varphi(\mathbf{x})$ is the density of the diffusing material, $D(\mathbf{x})$ is the diffusion coefficient, $\mu_a(\mathbf{x})$ is the absorption coefficient and $q(\mathbf{x})$ is the source density. We multiply (3.4) by weight functions $\psi_l(\mathbf{x})$, $l \in \mathbb{Z}$ and integrate over the domain Ω:

$$\int_\Omega (-\nabla \cdot (D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x})))\psi_l(\mathbf{x})d\mathbf{x} + \int_\Omega \mu_a(\mathbf{x})\hat{\varphi}(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x} - \int_\Omega q(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x} = 0. \tag{3.5}$$

Expression under the integral in the first term in Eq. (3.5) can be represented as follows:

$$[\nabla \cdot (D(\mathbf{x})\nabla\varphi(\mathbf{x}))]\psi(\mathbf{x}) = \nabla \cdot (\psi(\mathbf{x})D(\mathbf{x})\nabla\varphi(\mathbf{x})) - D\nabla\varphi(\mathbf{x}) \cdot \nabla\psi(\mathbf{x}), \tag{3.6}$$

that results in:

$$\begin{aligned} \int_\Omega (-\nabla \cdot (D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x})))\psi_l(\mathbf{x})d\mathbf{x} =\\ \int_\Omega D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}) \cdot \nabla\psi_l(\mathbf{x})d\mathbf{x} - \int_\Omega \nabla \cdot (\psi_l(\mathbf{x})D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}))d\mathbf{x}. \end{aligned} \tag{3.7}$$

The second term on the right side of (3.7) can be expressed via surface integral using the Divergence Theorem

$$\int_\Omega \nabla \cdot (\psi_l(\mathbf{x})D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}))d\mathbf{x} = \int_{\partial\Omega} \psi_l(\mathbf{x})D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}) \cdot \mathbf{n}ds. \tag{3.8}$$

Arranging all together we get:

$$\begin{aligned} \int_\Omega D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}) \cdot \nabla\psi_l(\mathbf{x})d\mathbf{x} - \int_{\partial\Omega} \psi_l(\mathbf{x})D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x}) \cdot \mathbf{n}ds +\\ \int_\Omega \psi_l(\mathbf{x})\mu_a(\mathbf{x})\hat{\varphi}(\mathbf{x})d\mathbf{x} - \int_\Omega \psi_l(\mathbf{x})q(\mathbf{x})d\mathbf{x} = 0. \end{aligned} \tag{3.9}$$

We substitute the Robin boundary condition

$$\nabla\hat{\varphi}(\mathbf{x})\cdot\mathbf{n} = -\frac{1}{2D(\mathbf{x})}\hat{\varphi}(\mathbf{x}) \tag{3.10}$$

into (3.9) and obtain

$$\begin{gathered}\int_\Omega D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x})\cdot\nabla\psi_l(\mathbf{x})d\mathbf{x}+ \\ \int_\Omega \mu_a(\mathbf{x})\hat{\varphi}(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x} + \frac{1}{2}\int_{\partial\Omega}\hat{\varphi}(\mathbf{x})\psi_l(\mathbf{x})ds = \int_\Omega q(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x}.\end{gathered} \tag{3.11}$$

Many formulations have a similar structure as (3.11), therefore it is useful to consider a more general variational formulation of the form

$$a(\varphi(\mathbf{x}),\psi(\mathbf{x})) = l(\psi(\mathbf{x})), \tag{3.12}$$

where $a(\cdot,\cdot)$ is an elliptic bilinear form and $l(\cdot)$ is a bounded linear functional on a Hilbert space $\mathbb{H}$. Basis (trial) functions $\varphi(\mathbf{x})$ and weight (test) functions $\psi(\mathbf{x})$ belong to a Sobolev space [58]: $\varphi(\mathbf{x}) \in \mathbb{W}^{1,2}(\Omega)$, $\psi(\mathbf{x}) \in \mathbb{W}^{1,2}(\Omega)$, $\mathbb{W}^{1,2}(\Omega) = \{f(x) : ||f(x)||_{\mathbb{L}^2(\Omega)} < \infty,\ ||f(x)||_{\mathbb{W}^{1,2}(\Omega)} < \infty\}$.

3.2.3 Ritz-Galerkin Formulation with B-splines

The approximation of the unknown function $\hat{\varphi}(\mathbf{x})$ is expanded over multivariate B-spline basis functions $\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x}) = \beta^n(x_1/h_1 - k_1)\cdots\beta^n(x_d/h_d - k_d)$, $\mathbf{h} \in \mathbb{R}^d$, attached to the nodes of a rectangular grid with indices $\mathbf{k} \in \mathbb{Z}^d$ as

$$\begin{aligned}\hat{\varphi}(\mathbf{x}) &= \sum_{k_1}\cdots\sum_{k_d} c_{k_1\cdots k_d}\beta^{n_b}(x_1/h_1 - k_1)\cdots\beta^{n_b}(x_d/h_d - k_d) \\ &= \sum_{\mathbf{k}\in\mathbb{Z}^d} c_\mathbf{k}\beta^{n_b}_{\mathbf{k},\mathbf{h}}(\mathbf{x}).\end{aligned} \tag{3.13}$$

In the same way we represent the coefficients of the PDE

$$\begin{aligned}D(\mathbf{x}) &\approx \sum_{\mathbf{m}\in\mathbb{Z}^d}\mathcal{D}_\mathbf{m}\beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}), \\ \mu_a(\mathbf{x}) &\approx \sum_{\mathbf{m}\in\mathbb{Z}^d}\mathcal{M}_\mathbf{m}\beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}),\end{aligned} \tag{3.14}$$

and the light source

$$q(\mathbf{x}) \approx \sum_{\mathbf{j}\in\mathbb{Z}^d}\mathcal{Q}_\mathbf{j}\beta^{n_s}_{\mathbf{j},\mathbf{h}}(\mathbf{x}). \tag{3.15}$$

The B-spline degrees used for representation of the unknown, coefficients and the source can be different, we denote 1) n_b is the B-spline degree of PDE unknown, 2) n_p is the B-spline degree of PDE coefficients, 3) n_s is the B-spline degree of a PDE source.

Functions $D(\mathbf{x})$, $\mu_a(\mathbf{x})$, and $q(\mathbf{x})$ in the expansions above are obtained by interpolation or approximation (e.g. $\mathbb{L}^2$-projection) [7] of known distributions. The B-spline coefficients $\mathcal{D}_\mathbf{m}$, $\mathcal{M}_\mathbf{m}$, and $\mathcal{Q}_\mathbf{j}$ are computed via efficient digital-filtering algorithms as will be described further in this chapter.

By choosing test functions $\psi_l(\mathbf{x}) = \beta^{n_b}_{\mathbf{l},\mathbf{h}}(\mathbf{x})$, we obtain the following Ritz-Galerkin formulation [25] (equivalent to the least squares method) of (3.11):

$$\begin{aligned}\sum_{\mathbf{k}} \mathcal{C}_{\mathbf{k}} \sum_{\mathbf{m}} \mathcal{D}_{\mathbf{m}} \int_{\Omega} \left(\nabla \beta^{n_b}_{\mathbf{k},\mathbf{h}}(\mathbf{x}) \cdot \nabla \beta^{n_b}_{\mathbf{l},\mathbf{h}}(\mathbf{x})\right) \beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}) d\mathbf{x} + \\ \sum_{\mathbf{k}} \mathcal{C}_{\mathbf{k}} \sum_{\mathbf{m}} \mathcal{M}_{\mathbf{m}} \int_{\Omega} \beta^{n_b}_{\mathbf{k},\mathbf{h}}(\mathbf{x}) \beta^{n_b}_{\mathbf{l},\mathbf{h}}(\mathbf{x}) \beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}) d\mathbf{x} + \\ \frac{1}{2\gamma} \sum_{\mathbf{k}} \mathcal{C}_{\mathbf{k}} \int_{\partial\Omega} \beta^{n_b}_{\mathbf{k},\mathbf{h}}(\mathbf{x}) \beta^{n_b}_{\mathbf{l},\mathbf{h}}(\mathbf{x}) ds = \\ \sum_{\mathbf{j}} \mathcal{Q}_{\mathbf{j}} \int_{\Omega} \beta^{n_s}_{\mathbf{j},\mathbf{h}}(\mathbf{x}) \beta^{n_b}_{\mathbf{l},\mathbf{h}}(\mathbf{x}) d\mathbf{x}, \ \ \mathbf{k}, \mathbf{m}, \mathbf{l}, \mathbf{j} \in \mathbb{Z}^d.\end{aligned} \tag{3.16}$$

3.3 Tensor Structure of the B-spline-based PDE Discretization

The discrete Ritz-Galerkin formulation (3.16) is inherently multidimensional. Indeed, the formulation has more dimensions than the initial PDE problem. In three-dimensional case the dimension of indices $\mathbf{k}, \mathbf{l}$ and $\mathbf{m}$ equal to 3, the naive multiplication of basis functions prior to the contraction with coefficients results in $3 \times 3 \times 3 = 9$ dimensions.

In order to deal with this situation, two main ideas are commonly used:

1. the basis functions are chosen with small support in order to make the discretization sparse and therefore the problem computationally feasible;

2. the multidimensional formulation is folded into sparse matrices and vectors to fit well-established routines of Matrix Algebra [20, 45, 25, 59], and afterwards, the solution is rearranged into the original dimensions of the problem.

However, the standard approach of matricizing the multidimensional formulation has its limitations. While it flattens and merges the different dimensions, the underlying structure (containing valuable information for efficient computations) appears to be hidden. Given this flattened representation, there is only limited room for optimization, mainly dealing with values and indices of the block-diagonal sparse matrix format. This format, however, typically has little in common with the original problem structure. Moreover, the structure of the (sparse) matrix needs to be represented, adding overhead to the implementation and rendering a software framework less generic.

Tensors preserve the dimensional structure and data coherence. While being slightly more complicated objects than matrices, tensors frequently allow for more elegant algorithms in a simpler way.

3.3.1 Multidimensional Tensor Indices

When manipulating with multidimensional data it is convenient to use multidimensional indexing. We introduce a convenient extension for multidimensional indices to the computational tensor framework. For tensors $\mathcal{A} \in \mathbb{R}^{K_1 \times K_2 \times K_3}$ and $\mathcal{B} \in \mathbb{R}^{K_1 \times K_2 \times K_3 \times L_1 \times L_2 \times L_3}$ the dot product along $k-$indices results in a tensor $\mathcal{C} \in \mathbb{R}^{L_1 \times L_2 \times L_3}$ that is defined as

$$\mathcal{C}^{l_1 l_2 l_3} = \mathcal{A}_{k_1 k_2 k_3} \mathcal{B}^{k_1 k_2 k_3 l_1 l_2 l_3}. \tag{3.17}$$

Combining indices l_1, l_2, l_3: $\mathbf{l} = l_1 l_2 l_3$ and k_1, k_2, k_3: $\mathbf{k} = k_1 k_2 k_3$ we end up with multidimensional indexing notation

$$\mathcal{C}^{\mathbf{l}} = \mathcal{A}_{\mathbf{k}} \mathcal{B}^{\mathbf{kl}}. \tag{3.18}$$

Therefore the multidimensional indices like $\mathbf{i} = i_1 i_2 ... i_N = ((1,1,..),(1,2,...),(2,1,...),...)$ will be used further in this work.

3.3.2 Tensor Structure of the Formulation

We apply the computational tensor algebra symbols to the Ritz-Galerkin formulation (3.16) and get the tensor formulation

$$\mathcal{C}^{\mathbf{k}} \mathcal{D}^{\mathbf{m}} \mathcal{W}_{\mathbf{klm}} + \mathcal{C}^{\mathbf{k}} \mathcal{M}^{\mathbf{m}} \mathcal{F}_{\mathbf{klm}} + \frac{1}{2} \mathcal{C}^{\mathbf{k}} \mathcal{H}_{\mathbf{kl}} = \mathcal{Q}^{\mathbf{j}} \mathcal{R}_{\mathbf{jl}} \quad \Leftrightarrow \quad F(\mathcal{C}) = \mathcal{T}, \tag{3.19}$$

where the tensors $\mathcal{W}_{\mathbf{klm}}$, $\mathcal{F}_{\mathbf{klm}}$, $\mathcal{H}_{\mathbf{kl}}$ and $\mathcal{R}_{\mathbf{jl}}$ represent the integrals in (3.16), the tensors $\mathcal{D}^{\mathbf{m}}$, $\mathcal{M}^{\mathbf{m}}$, $\mathcal{Q}^{\mathbf{j}}$, and $\mathcal{C}^{\mathbf{K}}$ represent the B-spline coefficients in the expansions of the PDE coefficients, source and the unknown correspondingly.

In expression (3.19), the multidimensional integrals are encapsulated, and arithmetic operations can be considered in terms of tensor algebra indices. First, the expression suggests that computations can be done either via an inner or an outer product. Second, it can be observed that different algorithms are defined where one of the indices $\mathbf{k}$, $\mathbf{l}$ or $\mathbf{m}$ is used in the algorithm's outermost loop.

The Number of Operations

We consider how a choice of a fixed index variance impacts on the number of performed operations. For the sake of simplicity we consider a two-dimensional case of the second term $\mathcal{C}^{\mathbf{k}} \mathcal{M}^{\mathbf{m}} \mathcal{F}_{\mathbf{klm}}$ in (3.19).

If we fix $k_1 k_2$ we get:

$$\mathcal{B}^{\tilde{l}_1 \tilde{l}_2} = \tilde{C} \mathcal{M}^{\tilde{m}_1 \tilde{m}_2} \mathcal{F}^{\tilde{l}_1}_{\tilde{m}_1} \mathcal{F}^{\tilde{l}_2}_{\tilde{m}_2}. \tag{3.20}$$

If we fix $l_1 l_2$ we get:

$$B = \mathcal{C}^{\tilde{k}_1 \tilde{k}_2} \mathcal{M}^{\tilde{m}_1 \tilde{m}_2} \mathcal{F}_{\tilde{k}_1 \tilde{m}_1} \mathcal{F}_{\tilde{k}_2 \tilde{m}_2}. \tag{3.21}$$

If we fix $m_1 m_2$ we get:

$$\mathcal{B}^{\tilde{l}_1 \tilde{l}_2} = \mathcal{C}^{\tilde{k}_1 \tilde{k}_2} \tilde{\mathcal{M}} \mathcal{F}^{\tilde{l}_1}_{\tilde{k}_1} \mathcal{F}^{\tilde{l}_2}_{\tilde{k}_2}. \tag{3.22}$$

Inner product reduces the dimensionality, the outer product expands the dimensionality. Giving a small span of indices for $\tilde{k}_1, \tilde{k}_2, \tilde{l}_1, \tilde{l}_2, \tilde{m}_1, \tilde{m}_2 = -1, 0, 1$ one can estimate the benefit of a certain algorithm as shown in Table 3.1.

Fixed ind.	Algorithm	Expand dim. ops.	Reduce dim. ops.	Gain
$k_1 k_2$	$\mathcal{B}^{\tilde{l}_1 \tilde{l}_2} = C \mathcal{M}^{\tilde{m}_1 \tilde{m}_2} \mathcal{F}^{\tilde{l}_1}_{\tilde{m}_1} \mathcal{F}^{\tilde{l}_2}_{\tilde{m}_2}$	$171_{\otimes}$ $90_{\oplus}$	$63_{\otimes}$ $54_{\oplus}$	$2.7_{\otimes}$ $1.6_{\oplus}$
$l_1 l_2$	$B = \mathcal{C}^{\tilde{k}_1 \tilde{k}_2} \mathcal{M}^{\tilde{m}_1 \tilde{m}_2} \mathcal{F}_{\tilde{k}_1 \tilde{m}_1} \mathcal{F}_{\tilde{k}_2 \tilde{m}_2}$	$171_{\otimes}$ $90_{\oplus}$	$63_{\otimes}$ $63_{\oplus}$	$2.7_{\otimes}$ $1.4_{\oplus}$
$m_1 m_2$	$\mathcal{B}^{\tilde{l}_1 \tilde{l}_2} = \mathcal{C}^{\tilde{k}_1 \tilde{k}_2} M \mathcal{F}^{\tilde{l}_1}_{\tilde{k}_1} \mathcal{F}^{\tilde{l}_2}_{\tilde{k}_2}$	$171_{\otimes}$ $90_{\oplus}$	$63_{\otimes}$ $54_{\oplus}$	$2.7_{\otimes}$ $1.6_{\oplus}$

Table 3.1: The number of operations (multiplications "$\otimes$" and additions "$\oplus$") depending on the order of tensor contractions.

3.3.3 Tensor B-spline Kernels

When the system of equations (3.19) has a large number of unknowns, the usual approach is an application of an iterative solver. In such a solver, it is critical to compute the system operator $F(\mathcal{C})$ as efficiently as possible. At first glance, the tensors $\mathcal{W}^{\mathbf{klm}}$, $\mathcal{F}^{\mathbf{klm}}$ could be of a large size and their direct computation appears to be intractable due to huge memory requirements. However, due to the finite support of B-spline functions these tensors have a large number of zeros, i.e. they are sparse. Moreover, the non-zero values are localized around the grid nodes and are translation invariant within the domain. Therefore, they have a kernel-like structure, where the width of these kernels depends on the B-spline degree.

For the sake of simplicity, we take $n = n_b = n_p$ and $h = 1$. We define the support of the product of three B-splines of degree n as Ω^s. Consider the following integral from (3.16):

$$\int_{\Omega^s} \left[\nabla\beta^n_{\mathbf{k}}(\mathbf{x}) \cdot \nabla\beta^n_{\mathbf{l}}(\mathbf{x})\right] \beta^n_{\mathbf{m}}(\mathbf{x}) d\mathbf{x}. \tag{3.23}$$

Consider a three-dimensional case. With the use of B-spline separability

$$\beta^n_{\mathbf{k}}(\mathbf{x}) = \beta^n_{k_1}(x_1)\beta^n_{k_2}(x_2)\beta^n_{k_3}(x_3) \tag{3.24}$$

the scalar product of gradients $\nabla\beta^n_{\mathbf{k}}(\mathbf{x}) \cdot \nabla\beta^n_{\mathbf{l}}(\mathbf{x})$ becomes

$$\begin{aligned}
\nabla\{\beta^n_{k_1}(x_1)\beta^n_{k_2}(x_2)\beta^n_{k_3}(x_3)\} \cdot \nabla\{\beta^n_{l_1}(x_1)\beta^n_{l_2}(x_2)\beta^n_{l_3}(x_3)\} = \\
\begin{pmatrix} \frac{d\beta^n_{k_1}(x_1)}{dx_1}\beta^n_{k_2}(x_2)\beta^n_{k_3}(x_3) & \beta^n_{k_1}(x_1)\frac{d\beta^n_{k_2}(x_2)}{dx_2}\beta^n_{k_3}(x_3) & \beta^n_{k_1}(x_1)\beta^n_{k_2}(x_2)\frac{d\beta^n_{k_3}(x_3)}{dx_3} \end{pmatrix} \cdot \\
\begin{pmatrix} \frac{d\beta^n_{l_1}(x_1)}{dx_1}\beta^n_{l_2}(x_2)\beta^n_{l_3}(x_3) & \beta^n_{l_1}(x_1)\frac{d\beta^n_{l_2}(x_2)}{dx_2}\beta^n_{l_3}(x_3) & \beta^n_{l_1}(x_1)\beta^n_{l_2}(x_2)\frac{d\beta^n_{l_3}(x_3)}{dx_3} \end{pmatrix} = \\
\frac{d\beta^n_{k_1}(x_1)}{dx_1}\frac{d\beta^n_{l_1}(x_1)}{dx_1}\beta^n_{k_2}(x_2)\beta^n_{l_2}(x_2)\beta^n_{k_3}(x_3)\beta^n_{l_3}(x_3) + \\
\beta^n_{k_1}(x_1)\beta^n_{l_1}(x_1)\frac{d\beta^n_{k_2}(x_2)}{dx_2}\frac{d\beta^n_{l_2}(x_2)}{dx_2}\beta^n_{k_3}(x_3)\beta^n_{l_3}(x_3) + \\
\beta^n_{k_1}(x_1)\beta^n_{l_1}(x_1)\beta^n_{k_2}(x_2)\beta^n_{l_2}(x_2)\frac{d\beta^n_{k_3}(x_3)}{dx_3}\frac{d\beta^n_{l_3}(x_3)}{dx_3}
\end{aligned} \tag{3.25}$$

Then we can write the expression (3.23) as

$$\begin{aligned}
&\int_{\Omega^s} \frac{d\beta^n_{k_1}(x_1)}{dx_1}\frac{d\beta^n_{l_1}(x_1)}{dx_1}\beta^n_{m_1}(x_1)dx_1 \int_{\Omega^s} \beta^n_{k_2}(x_2)\beta^n_{l_2}(x_2)\beta^n_{m_2}(x_2)dx_2 \int_{\Omega^s} \beta^n_{k_3}(x_3)\beta^n_{l_3}(x_3)\beta^n_{m_3}(x_3)dx_3 + \\
&\int_{\Omega^s} \beta^n_{k_1}(x_1)\beta^n_{l_1}(x_1)\beta^n_{m_1}(x_1)dx_1 \int_{\Omega^s} \frac{d\beta^n_{k_2}(x_2)}{dx_2}\frac{d\beta^n_{l_2}(x_2)}{dx_2}\beta^n_{m_2}(x_2)dx_2 \int_{\Omega^s} \beta^n_{k_3}(x_3)\beta^n_{l_3}(x_3)\beta^n_{m_3}(x_3)dx_3 + \\
&\int_{\Omega^s} \beta^n_{k_1}(x_1)\beta^n_{l_1}(x_1)\beta^n_{m_1}(x_1)dx_1 \int_{\Omega^s} \beta^n_{k_2}(x_2)\beta^n_{l_2}(x_2)\beta^n_{m_2}(x_2)dx_2 \int_{\Omega^s} \frac{d\beta^n_{k_3}(x_3)}{dx_3}\frac{d\beta^n_{l_3}(x_3)}{dx_3}\beta^n_{m_3}(x_3)dx_3.
\end{aligned} \tag{3.26}$$

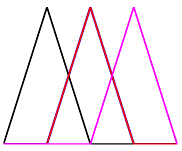
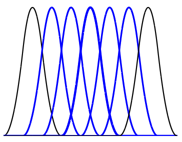
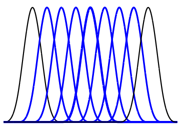
The expression (3.26) consists of two types of integrals

$$\widehat{\mathcal{W}}_{klm} = \int_{\Omega^s} \frac{\partial}{\partial x}\beta^n_{k,h}(x)\frac{\partial}{\partial x}\beta^n_{l,h}(x)\beta^n_{m,h}(x)dx, \tag{3.27}$$

$$\widehat{\mathcal{F}}_{klm} = \int_{\Omega^s} \beta^n_{k,h}(x)\beta^n_{l,h}(x)\beta^n_{m,h}(x)dx, \tag{3.28}$$

we call them tensor B-spline kernels. Due to the local support of B-splines these kernels expand over a limited span of indices k, l and m. Table 3.2 and Fig. 3.1 show examples of one- and two- dimensional B-spline kernels.

Table 3.2: B-splines functions of degree $n = 1, 2, 3$ and corresponding B-spline kernels $\hat{w}$ and $\hat{f}$ in the one-dimensional case for a fixed index l. The spans of indices corresponding to non-zero values of the kernels $\hat{\omega}_{km}$, $\hat{f}_{km}$ are designated as $\tilde{k}$, $\tilde{m}$.

n	**Basis functions**			**Plot of B-splines**	$\tilde{k}$	$\tilde{m}$	**Size of** $\hat{\omega}_{km}$, $\hat{f}_{km}$
1	$\beta^1_{\tilde{k},h}(x)$	$\beta^1_h(x)$	$\beta^1_{\tilde{m},h}(x)$		-1..1	-1..1	3×3
2	$\beta^2_{\tilde{k},h}(x)$	$\beta^2_h(x)$	$\beta^2_{\tilde{m},h}(x)$		-2..2	-2..2	5×5
3	$\beta^3_{\tilde{k},h}(x)$	$\beta^3_h(x)$	$\beta^3_{\tilde{m},h}(x)$		-3..3	-3..3	7×7

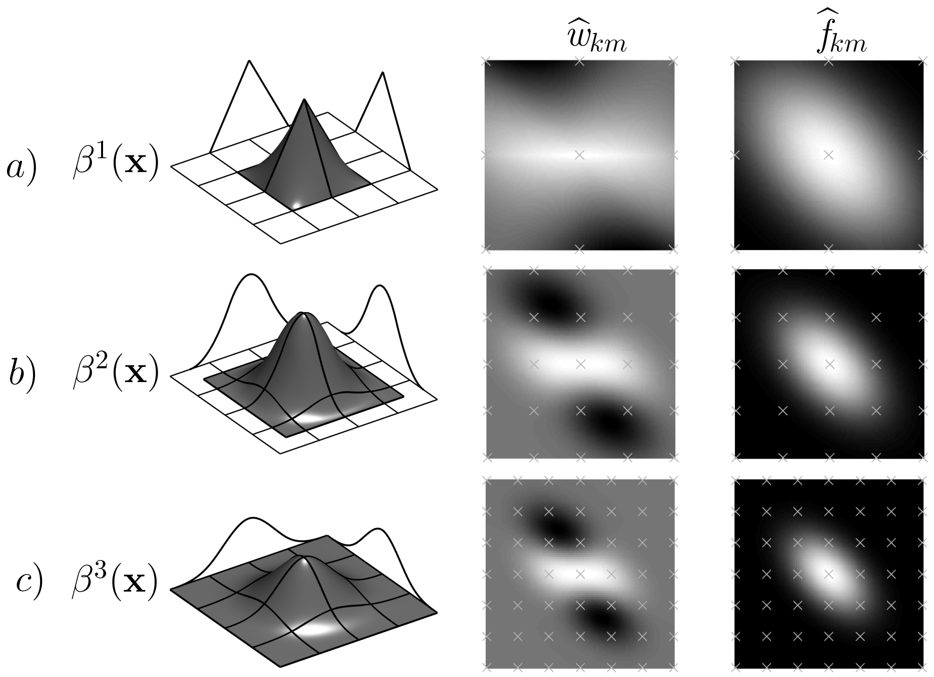

$\widehat{w}_{km}$
$\widehat{f}_{km}$
$a)$ $\beta^1(\mathbf{x})$
$b)$ $\beta^2(\mathbf{x})$
$c)$ $\beta^3(\mathbf{x})$

Figure 3.1: Bi-variate tensor-product B-splines corresponding tri-product kernels $\widehat{\omega}_{km} = \int_{-\infty}^{+\infty} \frac{\partial}{\partial x}\beta^n_{k,1}(x)\frac{\partial}{\partial x}\beta^n_{0,1}(x)\beta^n_{m,1}(x)dx$ and $\widehat{f}_{km} = \int_{-\infty}^{+\infty} \beta^n_{k,1}(x)\beta^n_{0,1}(x)\beta^n_{m,1}(x)dx$; $x,\ k,\ m\ \in \mathbb{R}$. (a) Linear $n = 1$, (b) quadratic $n = 2$, (c) cubic $n = 3$ cases. Support of $\beta^1(\mathbf{x})$, $\beta^2(\mathbf{x})$ and $\beta^3(\mathbf{x})$ is $[-1..1]$, $[-1.5..1.5]$ and $[-2..2]$ in each dimension respectively. Crosses in the kernel images correspond to integer shifts.

Remarkably, for B-splines fully inside the domain Ω, the kernels are translation-invariant and separable:

$$\hat{\mathcal{W}}_{\mathbf{klm}} = (1/h^2)\hat{w}_{k_1l_1m_1}\cdots\hat{f}_{k_dl_dm_d} + ... + (1/h^2)\hat{f}_{k_1l_1m_1}\cdots\hat{w}_{k_dl_dm_d} \tag{3.29}$$

$$\hat{\mathcal{F}}_{\mathbf{klm}} = \hat{f}_{k_1l_1m_1}\cdots\hat{f}_{k_dl_dm_d}. \tag{3.30}$$

This property allows efficient computations in the domain, even when high-degree B-splines are used.

3.3.4 Two-dimensional Case

Let $\mathbb{K} = K_1 \times K_2 \subset \mathbb{N}^2$, $\mathbb{L} = L_1 \times L_2 \subset \mathbb{N}^2$, $\mathbb{M} = M_1 \times M_2 \subset \mathbb{N}^2$, $K_1 = L_1 = M_1, K_2 = L_2 = M_2$, where $[K_1, K_2]$ is the size of the used rectangular grid. For an index $\mathbf{l} \in \mathbb{L}$ we define tensors $w_{\mathbf{km}} = \mathcal{W}_{\mathbf{klm}}|\mathbf{l}$, $f_{\mathbf{km}} = \mathcal{F}_{\mathbf{klm}}|\mathbf{l}$, $h_{\mathbf{k}} = h_{\mathbf{kl}}|\mathbf{l}$ and $r_{\mathbf{j}} = \mathcal{R}_{\mathbf{jl}}|\mathbf{l}$. Then for each $\mathbf{l}$, equation (3.19) reduces to

$$\mathcal{C}^{\mathbf{k}}\mathcal{D}^{\mathbf{m}}w_{\mathbf{km}} + \mathcal{C}^{\mathbf{k}}\mathcal{M}^{\mathbf{m}}f_{\mathbf{km}} + \frac{1}{2}\mathcal{C}^{\mathbf{k}}h_{\mathbf{k}} = \mathcal{Q}^{\mathbf{j}}r_{\mathbf{j}}, \mathbf{k} \in \mathbb{K}, \mathbf{m} \in \mathbb{M}. \tag{3.31}$$

Tensor kernels $w_{\mathbf{km}}$, $f_{\mathbf{km}}$, $h_{\mathbf{k}}$ and $r_{\mathbf{j}}$ in (3.31) have non-zero values only within sub-regions $[\mathbf{l}-n_b..\mathbf{l}+n_b, \mathbf{l}-r_{bp}..\mathbf{l}+r_{bp}]$, $[\mathbf{l}-n_b..\mathbf{l}+n_b]$ and $[\mathbf{l}-r_{bs}..\mathbf{l}+r_{bs}]$ respectively ($r_{bp} = \lfloor 0.5(n_b+n_p+1)\rfloor$, $r_{bs} = \lfloor 0.5(n_b + n_s + 1)\rfloor$). Kernels are classified into two types: 1) domain kernels $\widehat{w}_{\mathbf{km}}$, $\widehat{f}_{\mathbf{km}}$ and $\widehat{r}_{\mathbf{j}}$, which correspond to B-splines fully inside Ω, 2) boundary kernels $\overline{w}_{\mathbf{km}}$, $\overline{f}_{\mathbf{km}}$, $h_{\mathbf{k}}$ and $\overline{r}_{\mathbf{j}}$, which correspond to B-splines intersecting $\partial\Omega$.

For the domain kernels in the two-dimensional case we get

$$\widehat{w}_{\mathbf{km}} = (1/h_1)\widehat{w}_{k_1m_1}\widehat{f}_{k_2m_2} + (1/h_2)\widehat{f}_{k_1m_1}\widehat{w}_{k_2m_2}, \tag{3.32}$$

$$\widehat{f}_{\mathbf{km}} = \widehat{f}_{k_1m_1}\widehat{f}_{k_2m_2}, \tag{3.33}$$

$$\widehat{r}_{\mathbf{j}} = \widehat{r}_{j_1}\widehat{r}_{j_2}, \tag{3.34}$$

with the components shown in Fig. 3.1.

While B-spline properties provide advantages for computations inside the domain, special treatment is needed for boundary handling. Boundary tensor kernels are non-separable and have to be computed for each boundary node, though the number of boundary kernels is usually only a small fraction of all kernels as shown further in this chapter.

Structure of the System Operator

After all tensor kernels are computed and PDE coefficients and source are represented using B-splines, the solution $\mathcal{C}^{\mathbf{k}}$ in (3.31) can be computed by a linear solver, e.g. the Conjugate Gradient (CG) solver. In order to assemble the system operator $F(\mathcal{C})$, B-spline coefficients of diffusion $\mathcal{D}$ and absorption $\mathcal{M}$ have to be multiplied with kernels w and f respectively and summed up with the boundary kernel h, as shown in Algorithm 1. In the two-dimensional case, computations result in a $L_1 \times L_2$ set of kernels $\mathcal{T}$ of size $(2n_b+1) \times (2n_b+1)$ (cf. Fig. 3.2 (c)).

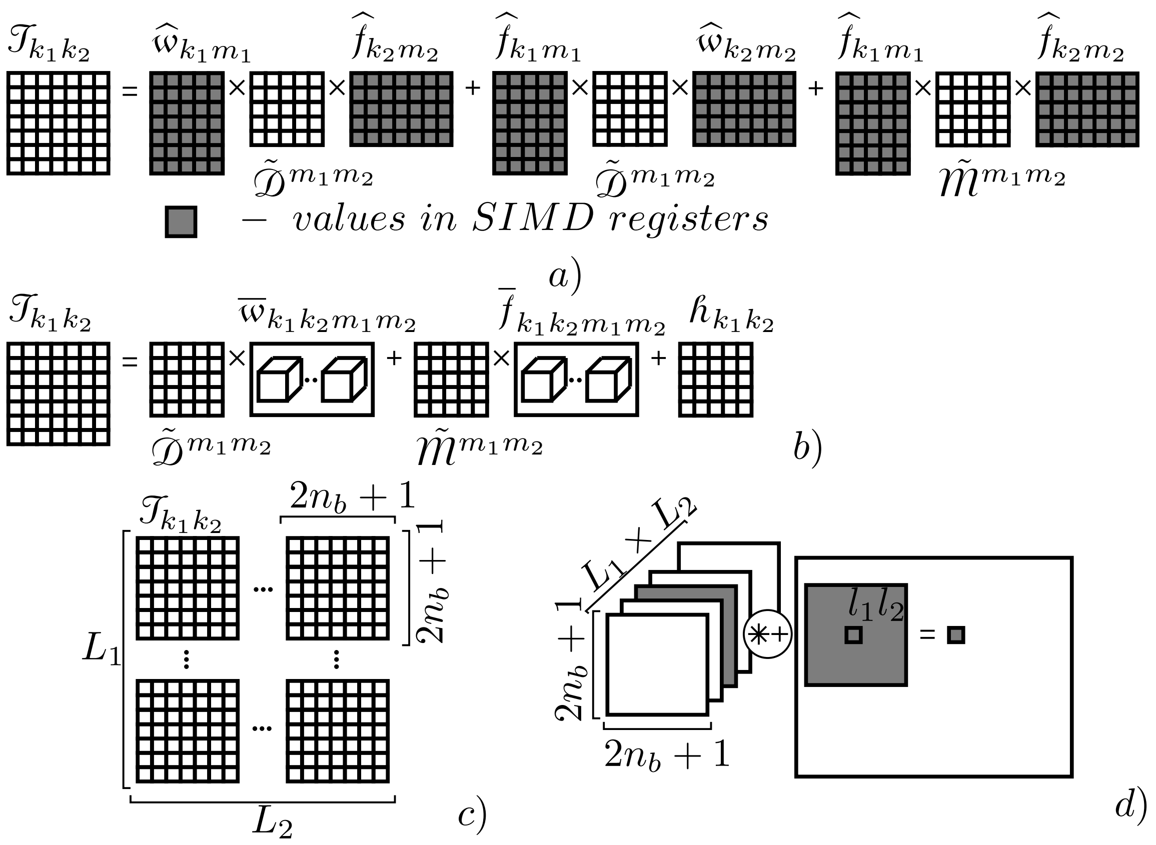


Figure 3.2: System operator computation and evaluation for $n_b = 3, n_p = 1$ (a) Tensor computations within the domain are implementable by SIMD, (b) Tensor computations on the boundary, (c) The system operator tensor, (d) System operator evaluation by convolution.

Algorithm 1 System operator computation in 2-D.

```
for $l_1 = 1..L_1$ do
    for $l_2 = 1..L_2$ do
        $\tilde{\mathcal{D}} \leftarrow \mathcal{D}_{l_1-r_{bp}..l_1+r_{bp},l_2-r_{bp}..l_2+r_{bp}}$
        $\tilde{m} \leftarrow m_{l_1-r_{bp}..l_1+r_{bp},l_2-r_{bp}..l_2+r_{bp}}$
        if $(l_1, l_2) \in Domain$ then                      ▷ domain computations
            $$\mathcal{T}_{k_1k_2} \leftarrow \hat{\omega}_{k_1m_1}\tilde{\mathcal{D}}^{m_1m_2}\hat{f}_{k_2m_2} + \hat{f}_{k_1m_1}\tilde{\mathcal{D}}^{m_1m_2}\hat{\omega}_{k_2m_2} + \hat{f}_{k_1m_1}\tilde{m}^{m_1m_2}\hat{f}_{k_2m_2}$$
        else if $(l_1, l_2) \in Boundary$ then               ▷ boundary computations
            $$\mathcal{T}_{k_1k_2} \leftarrow \tilde{\mathcal{D}}^{m_1m_2}\overline{\omega}_{k_1k_2m_1m_2} + \tilde{m}^{m_1m_2}\overline{f}_{k_1k_2m_1m_2} + h_{k_1k_2}$$
        end if
        $\mathcal{P}_{k_1k_2l_1l_2} \leftarrow \mathcal{T}_{k_1k_2}$
    end for
end for
```

A sparse system matrix of size $(L_1L_2) \times (L_1L_2)$ could be constructed here. However, keeping the dense data structure of the tensor $\mathcal{P}_{k_1k_2l_1l_2}$ (Fig. 3.2 (d)) allows optimal storage and block-wise access patterns. Then system operator evaluation at each linear solver iteration is defined by

$$\mathcal{Y}_{l_1 l_2} = \mathcal{P}_{k_1 k_2 l_1 l_2} \mathcal{C}^{k_1 k_2}, \tag{3.35}$$

that is shown as pseudo-code in the Algorithm 2 and graphically in Fig. 3.2 (d). From the signal processing viewpoint, this corresponds to convolution with a spatially varying two-dimensional finite impulse response. By preserving kernel-based structure and data locality, efficient memory transfers and high level of parallelism can be achieved. The structure of the tensor $\mathcal{R}$ allows efficient computation of the right-hand side in (3.19) by FIR filtering.

On hardware with limited memory bandwidth but multiple cores, the performance of the algorithm may be optimized by choosing the on-the-fly strategy, where the coefficients of the tensor $\mathcal{P}_{k_1 k_2 l_1 l_2}$ can be recomputed at every linear solver iteration based on the Algorithm 1, without storing the tensor explicitly. The fact that such on-the-fly computations heavily rely on small dense matrix multiplication (cf. Fig. 3.2 (a, b)) allows efficient code vectorization, e.g. through the use of SIMD. In this case, the spatially invariant domain tensor kernels can be stored directly in vector registers (e.g. Intel AVX or GPU registers), implying a significant reduction in memory transfers, and translating to efficient utilization of the parallel hardware. The next chapter provides more detailed description of matrix-free algorithms, and performs their evaluation and comparison.

Algorithm 2 System operator evaluation in 2-D.

```
for $l_1 = 1..L_1$ do
    for $l_2 = 1..L_2$ do
        $\tilde{\mathcal{C}} \leftarrow \mathcal{C}_{l_1 - n_b..l_1 + n_b, l_2 - n_b..l_2 + n_b}$
        $\tilde{\mathcal{P}}_{k_1 k_2} \leftarrow \mathcal{P}_{..,..,l_1 l_2}$
        $\mathcal{Y}_{l_1 l_2} \leftarrow \tilde{\mathcal{C}}^{k_1 k_2} \tilde{\mathcal{P}}_{k_1 k_2}$
    end for
end for
```

3.3.5 Three-dimensional Case

Let $\mathbb{K} = K_1 \times K_2 \times K_3 \subset \mathbb{N}^3$, $\mathbb{L} = L_1 \times L_2 \times L_3 \subset \mathbb{N}^3$, $\mathbb{M} = M_1 \times M_2 \times M_3 \subset \mathbb{N}^3$, $K_1 = L_1 = M_1, K_2 = L_2 = M_2, K_3 = L_3 = M_3$, where $[K_1, K_2, K_3]$ is the size of the used rectangular grid. For a given grid node with index $\mathbf{l} \in \mathbb{L}$, we define tensors $\omega_{\mathbf{km}} = \mathcal{W}_{\mathbf{klm}}|\mathbf{l}$, $f_{\mathbf{km}} = \mathcal{F}_{\mathbf{klm}}|\mathbf{l}$, $h_{\mathbf{k}} = \mathcal{H}_{\mathbf{kl}}|\mathbf{l}$ and $r_{\mathbf{j}} = \mathcal{R}_{\mathbf{jl}}|\mathbf{l}$. Then for each $\mathbf{l}$ equation (3.19) becomes

$$\mathcal{C}^{\mathbf{k}} \mathcal{D}^{\mathbf{m}} \omega_{\mathbf{km}} + \mathcal{C}^{\mathbf{k}} \mathcal{M}^{\mathbf{m}} f_{\mathbf{km}} + \frac{1}{2} \mathcal{C}^{\mathbf{k}} h_{\mathbf{k}} = \mathcal{Q}^{\mathbf{j}} r_{\mathbf{j}}. \tag{3.36}$$

The tensors $\omega_{\mathbf{km}}$, $f_{\mathbf{km}}$, $h_{\mathbf{k}}$ and $r_{\mathbf{j}}$ in (3.36) have non-zero values only within sub-regions $[\mathbf{l} - n_b..\mathbf{l} + n_b, \mathbf{l} - r_{bp}..\mathbf{l} + r_{bp}]$, $[\mathbf{l} - n_b..\mathbf{l} + n_b]$ and $[\mathbf{l} - r_{bs}..\mathbf{l} + r_{bs}]$ respectively ($r_{bp} = \lfloor 0.5(n_b + n_p + 1) \rfloor$, $r_{bs} = \lfloor 0.5(n_b + n_s + 1) \rfloor$). Kernels can be of two types: 1) domain kernels $\widehat{\omega}_{\mathbf{km}}$, $\widehat{f}_{\mathbf{km}}$ and $\widehat{r}_{\mathbf{j}}$, which correspond to B-splines fully inside Ω, and 2) boundary kernels $\overline{\omega}_{\mathbf{km}}$, $\overline{f}_{\mathbf{km}}$, $\overline{h}_{\mathbf{k}}$ and $\overline{r}_{\mathbf{j}}$, which correspond to B-splines intersecting $\partial\Omega$. Domain kernels are separable

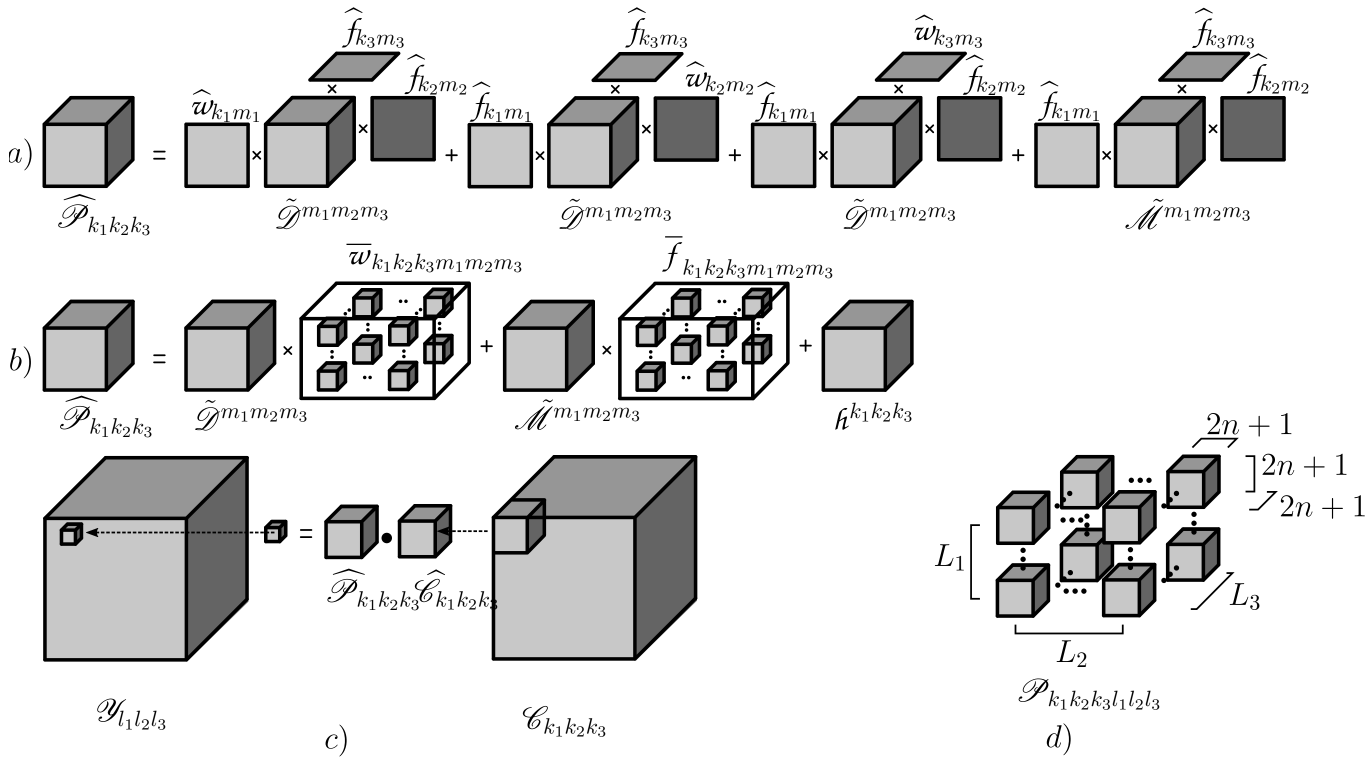
$$\widehat{\omega}_{\mathbf{km}} = (1/h_1^2) \widehat{\omega}_{k_1 m_1} \widehat{f}_{k_2 m_2} \widehat{f}_{k_3 m_3} + (1/h_2^2) \widehat{f}_{k_1 m_1} \widehat{\omega}_{k_2 m_2} \widehat{f}_{k_3 m_3} + (1/h_3^2) \widehat{f}_{k_1 m_1} \widehat{f}_{k_2 m_2} \widehat{\omega}_{k_3 m_3},$$

$$\widehat{f}_{\mathbf{km}} = \widehat{f}_{k_1 m_1} \widehat{f}_{k_2 m_2} \widehat{f}_{k_3 m_3}, \tag{3.37}$$

$$\widehat{r}_{\mathbf{j}} = \widehat{r}_{j_1} \widehat{r}_{j_2} \widehat{r}_{j_3}. \tag{3.38}$$

Structure of the System Operator

Convolutional structure and sparsity of (3.19), and separability of domain kernels along with the natural tensor structure of Ritz-Galerkin formulation of the problem (shown in Fig. 3.3) are the key factors for designing an efficient computational algorithm to solve for $\mathcal{C}^{\mathbf{k}}$. It is interesting to note that the terms in the domain part of the algorithm (Fig. 3.3, (a)) have a structure that corresponds to the tensor factorization [13].



a)
b)
c)
d)
$\widehat{\mathscr{P}}_{k_1k_2k_3}$
$\widehat{w}_{k_1m_1}$
$\widehat{f}_{k_3m_3}$
$\widehat{f}_{k_2m_2}$
$\widehat{w}_{k_2m_2}$
$\widehat{w}_{k_3m_3}$
$\widehat{f}_{k_1m_1}$
$\tilde{\mathscr{D}}^{m_1m_2m_3}$
$\tilde{\mathscr{M}}^{m_1m_2m_3}$
$\overline{w}_{k_1k_2k_3m_1m_2m_3}$
$\overline{f}_{k_1k_2k_3m_1m_2m_3}$
$h^{k_1k_2k_3}$
$\mathscr{Y}_{l_1l_2l_3}$
$\mathscr{C}_{k_1k_2k_3}$
$\widehat{\mathscr{P}}_{k_1k_2k_3}\widehat{\mathscr{C}}_{k_1k_2k_3}$
$2n+1$
L_1
L_2
L_3
$\mathscr{P}_{k_1k_2k_3l_1l_2l_3}$

Figure 3.3: (a) Tensor structure of computations for separable domain kernels corresponding to block term decomposition, (b) structure of computations for non-separable boundary kernels, (c) convolution strategy (corresponds to filtering with a spatially varying impulse response), (d) structure of the system operator. The symbol "$\times$" designates the tensor-matrix product, symbol "$\bullet$" designates the scalar product.

A real-life three-dimensional problem may result in a large system of equations. This requires the use of memory efficient iterative methods, e.g. the Conjugate Gradient (CG) method. In this method, a solution $\mathcal{C}^{\mathbf{k}}$ is iteratively refined via the system operator evaluation. The system operator $\mathscr{P}_{\mathbf{kl}} = \mathcal{D}^{\mathbf{m}}\mathscr{W}_{\mathbf{klm}} + \mathcal{M}^{\mathbf{m}}\mathscr{F}_{\mathbf{klm}} + \frac{1}{2}\mathscr{H}_{\mathbf{kl}}$ is assembled from products of coefficients $\mathcal{D}$ and $\mathcal{M}$ with respective kernels w and f and incorporates kernels h (Fig. 3.3, a, b) and corresponds to an $L_1 \times L_2 \times L_3$ set of kernels $\widehat{\mathscr{P}}_{\mathbf{k}}$ (Fig. 3.3, d). The system operator can be assembled before the iterative solving (Fig. 3.3, c) (convolution strategy), or can be assembled and convolved at each iteration ("on-the-fly" strategy). These two strategies are potentially beneficial as they use the natural tensor structure of the problem that allows optimal storage, data locality, efficient block-wise memory access and good parallelism in contrast to a standard sparse matrix-vector multiplication (SpMV) approach (although it is still possible to apply SpMV by rearranging the system operator into a sparse matrix).

3.3.6 Domain Tensors Filters

The kernel-based decomposition structure of the system operator induces filtering-like algorithms, which makes it possible to define algorithms from a signal processing viewpoint. The translation invariant kernels are the crucial component in these efficient filtering-like algorithms. Inside the domain, the operations are represented by multilinear convolution, with shift-invariant separable kernels, as depicted in Fig. 3.4.

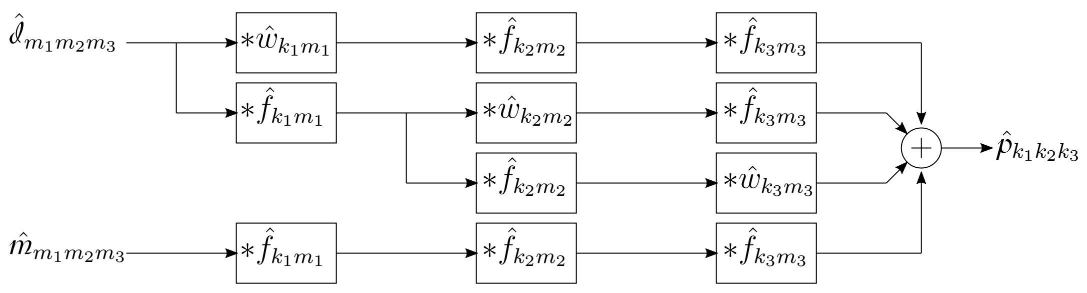


Figure 3.4: Domain computations are implemented as filtering algorithm.

3.4 Numerical Integration

Efficient integration is an important building block for any PDE numerical method. The Tensor B-spline method uses shift-invariant kernels inside the domain for which the integration performed only once and the values of domain tensor kernels are reused. For the boundary kernels, the situation is different. Boundary tensor kernels are not shift-invariant and non-separable and have to be computed for each boundary node, though the number of boundary kernels is usually only a fraction of all kernels. They could be computed cell-wise by standard numerical quadrature as done in [46]. Here, we introduce a different approach that exploits the Divergence Theorem and the properties of B-splines for integration over boundary cells.

3.4.1 Domain Kernels

Separable domain kernels

$$\widehat{w}_{km} = \int_{-\infty}^{+\infty} \frac{\partial}{\partial x}\beta_{k,1}^{n_b}(x)\frac{\partial}{\partial x}\beta_{0,1}^{n_b}(x)\beta_{m,1}^{n_p}(x)dx, \tag{3.39}$$

$$\widehat{f}_{km} = \int_{-\infty}^{+\infty} \beta_{k,1}^{n_b}(x)\beta_{0,1}^{n_b}(x)\beta_{m,1}^{n_p}(x)dx, \tag{3.40}$$

$$\widehat{\tau}_{j} = \int_{-\infty}^{+\infty} \beta_{j,1}^{n_s}(x)\beta_{0,1}^{n_b}(x)dx, \tag{3.41}$$

are computed only once an are then reused everywhere in the domain. They can be computed analytically using the properties of B-splines or using standard quadratures.

3.4.2 Boundary Kernels

When the domain boundary is of arbitrary shape, the rectangular grid of the B-spline basis does not conform to the boundary geometry. Therefore, B-splines will be truncated on the boundary. These truncated B-splines result in non-separable kernels.

With the increase of the problem size (number of degrees of freedom) the number of domain kernels grows much faster than the number of boundary kernels. As an example, Fig. 3.5, and Fig. 3.6 show the boundary kernels percentage for different problem sizes and B-spline degrees.

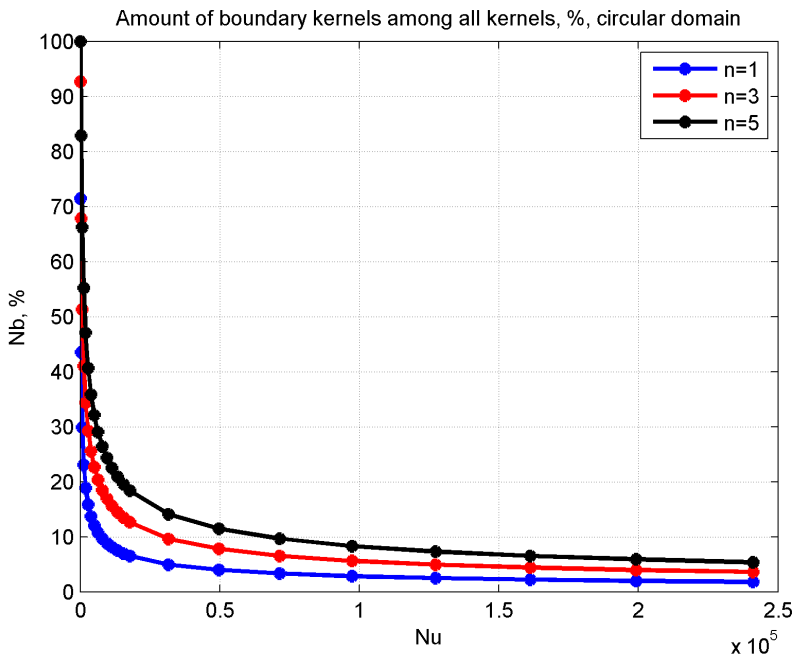


Figure 3.5: Number of boundary kernels (N_b) in % for B-splines of degrees $n = 1$, $n = 3$ and $n = 5$ for different degrees of freedom (circular domain) versus the number of unknowns (N_u).

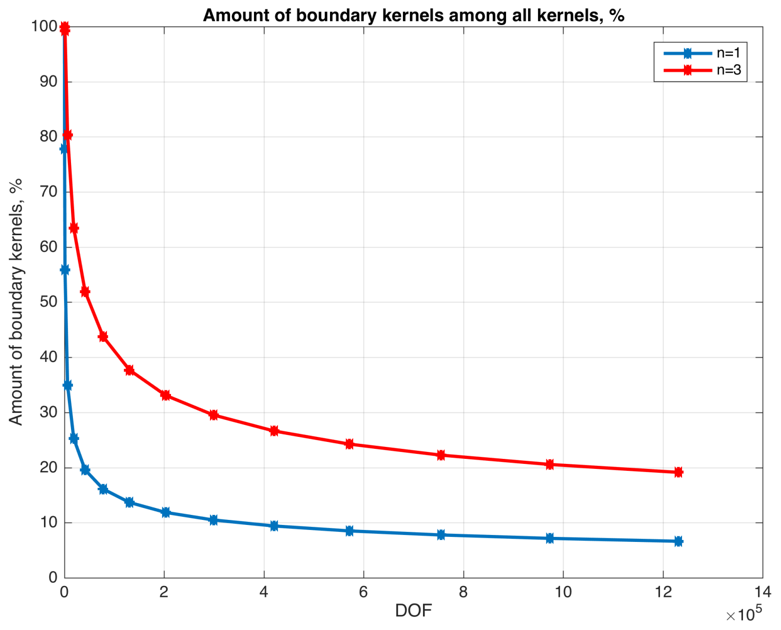


Figure 3.6: Number of boundary kernels (N_b) in % for B-splines of degrees $n = 1$ and $n = 3$ for different degrees of freedom (spherical domain).

The relative amount of boundary kernels becomes very small (in the two-dimensional case $\approx 1-2\%$, and in the three-dimensional case $\approx 10-20\%$ for circular, and for spherical domains, respectively, with approx. 10E6 unknowns). This means that for very large problem sizes, the relative amount of integration drops because domain kernels are computed only once.

3.4.3 Integration Using the Divergence Theorem

We propose an efficient integration method for the boundary tensor B-spline kernels. The method is based on the Divergence Theorem and properties of B-splines.

For a continuously differentiable vector field $\mathbf{g}(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, on a neighborhood of $V \in \mathbb{R}^3$, which is closed by a piecewise smooth boundary S (closed surface), the Divergence Theorem states that

$$\int_V (\nabla \cdot \mathbf{g}(\mathbf{x}))dV = \oint_S (\mathbf{g}(\mathbf{x}) \cdot \mathbf{n})dS, \tag{3.42}$$

where $\mathbf{n}$ is the outward unit normal to the boundary surface S (see Fig. 3.7). The left side of the equation (3.42) contains a volume integral over the V with the divergence of the vector field $\mathbf{g}$ as integrand, the right side of (3.42) contains surface integral over the boundary S that encloses the volume V.

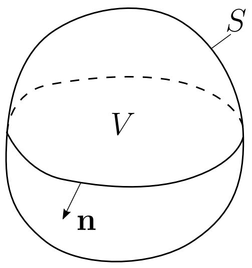


Figure 3.7: Some volume V bounded by a surface S, and normal $\mathbf{n}$ to a surface S.

We re-write the expression above using vector field components:

$$\int_V \left(\frac{\partial g_1}{\partial x_1} + \frac{\partial g_2}{\partial x_2} + \frac{\partial g_3}{\partial x_3} \right) dV = \oint_S (g_1 n_1 + g_2 n_2 + g_3 n_3)dS. \tag{3.43}$$

Let the vector field $\mathbf{g}$ has only one non-zero component, $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), 0, 0)$. Then (3.43) reduces to

$$\int_V \frac{\partial g_1}{\partial x_1} dV = \oint_S g_1 n_1 dS. \tag{3.44}$$

In case when a scalar field g_1 is constructed via tensor product, i.e. g_1 is separable

$$g_1(\mathbf{x}) = f(x_1)f(x_2)f(x_3), \tag{3.45}$$

we can write

$$\int_V f(x_1)f(x_2)f(x_3)dV = \oint_S F(x_1)f(x_2)f(x_3)n_1 dS, \tag{3.46}$$

where $F(x) = \int f(x)dx$ is the antiderivative of $f(x)$.

We introduce $dV = dx_1 dx_2 dx_3$ and $n_1 dS = dx_2 dx_3$. Then we write

$$\iiint_V f(x_1)f(x_2)f(x_3)dx_1 dx_2 dx_3 = \oiint_S F(x_1)f(x_2)f(x_3)dx_2 dx_3. \tag{3.47}$$

We use parametric representation of the surface S:

$$x_2 = x_2(t_1, t_2), \tag{3.48}$$
$$x_3 = x_3(t_1, t_2), \tag{3.49}$$

with the corresponding Jacobian

$$J = \begin{bmatrix} \frac{\partial x_2}{\partial t_1} \frac{\partial x_2}{\partial t_2} \\ \frac{\partial x_3}{\partial t_1} \frac{\partial x_3}{\partial t_2} \end{bmatrix}. \tag{3.50}$$

Finally, the integration is performed as

$$\begin{aligned} \oiint_S F(x_1)f(x_2)f(x_3)dx_2dx_3 = \\ \oiint_S F_{x_1}(x_1)f_{x_2}(x_2)f_{x_3}(x_3)|J|dt_1dt_2 = \\ \oiint_S F_{x_1}[x_1(t_1,t_2)]f_{x_2}[x_2(t_1,t_2)]f_{x_3}[x_3(t_1,t_2)]\left(\frac{\partial x_2}{\partial t_1}\frac{\partial x_3}{\partial t_2} - \frac{\partial x_2}{\partial t_2}\frac{\partial x_3}{\partial t_1}\right)dt_1dt_2 \end{aligned} \tag{3.51}$$

The described integration method enables the volumetric integration of a multivariate separable function $g(\mathbf{x})$ via integration over the surface S that encloses this volume. The method effectively reduces the number of dimensions over which the integration is performed by one. In the three-dimensional case, the proposed method does not require building explicit 3-D quadratures as in [46] but uses 2-D quadratures shared for computation of all volume and surface integrals in (3.16) for a given curvilinear segment.

3.4.4 Two-dimensional Case

In the two-dimensional case (3.47) becomes

$$\int_\Omega f_1(x_1)f_2(x_2)dx_1dx_2 = \oint_C F_1(x_1)f_2(x_2)dx_2, \tag{3.52}$$

where $F_1(t) = \int_{-\infty}^{t} f_1(x)dx$ is an antiderivative of $f_1(x)$. In our case, functions $f_1(x)$ and $f_2(x)$ are B-spline functions and their anti-derivatives can be computed analytically [17].

We represent the domain boundary $\partial\Omega$ by a parametric spline [9] of degree n_c. In 2-D for $t \in \mathbb{R}$ it is defined by:

$$u_1(t) = \sum_{i=0}^{N_c-1} u_1(i)\beta_{i,1}^{n_c}(t), \;\; u_2(t) = \sum_{i=0}^{N_c-1} u_2(i)\beta_{i,1}^{n_c}(t). \tag{3.53}$$

Using (3.52) the integral along the parametric spline (3.53) within the interval $t \in [t', t'']$ bounded by a grid cell (see Fig. 3.8) is computed as:

$$I = \int_{t'}^{t''} F_1[u_1(t)]f_2[u_2(t)]\frac{du_2}{dt}(t)dt. \tag{3.54}$$

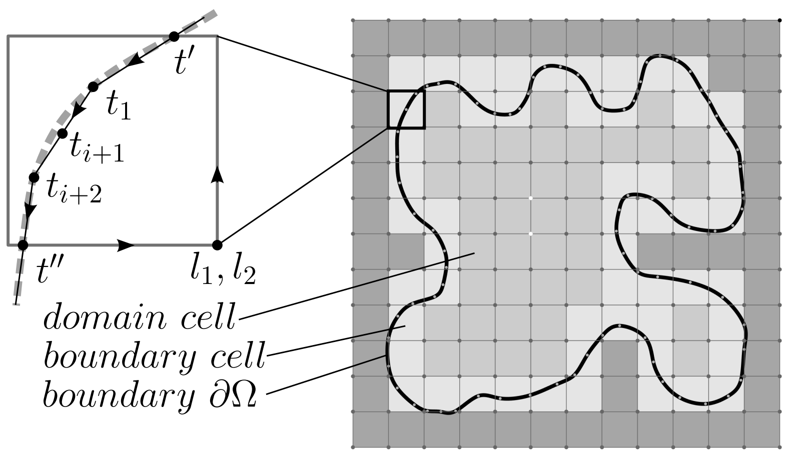


Figure 3.8: Domain Ω bounded by a parametric contour $(u_1(t), u_2(t))$, regular grid of tensor-product cubic B-spline basis and cell classification.

Expression (3.54) can be evaluated efficiently using one-dimensional standard quadratures [60]. When the parametric spline is linear ($n_c = 1$, cf. Fig. 3.12), integration is straightforward because $\frac{du_2}{dt}(t)$ is constant along each segment and finding intersections is trivial. We approximated function $F_1(t)$ in (3.54) using B-spline expansions resulting in high accuracy at look-up table computational complexity.

Knowing coordinates $(x_1(k), x_2(k))$ for $t_0, t_1, ..., t_k, ...$ we have to determine intersection points (x_1', x_2'), (x_1'', x_2'') with region boundary where function under integral is non-zero. Then for (x_1', x_2'), (x_1'', x_2'') we have to determine t' and t''. This is an inverse interpolation problem when knowing the function value we have to find it's argument. Fig. 3.9 clarifies the problem.

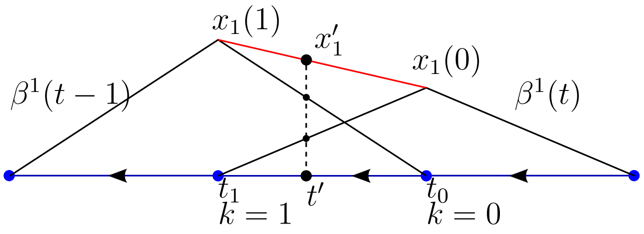


Figure 3.9: B-spline interpolation of the contour.

From Fig. 3.9 we can write that

$$x_1' = x_1(0)\beta^1(t') + x_1(1)\beta^1(t'-1), \tag{3.55}$$

$$x_1' = x_1(0)(t_1 - t') + x_1(1)(t' - t_0), \tag{3.56}$$

$$t' = \frac{x_1(1)t_0 - x_1(0)t_1 + x_1'}{x_1(1) - x_1(0)}. \tag{3.57}$$

And the derivative

$$\frac{dx_1'(t)}{dt} = x_1(1) - x_1(0). \tag{3.58}$$

3.4.5 Three-dimensional Case

For a single-component vector field $\mathbf{g}(\mathbf{x}) = (g_1(\mathbf{x}), 0, 0)$, continuously differentiable on a neighborhood of volume $V \subset \mathbb{R}^3$ bounded by a piecewise smooth boundary S, the Divergence Theorem reduces to

$$\int_V \frac{\partial g_1}{\partial x_1} dV = \oint_S g_1 n_1 dS, \tag{3.59}$$

where n_1 is a component of the outward unit normal field of the closed boundary surface S. By substituting g_1 with its antiderivative we get

$$\int_V g_1 dV = \oint_S \left(\int g_1 dx_1 \right) n_1 dS. \tag{3.60}$$

When the scalar field g_1 is constructed via tensor product (separable), i.e. $g_1 = f_1(x_1)f_2(x_2)f_3(x_3)$, expression (3.60) becomes

$$\int_V g_1(\mathbf{x}) dV = \oint_S F_1(x_1)f_2(x_2)f_3(x_3)n_1 dS, \tag{3.61}$$

where $F_1(x) = \int f_1(x)dx$ is the antiderivative of $f_1(x)$. Thus, rather than integrating the field g_1 over the volume V, it is integrated over its boundary S.

Using the Divergence Theorem the integration over the domain Ω can be reduced to the integration over multiple $(d-1)-$ dimensional boundary segments $\delta\Omega_i \subset \delta\Omega$:

$$\int_\Omega g_1(\mathbf{x}) d\mathbf{x} = \sum_{i=1}^{m} \int_{\delta\Omega_i} F_1(x_1)f_2(x_2)f_3(x_3)n_{1i} dS, \tag{3.62}$$

where m is the number of segments.

According to the described integration strategy, boundary tensors $\overline{\mathscr{W}}_{\mathbf{klm}}$, $\overline{\mathscr{F}}_{\mathbf{klm}}$ and $\overline{\mathscr{R}}_{\mathbf{kl}}$ are computed as follows:

$$\begin{aligned} \overline{\mathscr{W}}_{\mathbf{klm}} = & \int_S F_1(x_1)f_1(x_2)f_1(x_3)n_1 dS + \\ & \int_S F_2(x_1)f_2(x_2)f_1(x_3)n_1 dS + \\ & \int_S F_2(x_1)f_1(x_2)f_2(x_3)n_1 dS, \end{aligned} \tag{3.63}$$

$$\overline{\mathscr{F}}_{\mathbf{klm}} = \int_S F_2(x_1)f_1(x_2)f_1(x_3)n_1 dS, \tag{3.64}$$

$$\overline{\mathscr{R}}_{\mathbf{kl}} = \int_S F_3(x_1)f_3(x_2)f_3(x_3)n_1 dS, \tag{3.65}$$

where

$$F_1(x) = \int_{-\infty}^{x} \frac{d}{dt}\{\beta_k^{n_b}(t)\}\frac{d}{dt}\{\beta_l^{n_b}(t)\}\beta_m^{n_p}(t)dt, \tag{3.66}$$

$$F_2(x) = \int_{-\infty}^{x} \beta_k^{n_b}(t)\beta_l^{n_b}(t)\beta_m^{n_p}(t)dt, \tag{3.67}$$

$$F_3(x) = \int_{-\infty}^{x} \beta_k^{n_s}(t)\beta_l^{n_b}(t)dt, \tag{3.68}$$

$$f_1(x) = \beta_k^{n_b}(x)\beta_l^{n_b}(x)\beta_m^{n_p}(x), \tag{3.69}$$

$$f_2(x) = \frac{d}{dx}\{\beta_k^{n_b}(x)\}\frac{d}{dx}\{\beta_l^{n_b}(x)\}\beta_m^{n_p}(x), \tag{3.70}$$

$$f_3(x) = \beta_k^{n_s}(x)\beta_l^{n_b}(x). \tag{3.71}$$

We found that antiderivatives $F_1(x), F_2(x), F_3(x)$ are smooth functions and can be accurately approximated using uniform B-spline interpolation. This allows the replacement of an explicit computation of the antiderivatives with inexpensive evaluation of a spline and leads to a significantly faster integration process.

Finally, the computation of $\overline{\mathscr{H}}_{\mathbf{kl}}$ is done as:

$$\overline{\mathscr{H}}_{\mathbf{kl}} = \int_{\partial\Omega} f_h(x_1) f_h(x_2) f_h(x_3) d\partial\Omega, \tag{3.72}$$

where

$$f_h(x) = \beta_k^{n_b}(x)\beta_l^{n_b}(x). \tag{3.73}$$

3.4.6 Boundary Surface Processing

The proposed integration method does not require building a three-dimensional mesh for integration, while at the same time, it allows integration over complicated, non-convex domains.

Fig. 3.10 presents the main steps of the geometry processing algorithm for the three-dimensional case. Fig. 3.12 visualizes some of the steps, where boundary surface intersections with the regular grid cells are determined (Fig. 3.12, (a)), cells are classified into domain and boundary cells (Fig. 3.12, (b)), complex boundary segments for each cell are triangulated (Fig. 3.12, (c)), and two-dimensional quadratures are constructed on the triangles (Fig. 3.12, (d)). More efficient adaptive quadrature schemes could be used directly on the polygons avoiding any triangulation.

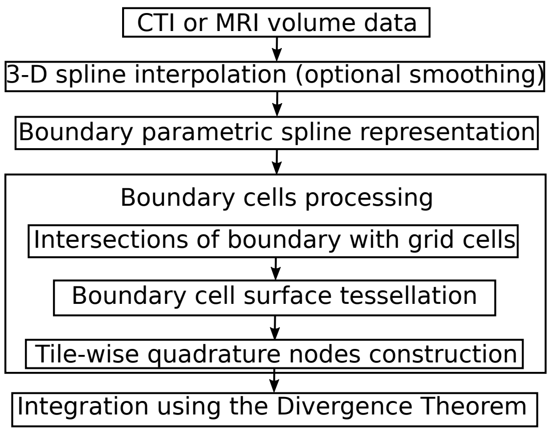

CTI or MRI volume data
3-D spline interpolation (optional smoothing)
Boundary parametric spline representation
Boundary cells processing
Intersections of boundary with grid cells
Boundary cell surface tessellation
Tile-wise quadrature nodes construction
Integration using the Divergence Theorem

Figure 3.10: Geometry processing algorithm outline.

Domain Surface Representation

Different surface representations are possible, including 1) implicit form $F(\mathbf{x}) = 0$; 2) parametric form $\mathbf{x} = F(u, v)$; 3) polygon mesh form; 4) R-functions-based (Rvachev functions) [61].

In this work we consider a parametric surface representation. For example, in the medical applications, CT and MRI scans can be used to construct a closed genus-zero domain surface. The original volumetric data is smoothed with a low-pass filter and interpolated using regular B-splines with the desired grid step, defining the quality of the surface approximation. Then an iso-surface is extracted and transformed into a parametric spline surface

$$x_1(\mathbf{u}) = \sum_{i_1=0}^{N_1-1} \sum_{i_2=0}^{N_2-1} x_{1 i_1 i_2} \beta^{n_c}(u_1 - i_1)\beta^{n_c}(u_2 - i_2), \tag{3.74}$$

$$x_2(\mathbf{u}) = \sum_{i_1=0}^{N_1-1} \sum_{i_2=0}^{N_2-1} x_{2 i_1 i_2} \beta^{n_c}(u_1 - i_1)\beta^{n_c}(u_2 - i_2), \tag{3.75}$$

$$x_3(\mathbf{u}) = \sum_{i_1=0}^{N_1-1} \sum_{i_2=0}^{N_2-1} x_{3 i_1 i_2} \beta^{n_c}(u_1 - i_1) \beta^{n_c}(u_2 - i_2), \tag{3.76}$$

where $\mathbf{u} \in \mathbb{R}^2$ is the parameter vector, and $\mathbf{i} \in \mathbb{Z}^2$ is the parametric grid node index. Here, for simplicity reasons, we use linear B-spline basis $n_c = 1$. To ensure closeness of the spline surface, the periodic boundary condition is used.

Patch-Cell Intersections

The algorithm iterates through the parametric surface patches. Each patch is split into two triangles to simplify patch-cell intersection algorithms. For each triangle, a three-dimensional bounding box is determined that could contain neighboring cells that might intersect this triangle. This reduces the amount of line-plane intersection procedure calls.

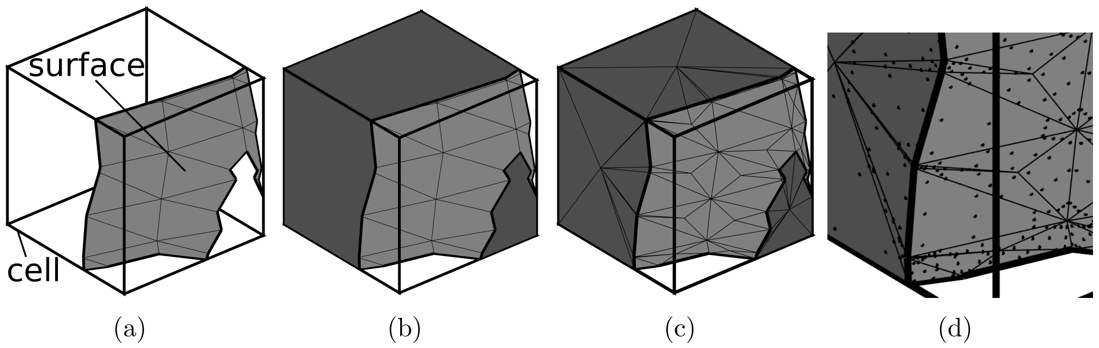


Figure 3.11: The result of a boundary surface-cell intersection (a), a boundary cell (grid cell cut by the boundary surface) (b), additional triangulation (c), quadrature nodes (d). Note the concave geometry of the surface.

For a patch and each neighboring cell face, a standard line-plane intersection detector is applied that returns coordinates of intersection points. The cases where patch vertices are inside a cell and where an intersection polygon has edges inside a cell are handled carefully, duplicated, and irrelevant intersection points are deleted. The result of a patch-cell intersection is shown in Fig. 3.11, (a).

Polygons and Quadratures

In order to integrate over the surface of a three-dimensional volume that is a result of boundary surface-cell intersection, the polygons that compose this surface have to be determined. The surface of intersection is composed of 1) the subsurface that is a result of a boundary surface-cell intersection (Fig. 3.11, (a), light gray), 2) cell faces trimmed by intersection segments (Fig. 3.11, (b), dark gray), 3) full cell faces.

A subsurface usually comprises several polygons defined by intersection line segments and in some cases involving cell edges; these polygons lie mostly in different planes. For each cell, line segments that belong to the same parametric surface patch compose a polygon. Connections between those segments are determined with the knowledge about the information about node sharing between adjacent line segments. The obtained polygons are split into triangles (Fig. 3.11, (c)). For each obtained triangle a two-dimensional Gauss quadrature is constructed (Fig. 3.12, (d) and (Fig. 3.11, (d)).

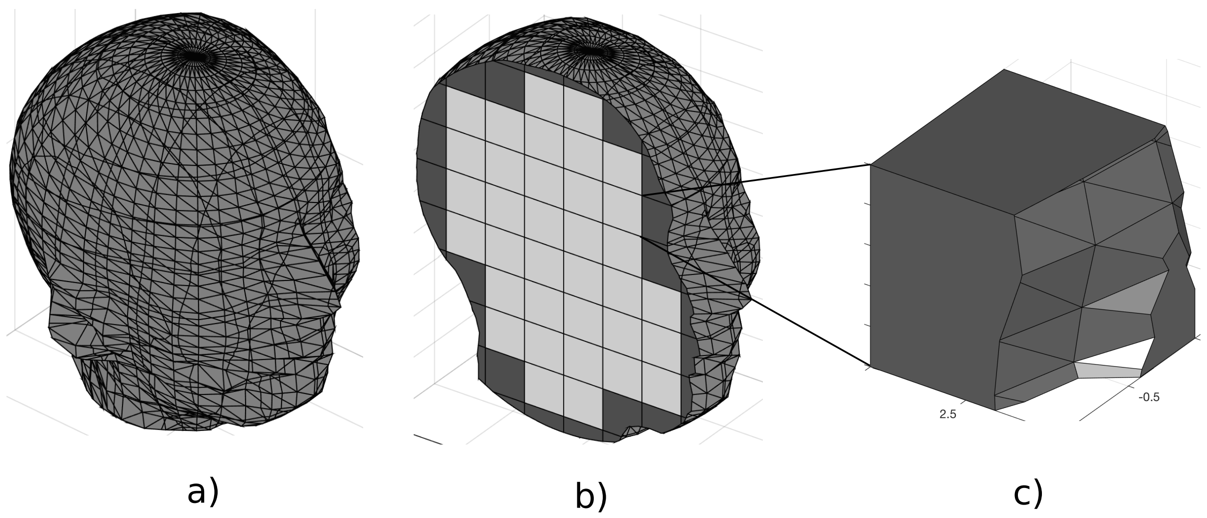


Figure 3.12: Domain polygonal surface dissected by three-dimansional grid cells (a), domain cut shows domain cells (light gray), and boundary cells (dark gray) that are cells intersected by domain surface (b), a boundary cell with domain surface side triangulated (c).

3.5 Application of Boundary Conditions

The Tensor B-spline method allows an application of different boundary conditions. Consider the boundary condition of the form

$$-\mathbf{n}\cdot(D(\mathbf{x}))\nabla\varphi(\mathbf{x}) = \gamma(\varphi(\mathbf{x}) - g_D(\mathbf{x})) + g_N(\mathbf{x}),\ \mathbf{x}\in\partial\Omega. \tag{3.77}$$

Using (3.77) the variational formulation (3.9) becomes

$$\begin{aligned}\int_\Omega D(\mathbf{x})\nabla\hat{\varphi}(\mathbf{x})\cdot\nabla\psi_l(\mathbf{x})d\mathbf{x} + \int_\Omega \mu_a(\mathbf{x})\hat{\varphi}(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x} +\\ \int_{\partial\Omega}\gamma\hat{\varphi}(\mathbf{x})\psi_l(\mathbf{x})ds + \int_{\partial\Omega}(\gamma g_D(\mathbf{x}) - g_N(\mathbf{x}))\psi_l(\mathbf{x})ds = \int_\Omega q(\mathbf{x})\psi_l(\mathbf{x})d\mathbf{x}.\end{aligned} \tag{3.78}$$

3.5.1 Dirichlet BC

The Dirichlet BC is obtained under the condition $g_N(\mathbf{x}) = 0$ and $\gamma\to+\infty$:

$$\varphi(\mathbf{x}) = g_D(\mathbf{x}) - \frac{1}{\gamma}\mathbf{n}\cdot(D(\mathbf{x}))\nabla\varphi(\mathbf{x}),\ \mathbf{x}\in\partial\Omega, \tag{3.79}$$

$$\gamma\to+\infty\ :\ \varphi(\mathbf{x}) = g_D(\mathbf{x}),\ \mathbf{x}\in\partial\Omega. \tag{3.80}$$

This approach is used for Dirichlet BC approximation in the boundary penalty method [62, 63], where

$$\mathbf{n}\cdot(D(\mathbf{x}))\nabla\varphi(\mathbf{x}) = \varepsilon^{-1}(\varphi(\mathbf{x}) - g_D(\mathbf{x})), \mathbf{x}\in\partial\Omega, \tag{3.81}$$

and $\varepsilon > 0$ is a penalty factor. It can be shown [62] that for $\varepsilon\le h^{k+1}$ the solution of PDE converges and the error is dominated by the interpolation factor.

3.5.2 Neumann BC

The inhomogeneous Neumann BC can be obtained from (3.77) by setting $\gamma = 0$:

$$-\mathbf{n}\cdot(D(\mathbf{x}))\nabla\varphi(\mathbf{x}) = g_N(\mathbf{x}),\ \mathbf{x}\in\partial\Omega. \tag{3.82}$$

Setting the $g_N(\mathbf{x}) = 0$ in (3.82) gives the homogeneous Neumann BC :

$$-\mathbf{n}\cdot(D(\mathbf{x}))\nabla\varphi(\mathbf{x}) = 0,\ \mathbf{x}\in\partial\Omega. \tag{3.83}$$

3.5.3 Robin BC

The Robin BC can be easily obtained from (3.77) by setting $\varphi(\mathbf{x}) = 0$:

$$-\mathbf{n} \cdot (D(\mathbf{x}))\nabla\varphi(\mathbf{x}) = \gamma g_D(\mathbf{x}) + g_N(\mathbf{x}),\ \mathbf{x} \in \partial\Omega. \tag{3.84}$$

3.5.4 Mixed BC

In the case of Mixed BC different boundary conditions are satisfied on disjoint parts of the boundary $\partial\Omega$, for example for $\partial\Omega_1 \subset \partial\Omega$ and $\partial\Omega_2 \subset \partial\Omega$

$$\varphi(\mathbf{x}) = \varphi_0,\ \mathbf{x} \in \partial\Omega_1, \tag{3.85}$$

$$\nabla\varphi(\mathbf{x}) \cdot \mathbf{n} = 0,\ \mathbf{x} \in \partial\Omega_2. \tag{3.86}$$

3.6 Solution of Time-dependent PDEs

The solution of a time-dependent problem is usually obtained using a finite difference approximation in time. Examples of finite difference methods are forward and backward Euler methods [24], Crank–Nicolson method [64].

3.6.1 Solution of a Parabolic PDE

Consider an application of the Crank–Nicolson method to the time-dependent Diffusion PDE with the Robin boundary condition

$$\frac{\partial\varphi(\mathbf{x},t)}{\partial t} = \nabla \cdot [D(\mathbf{x})\nabla\varphi(\mathbf{x},t)],\ \mathbf{x} \in \Omega, \tag{3.87}$$

$$2D(\mathbf{x})(\nabla\varphi(\mathbf{x}) \cdot \mathbf{n}) + \varphi(\mathbf{x}) = 0,\ \mathbf{x} \in \partial\Omega, \tag{3.88}$$

where $\varphi(\mathbf{x},t)$ is the density of the diffusing material at location $\mathbf{x}$ and time t, and $D(\mathbf{x})$ is the diffusion coefficient at location $\mathbf{x}$.

The Crank–Nicolson method is an implicit, numerically stable, second-order method in time. Consider the unknown function discretized in time $\varphi(\mathbf{x}, k\Delta t) = \varphi^k(\mathbf{x})$, where $k = 0, 1, 2...$, and Δt is a time step. The PDE discretization in time is written as

$$\frac{\varphi^{k+1}(\mathbf{x}) - \varphi^k(\mathbf{x})}{\Delta t} = \frac{1}{2}\Big[\nabla \cdot [D(\mathbf{x})\nabla\varphi^{k+1}(\mathbf{x})] + \nabla \cdot [D(\mathbf{x})\nabla\varphi^k(\mathbf{x})]\Big], \tag{3.89}$$

and results in

$$-\frac{1}{2}\nabla \cdot [D(\mathbf{x})\nabla\varphi^{k+1}(\mathbf{x})] + \frac{1}{\Delta t}\varphi^{k+1}(\mathbf{x}) = \frac{1}{\Delta t}\varphi^k(\mathbf{x}) + \nabla \cdot [D(\mathbf{x})\nabla\varphi^k(\mathbf{x})], \tag{3.90}$$

which is a linear second-order elliptic PDE in respect to the unknown $\varphi^{k+1}(\mathbf{x})$ at time $(k+1)\Delta t$. This supports the applicability of our Tensor B-spline method to solve parabolic PDEs as (3.87).

3.6.2 Solution of a Hyperbolic PDE

Consider a wave equation

$$\frac{\partial^2\varphi(\mathbf{x},t)}{\partial t^2} = c^2\nabla^2\varphi(\mathbf{x},t),\ \mathbf{x} \in \Omega, \tag{3.91}$$

$$2D(\mathbf{x})(\nabla\varphi(\mathbf{x}) \cdot \mathbf{n}) + \varphi(\mathbf{x}) = 0,\ \mathbf{x} \in \partial\Omega, \tag{3.92}$$

where $\varphi(\mathbf{x},t)$ is a pressure field, c is the speed of the wave propagation in the media, and $f(\mathbf{x},t)$ is a force term.

A discretization of (3.91) is written as

$$\frac{\varphi^{k+1}(\mathbf{x}) - 2\varphi^{k}(\mathbf{x}) + \varphi^{k-1}(\mathbf{x})}{\Delta t^2} = \nabla \cdot [D(\mathbf{x})\nabla\varphi^{k+1}(\mathbf{x})], \tag{3.93}$$

and results in

$$-\nabla \cdot [D(\mathbf{x})\nabla\varphi^{k+1}(\mathbf{x})] + \frac{1}{\Delta t^2}\varphi^{k+1}(\mathbf{x}) = \frac{2}{\Delta t^2}\varphi^{k}(\mathbf{x}) - \frac{1}{\Delta t^2}\varphi^{k-1}(\mathbf{x}), \tag{3.94}$$

which is a linear second-order elliptic PDE in respect to the unknown $\varphi^{k+1}(\mathbf{x})$ at time $(k+1)\Delta t$. Thus, the wave equation is also amenable to solving by our method.

3.7 Solution of Convection-Diffusion PDEs

As an example of a convection-diffusion PDE we consider the Navier-Stokes Equation for incompressible and homogeneous fluid.

Given the initial values of the velocity and pressure, the state of the fluid over time can be described by the Navier-Stokes equation for incompressible flow:

$$\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} = -(\mathbf{u}(\mathbf{x},t)\cdot\nabla)\mathbf{u}(\mathbf{x},t) - \frac{1}{\rho}\nabla p(\mathbf{x},t) + \nu\nabla^2\mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x}), \tag{3.95}$$

$$\nabla \cdot \mathbf{u}(\mathbf{x},t) = 0, \tag{3.96}$$

where $\mathbf{u}(\mathbf{x},t)$ is the vector velocity field, $p(\mathbf{x},t)$ is the scalar pressure field, ρ is the fluid density, ν is the kinematic viscosity and $\mathbf{f}$ represents external forces that act on the fluid.

One of the methods [65] of the solution of (3.95) is an application of Helmholtz-Hodge Decomposition that states that vector field $\mathbf{w}(\mathbf{x})$ can be uniquely decomposed into the form:

$$\mathbf{w}(\mathbf{x}) = \mathbf{u}(\mathbf{x}) + \nabla p(\mathbf{x}), \tag{3.97}$$

where the field $\mathbf{u}(\mathbf{x})$ has zero divergence ($\nabla\cdot\mathbf{u}(\mathbf{x}) = 0$) and $p(\mathbf{x})$ is scalar field.

We define an operator P which projects vector field $\mathbf{w}(\mathbf{x},t)$ onto its divergence free part

$$\mathbf{u}(\mathbf{x},t) = P\mathbf{w}(\mathbf{x},t) = \mathbf{w}(\mathbf{x},t) - \nabla p(\mathbf{x},t). \tag{3.98}$$

In order to find $p(\mathbf{x},t)$ we multiply (3.97) by ∇ and get

$$\nabla\cdot\mathbf{w}(\mathbf{x},t) = \nabla\cdot\mathbf{u}(\mathbf{x},t) + \nabla\cdot\nabla p(\mathbf{x},t), \tag{3.99}$$

$$\nabla^2 p(\mathbf{x},t) = \nabla\cdot\mathbf{w}(\mathbf{x},t). \tag{3.100}$$

The equation (3.100) is the Poisson equation that is solved for $p(\mathbf{x},t)$.

We apply the projection operator P to (3.95):

$$P\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} = P\left(-(\mathbf{u}(\mathbf{x},t)\cdot\nabla)\mathbf{u}(\mathbf{x},t) - \frac{1}{\rho}\nabla p(\mathbf{x},t) + \nu\nabla^2\mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x})\right). \tag{3.101}$$

Because $\mathbf{u}(\mathbf{x},t)$ is derivative-free $P\frac{\partial\mathbf{u}}{\partial t}(\mathbf{x},t) = \frac{\partial\mathbf{u}}{\partial t}(\mathbf{x},t)$, also $P\nabla p(\mathbf{x},t) = 0$. Finally we obtain:

$$\frac{\partial \mathbf{u}(\mathbf{x},t)}{\partial t} = \mathscr{P}\left(-(\mathbf{u}(\mathbf{x},t)\cdot\nabla)\mathbf{u}(\mathbf{x},t) + \nu\nabla^2\mathbf{u}(\mathbf{x},t) + \mathbf{f}(\mathbf{x})\right). \tag{3.102}$$

Given $\mathbf{u}(\mathbf{x},t) = \mathbf{w}$ at time t the solving algorithms consists of the following steps:

1. Computation of advection $-(\mathbf{w} \cdot \nabla)\mathbf{w}$;
2. Computation of diffusion $\nu \nabla^2 \mathbf{w}$;
3. Application of forces $\mathbf{f}$;
4. Projecting of $\mathbf{w}$ to divergence-free $\mathbf{u} = P\mathbf{w}$ in two steps:
   (a) Find p from $\nabla^2 p = \nabla \cdot \mathbf{w}$;
   (b) Find $\mathbf{u} = \mathbf{w} - \nabla p$. Obtained $\mathbf{u} = \mathbf{u}(\mathbf{x}, t + \Delta t)$. Return to step 1 with $\mathbf{w} := \mathbf{u}$.

Advection is the process by wich a fluid's velocity transports itself and other quantities in the fluid. To compute advection the method of characteristics can be used. It states that to update a quantity q (that could be velocity, density, or any quantity carried by the fluid) the following equation is used:

$$q(\mathbf{x}, t + \Delta t) = q(\mathbf{x} - \mathbf{u}(\mathbf{x}, t)\Delta t, t). \tag{3.103}$$

It requires to compute the value of the field q at a new position $\mathbf{x} - \mathbf{u}(\mathbf{x}, t)\Delta t$ that is accomplished by an interpolation procedure.

The viscous diffusion equation

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u} \tag{3.104}$$

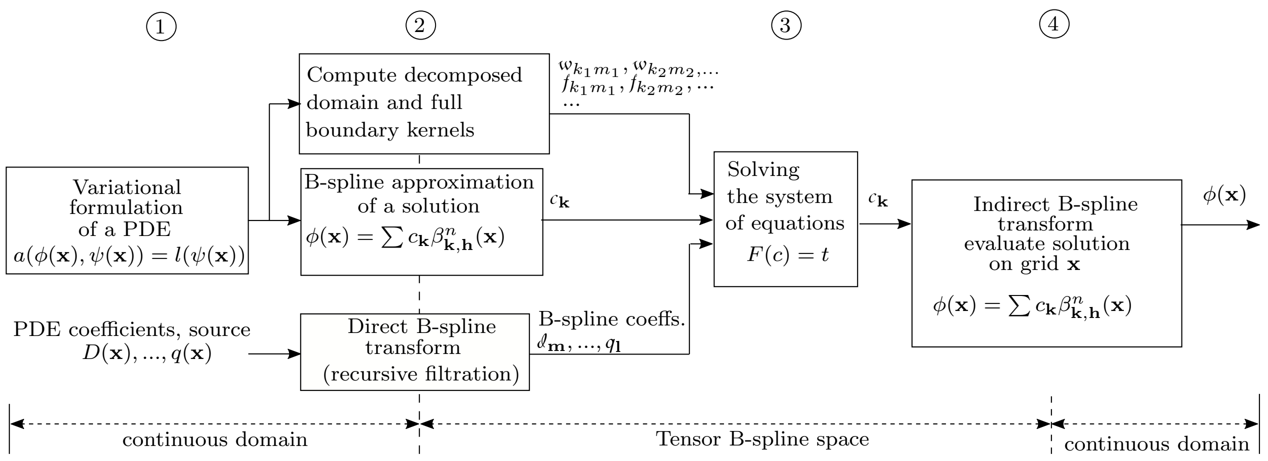
is solved in time as

$$-\nu \Delta t \nabla^2 \mathbf{u}(\mathbf{x}, t + \Delta t) + \mathbf{u}(\mathbf{x}, t + \Delta t) = \mathbf{u}(\mathbf{x}, t). \tag{3.105}$$

B-splines can be used pervasively at each step of the solving process. Computation of the advection can be performed efficiently via B-spline interpolation. Solving of the diffusion equation (3.104) as well as the computation of the pressure field p can be done efficiently using the Tensor B-spline method. The divergence and gradient operations can be computed analytically using B-spline properties. Thus, our method is a well-applicable building block for solving convection-diffusion equations.

3.8 Method Summary

An overview of the Tensor B-spline numerical method for PDEs is given in Fig. 3.13. The method starts in the continuous domain with the variational formulation of a PDE, given coefficient and source functions ①. The continuous formulation is discretized with Tensor B-splines ②. At this stage, the domain and boundary kernels are computed and both the source and the coefficients are transformed into B-spline space via the direct B-spline transform. The obtained system of equations is solved using an approach that meets a desirable criterion (speed, memory-efficiency) ③. Different methods for solving the system of equations are described in the next chapter. The obtained coefficients of the solution are transformed back into the continuous domain by way of the indirect B-spline transform ④.



1
2
3
4
Variational formulation of a PDE $a(\phi(\mathbf{x}),\psi(\mathbf{x}))=l(\psi(\mathbf{x}))$
Compute decomposed domain and full boundary kernels
$\omega_{k_1m_1},\omega_{k_2m_2},\ldots$ $f_{k_1m_1},f_{k_2m_2},\ldots$ ...
B-spline approximation of a solution $\phi(\mathbf{x})=\sum c_{\mathbf{k}}\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x})$
$c_{\mathbf{k}}$
PDE coefficients, source $D(\mathbf{x}),\ldots,q(\mathbf{x})$
Direct B-spline transform (recursive filtration)
B-spline coeffs. $d_{\mathbf{m}},\ldots,q_{\mathbf{l}}$
Solving the system of equations $F(c)=t$
$c_{\mathbf{k}}$
Indirect B-spline transform evaluate solution on grid $\mathbf{x}$ $\phi(\mathbf{x})=\sum c_{\mathbf{k}}\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x})$
$\phi(\mathbf{x})$
continuous domain
Tensor B-spline space
continuous domain

Figure 3.13: An overview of the Tensor B-spline numerical method for PDEs.

3.8.1 Computation of B-spline Coefficients

For the B-spline approximations of PDE coefficients and source

$$D(\mathbf{x}) \approx \sum_{\mathbf{m}} \mathcal{D}_{\mathbf{m}} \beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}), \tag{3.106}$$

$$\mu_a(\mathbf{x}) \approx \sum_{\mathbf{m}} \mathcal{M}_{\mathbf{m}} \beta^{n_p}_{\mathbf{m},\mathbf{h}}(\mathbf{x}), \tag{3.107}$$

$$q(\mathbf{x}) \approx \sum_{\mathbf{j}} \mathcal{Q}_{\mathbf{j}} \beta^{n_s}_{\mathbf{j},\mathbf{h}}(\mathbf{x}), \tag{3.108}$$

the B-splines coefficients $\mathcal{D}_{\mathbf{m}}$, $\mathcal{M}_{\mathbf{m}}$, and $\mathcal{Q}_{\mathbf{j}}$ have to be computed. This can be done either according to the interpolation(2.59) or approximation (2.64), (minimization of $\mathbb{L}^2$ norm) criteria as was described in Chapter 2. We propose to perform both tasks via separable digital filtering.

As an example, we consider the computation of coefficients $\mathcal{D}_{\mathbf{m}}$ using B-spline interpolation for the three-dimensional case. We define a regular tensor-product grid $g_{\mathbf{m}} \in \mathbb{R}$. It is important that the grid $g_{\mathbf{m}}$ slightly extends beyond the domain of interest Ω.

For an analytically defined coefficient $D(\mathbf{x}) = f_D(\mathbf{x})$, $\mathbf{x} \in \mathbb{R}^3$, the computation of B-spline coefficients consists of (see Fig. 3.14): 1) evaluation of $f_D(\mathbf{x})$ at grid points $g_{\mathbf{m}}$, 2) an application of a boundary condition, 3) successive filtering along m_1, m_2, and m_3.

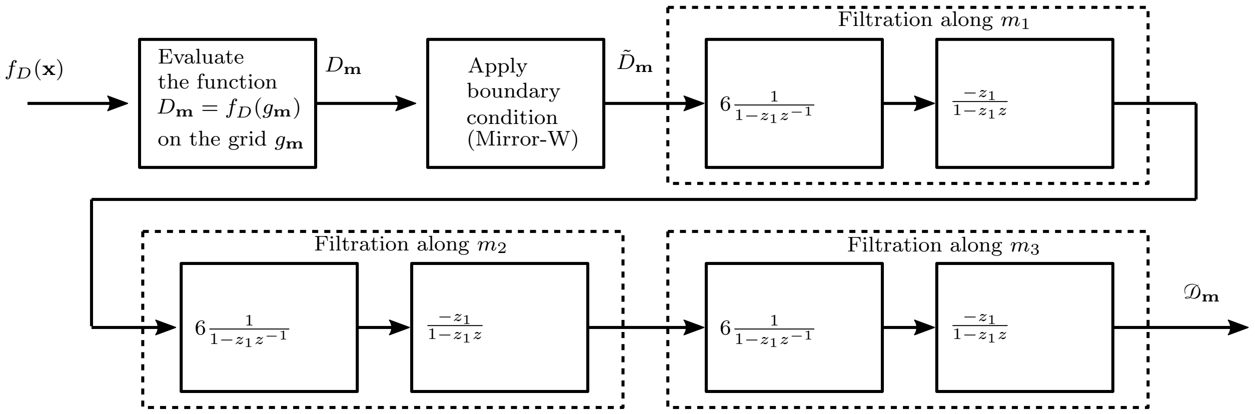


Figure 3.14: An example processing chian for a B-spline coefficients computation.

When the coefficients of PDE are represented as a set of samples $\{D_{\mathbf{k}} \in \mathbb{R}\}_{\mathbf{k}=\mathbf{0}}^{\mathbf{K}-\mathbf{1}}$ on a grid $g_{\mathbf{k}}$, a simple way of representing the coefficient on another grid $g_{\mathbf{m}}$ is as follows:

1. the B-spline representation of the $D_{\mathbf{k}}$ is constructed $D_s(\mathbf{x}) = \sum_{\mathbf{k}} c_{\mathbf{k}} \beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x})$, $c_{\mathbf{k}} = (b^n_1(\mathbf{k}))^{-1} * D_{\mathbf{k}}$;
2. the spline $D_s(\mathbf{x})$ is evaluated at the nodes of grid $g_{\mathbf{m}}$: $D_{\mathbf{m}} = D_s(g_{\mathbf{m}})$;
3. B-spline boundary condition is applied and successive filtering along each dimension is applied (Fig. 3.14).

Alternatively, a more optimal representation can be achieved using $\mathbb{L}^2$-projection.

The described approach allows efficient computation of B-spline coefficients using successive 1-D filtering of input data along dimensions. It requires less operations and memory space in comparison to the classical approach, where a sparse matrix equations are solved [11].

3.8.2 Evaluation of the Solution

The computed B-spline coefficients $C^{\mathbf{k}}$ determine a continuous spline form of the PDE solution:

$$\hat{\varphi}(\mathbf{x}) = \sum_{\mathbf{k}} C^{\mathbf{k}} \beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x}), \ \mathbf{x} \in \Omega. \tag{3.109}$$

The evaluation of this function at integer nodes can be performed with high efficiency via indirect B-spline digital filtering [7]. In a more general case, of arbitrarily-distributed points, the evaluation of (3.109) can be performed with low computational complexity thanks to the separability and finite support properties of tensor B-splines. Moreover, the differentiation B-spline property (2.55) enables efficient evaluation of the derivatives of $\hat{\varphi}(\mathbf{x})$, for example

$$\frac{\partial \hat{\varphi}}{\partial x_i}(\mathbf{x}) = \sum_{\mathbf{k}} C^{\mathbf{k}} \frac{\partial \beta^n_{\mathbf{k},\mathbf{h}}}{\partial x_i}(\mathbf{x}). \tag{3.110}$$

In the reconstruction problems like ODT, the solution $C^{\mathbf{k}}$ could be used directly in the iterative reconstruction algorithm.

3.9 Discussion

In this chapter, the theory and principles of the Tensor B-spline numerical method for solving PDEs were presented. The method can be used to solve a range of PDEs with different boundary conditions. The method employs B-spline representations of the PDE terms and considers the PDE discrete formulation from the computational tensor algebra viewpoint. The outcome is a decomposed tensor structure of the discrete formulation within the domain. This tensor decomposition consists of separable shift-invariant tensor B-spline kernels that are convolved with B-spline coefficients representing all PDE terms. This allows efficient integration and matrix-free computations of the system operator. The B-spline kernels that intersect the domain boundary are non-separable, thought the ratio of such kernels is small. While separable domain kernels are integrated only once and then are reused, boundary kernels are unique and have to be stored. An efficient, novel integration method is proposed for integration of boundary kernels.

3.9.1 B-splines

B-splines link the continuous signal domain with a discrete problem representation while preserving analytical precision even if discrete algorithms are applied [7]. Excellent approximation properties of B-splines are complemented by their efficient processing algorithms. Simple linear and high-order approximations are possible, where cubic B-splines are of particular interest because of their minimal curvature property [7].

B-spline Representations via Digital Filtering

B-spline coefficients for the spline-based representations of the PDE coefficients and source term are obtained via digital filtering algorithms; this approach is more efficient than its classic matrix-based equivalent [11]. A comprehensive B-spline signal processing framework is integrated into a Tensor B-spline numerical method such that it allows to work with large-scale data in memory-efficient way. The solution of a PDE is obtained in the B-spline space, and then is transformed into the signal space with the desired smoothness and resolution. High-quality representations with limited resources are possible with cubic B-splines.

3.9.2 Tensor Structure

Tensor representations are more beneficial compared to flat-view matrix models in signal processing applications [8, 13] and in numerical methods for PDEs as was shown in this work. In the Tensor B-spline method, tensors allow the natural representation of the PDE formulation where a multidimensional structure is preserved, in comparison to matrices, where the initial spatial structure of the data is folded.

For example, the structure of the terms in the sum in Fig. 3.3 (a) corresponds to the Tucker factorization (2.69) (Fig. 2.15), and the whole expression corresponds to the case of block term decomposition (2.70). Remarkably, such a non-trivial decomposition results directly from the PDE formulation, without any tedious numerical analysis. It is difficult to reveal such structure from matrix-based representation, where the dimensions are merged and the structure information is lost.

Tensor Decompositions

Tensor decompositions are widely used in signal processing applications [66, 13, 14] where a multidimensional tensor is factorized in order to reveal its components based on some constraints (orthogonality, non-negativity, etc.). In this work, we showed the opposite case, where the application of tensors and B-splines introduces the decomposition initially (Fig. 3.2 and Fig. 3.3). Our approach eliminates any expensive analysis that might be needed if the problem were boiled down, for example, to the sparse matrix format, and one would like to optimize the computations afterward.

Tensor-inspired Computational Algorithms

An important aspect of tensors that comes from the computational tensor algebra [15, 16] is the identification of repeated patterns, local kernels and convolution procedures in order to derive efficient data processing algorithms. We showed that the discrete tensor formulation of the Ritzh-Galerkin formulation (3.19) preserves and highlights the geometric structure of (3.16), and thereby suggests multiple ways of operation order and helps to choose the optimal one.

Tensor-based Storage Format

When a discrete formulation benefits from sparsity, the sparse matrix and sparse matrix-vector multiplication are commonly used. In comparison to this standard approach, the Tensor B-spline method uses a kernel-based tensor array format in order to avoid the sparse matrix format overhead, where indices of the non-zero elements have to be stored and accessed. Due to the dense nature of small-size kernels, it allows more efficient memory access and optimal utilization of CPU vector units (SIMD). Potentially, a matrix-free nature of the tensor formulation gives more freedom for optimizations during algorithm implementations. At last, the decomposed tensor format requires less memory than the sparse matrix and the full-tensor format.

3.9.3 Domain Discretization

Any element-based numerical method relies on a specific domain discretization. This discretization is represented either by elements (in case of FEM) or by grid cells (for Tensor B-splines). A key aspect of many FEM-based approaches is the use of a domain-conforming mesh (which is often unstructured). In FEM, meshing automation is a major field of study, and finding suitable automation strategies is highly challenging in practice [35]. Meshless methods, on the other hand, require much less effort when it comes to discretizing the domain, but the process of numerical integration typically involves very high computational costs [44]. Contrary to these approaches, tensor B-splines on regular grids have the advantage of regularity and shift-invariance in the domain. For these methods, however, boundary integration is a challenging problem, as well as the use of Dirichlet BC on non-rectangular domains. In our papers [17, 18], as well as in this work, an efficient integration method was proposed, based on the Divergence Theorem. Interestingly, recent advantages in FEM also consider similar approaches, where an easy-to-build regular mesh is used inside the domain, and the conforming mesh is adjusted to the domain boundaries [35]. Due to intrinsic B-spline multi-resolution properties, however, local grid refinements – similar to such mesh adaptations in FEM – are also possible in Tensor B-spline approaches.

3.9.4 B-spline Boundaries and Efficient Integration

B-splines are well suited for the parametric representation of contours [9], and their properties allow for analytical integration [7] – these are the essential features we employed in the context of solving PDEs. An accurate parametric description of a domain boundary allows us to reduce grid density when compared to the more straightforward approach, where boundaries are approximated using a limited number of simple elements as in [54].

Integration over domains with a complex shaped boundary is one of the most challenging parts of numerical methods that do not use meshes. In meshless methods, for example, the integration slows down the computations significantly [44]. In order to simplify the integration process, FEM splits a domain into primitive geometrical shapes that are boundary-fitted such that the integration over them becomes trivial and boundary conditions apply naturally. However, FEM needs mesh generation, which is, again, the most challenging part in FEM. We found that as an alternative to using standard numeric quadrature schemes, the integration steps required for solving PDE can be conveniently and efficiently achieved by the application of the Divergence Theorem. We showed how the application of the Divergence Theorem and the use of B-spline properties lead to an efficient integration method on the boundary. This method reduces the dimensionality of integration and requires simple quadratures. Thus, our approach is more straightforward geometrically and simpler to implement than those relying on a mesh. Moreover, our algorithm can process non-convex volumes composed of non-convex irregular polygons in contrast to algorithm [67] that supports only convex irregular polygons.

3.9.5 Dirichlet BC

The application of the Dirichlet BC is challenging in methods with basis functions that do not conform to the domain boundary. As was proposed in [20], one can use a weighted basis that vanishes on the boundary. The application of the Dirichlet boundary condition may require the use of weighting functions. Rvachev R-functions are good candidates for that [61]. In this work, we showed that Tensor B-spline solvers can successfully make use of the boundary penalty method [63]. In this method, the Dirichlet BC is approximated with Robin BC weighted by some penalty factor.

3.9.6 Limitations

The efficiency of the Tensor B-spline method has a dependency on the percentage ratio of the number of domain kernels to the number of boundary kernels - the higher is the prevalence of the domain kernels, the higher is the efficiency. On contrary, with the growth of the relative number of non-separable boundary tensor kernels, the method benefits less from efficient separable domain computations, and requires more time for integration and more memory space for storage. This could happen when the domain discretization is coarse or the boundary has a very irregular non-convex shape. However, the memory space for boundary kernels can be reduced when these kernels are pre-convolved with PDE coefficients right after the integration stage and only such compressed tensors are stored.

Chapter 4

Tensor B-spline Numerical Solver Implementation and Evaluation

4.1 Main Contributions

This chapter presents Tensor B-spline numerical solver implementation, evaluation, and comparison with state-of-the-art FEM. Different computational strategies are analysed, implemented, evaluated, and compared. The Tensor B-spline method is applied to the forward problem of Optical Diffusion Tomography and compared with the FEM from an ODT reconstruction framework. We present a high-performance solution of a large-scale problem consisting of 0.8 billion nodes that was obtained in memory-efficient fashion on an high-performance workstation.

4.1.1 Computational Strategies

The use of the decomposed tensor structure inspires efficient computational techniques for the computation of the system operator. We describe a convolution-based and a so-called "on-the-fly" techniques. These techniques are potentially more efficient than the classic sparse matrix-vector multiplication (SpMV) approach.

4.1.2 Application to Optical Diffusion Tomography

The importance of the fast and accurate PDE numerical solver can be seen in the example of ODT image reconstruction. For biological tissue, where light scattering dominates absorption, light propagation is commonly modeled by the Diffusion PDE [1]. ODT image reconstruction is a challenging non-linear inverse problem and is typically solved using iterative numerical methods, e.g., Newton methods [1, 68] where each iteration step requires solving a number of forward problems corresponding to different light source locations and parameter distributions characterizing the optical properties of the tissue. Accuracy and computational performance of the forward problem solver are critical for the overall efficiency of ODT image reconstruction [69, 70], especially in challenging cases of large whole-body imaging datasets [71]. We apply the Tensor B-spline method to the forward problem of the ODT [17, 18], and show that the proposed approach outperforms standard FEM in state-of-the-art ODT framework.

4.1.3 Comparison with state-of-the-art FEM

In order to prove the efficiency of the Tensor B-spline solver, we have performed a detailed comparison with the state-of-the-art Finite Element Method. The comparison includes one-, two-, and three-dimensional cases, domains with regular and irregular boundaries, and is based on synthetic and real-world data.

4.1.4 Solution of Large-Scale Problems

We have proved the method's ability to solve problems on dense grids using hardware with limited resources. For that, we have solved the heat transfer problem on a domain constructed from a CT scan and consisting of 0.8 billions nodes.

4.2 Computational Strategies for System Operator Evaluation

Consider a system operator from an example in the previous chapter

$$F(\mathcal{C}) = \mathcal{C}^{\mathbf{k}}\mathcal{D}^{\mathbf{m}}\mathcal{W}_{\mathbf{klm}} + \mathcal{C}^{\mathbf{k}}\mathcal{M}^{\mathbf{m}}\mathcal{F}_{\mathbf{klm}} + \frac{1}{2}\mathcal{C}^{\mathbf{k}}\mathcal{H}_{\mathbf{kl}}. \tag{4.1}$$

The computation of $F(\mathcal{C})$ is crucial because it is a fundamental building block of iterative methods, and it defines the most time-consuming stage when solving large sparse linear systems.

The SpMV, tensor convolution and "on-the-fly" strategies are possible (Fig. 4.1). Here, we briefly discuss their properties, while an extensive evaluation and comparison will be provided in the next sections.

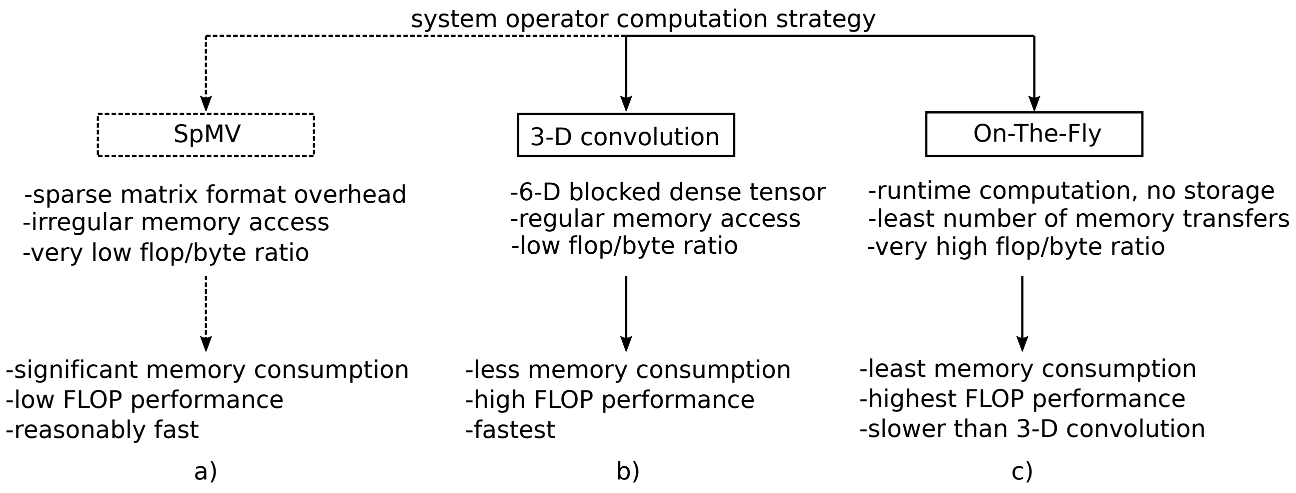

system operator computation strategy
SpMV
-sparse matrix format overhead
-irregular memory access
-very low flop/byte ratio
-significant memory consumption
-low FLOP performance
-reasonably fast
a)
3-D convolution
-6-D blocked dense tensor
-regular memory access
-low flop/byte ratio
-less memory consumption
-high FLOP performance
-fastest
b)
On-The-Fly
-runtime computation, no storage
-least number of memory transfers
-very high flop/byte ratio
-least memory consumption
-highest FLOP performance
-slower than 3-D convolution
c)

Figure 4.1: Computation strategies of the Tensor B-spline method.

4.2.1 Sparse Matrix Strategy

A classical sparse matrix-based approach can be applied via merging dimensions of the resulting tensor after all tensors were contracted and summed-up in (4.1). This operation flattens the multidimensional structure into a sparse two-dimensional matrix representation. Afterward, a standard sparse matrix-vector multiplication (SpMV) procedure can be used for a system operator evaluation.

We show that this approach appears to be the least efficient [17, 18]. This follows from the overhead due to the sparse matrix format, from non-regular memory access, from a very low flop-to-byte ratio [72, 73], and from problems concerning load imbalance [74] (Fig. 4.1, a). Since SpMV is a memory-bound procedure, performance optimizations do not overcome the issue of considerable memory consumption.

4.2.2 Tensor-based Strategies

Tensor Convolution

The use of a tensor structure permits the implementation of more efficient computing algorithms than SpMV. The first one uses a natural 6-D block tensor that reduces memory consumption in comparison to SpMV and provides regular memory access. The 6-D block tensor

$$\mathscr{P}_{\mathbf{kl}} = \mathscr{D}^{\mathbf{m}}\mathscr{W}_{\mathbf{klm}} + \mathscr{M}^{\mathbf{m}}\mathscr{F}_{\mathbf{klm}} + \frac{1}{2}\mathscr{H}_{\mathbf{kl}} \tag{4.2}$$

is assembled before the iterative solving. At each iteration, it is multiplied with $\mathscr{C}_{\mathbf{k}}$. This algorithm is the fastest [17, 18] but still is not feasible for large systems (Fig. 4.1, b).

On-the-Fly Computations

A kernel-based structure of the system operator inspires the efficient in-the-runtime computations of the expression

$$\mathscr{C}^{\mathbf{k}}\mathscr{D}^{\mathbf{m}}\mathscr{W}_{\mathbf{klm}} + \mathscr{C}^{\mathbf{k}}\mathscr{M}^{\mathbf{m}}\mathscr{F}_{\mathbf{klm}} + \frac{1}{2}\mathscr{C}^{\mathbf{k}}\mathscr{H}_{\mathbf{kl}} \tag{4.3}$$

when the tensor contractions are performed at each iteration without an explicit assembling of the large tensor like in (4.2). This approach results in a significant reduction in memory usage (Fig. 4.1, c). It has a very high flop-to-byte ratio and gains from the high floating point performance of CPUs and GPUs, as we are showing in the next sections.

4.3 Tensor B-spline Method application to the Optical Diffusion Tomography

The numerical solution of the forward problem PDEs in state-of-the-art image reconstruction frameworks like Toast++ [28], NIRFAST [29] and EIDORS [30], etc. [31] is based on the use of the Finite Element Method (FEM) [25]. So far, a B-spline-based forward model for ODT was proposed in [54] but was limited to linear splines and simple boundary approximation. WEB method was applied in physics to electromagnetism and in medicine for computing heat distribution in the human eye [75, 76].

Motivation in a high-order, mesh-free, efficient, and simple method leads to the use of B-splines functions together with tensors in a discrete PDEs formulation. The properties of splines and computational tensor algebra resulted in an elegant and straightforward derivation of remarkably efficient computational algorithms. In this work we apply the Tensor B-spline solver to the forward problem of the ODT.

4.4 Comparison with FEM: One-dimensional Case

4.4.1 B-spline and Lagrange FEM bases

A comparison of B-spline and Lagrange polynomial bases reveals the important properties and aspects of both the Tensor B-spline method and the FEM. We depict the univariate B-splines in contrast to the Lagrange polynomial functions used in FEM in Fig. 4.2 (a, b). This simple one-dimensional example shows some crucial differences between the bases.

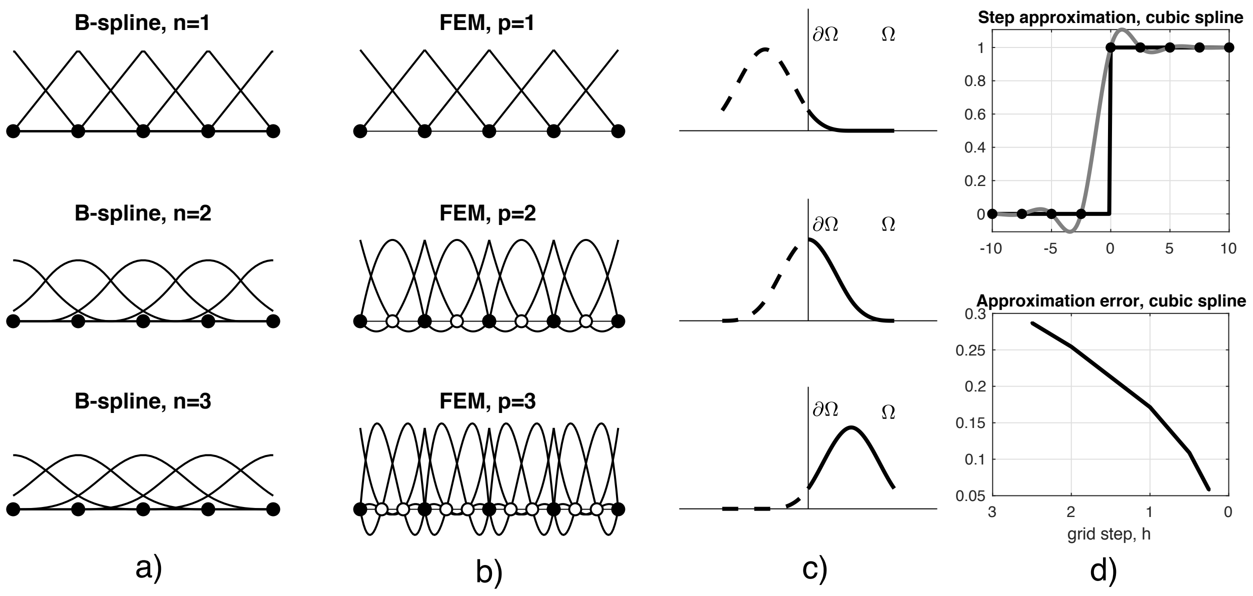


Figure 4.2: (a) univariate B-spline bases, (b) Lagrange polynomial FEM bases, (c) an intersection of a B-spine function and a domain boundary, (d) step function approximation with cubic B-spline and an error of approximation.

While linear FEM and B-splines coincide, an increasing basis order introduces additional nodes in FEM (white circles, Fig. 4.2 (b) and spreads the support of B-splines (Fig. 4.2 (a)). B-splines do not conform to the domain boundary (Fig. 4.2 (c)): for $n > 1$, B-splines beyond the domain also contribute to the solution. Fig. 4.2 (d) shows the approximation of the step function with cubic B-spline and the approximation error. When the step function is sampled, the B-splines coinciding with grid nodes will result in an exact representation of the step function. High-order Lagrange basis functions in FEM a not shift-invariant and tend to oscillate (2.5 (b)).

4.4.2 One-dimensional comparison

We begin with a didactic one-dimensional example that compares the accuracies of a Tensor B-spline solver and an FEM solver. The problem is defined by a Diffusion PDE with Robin BC on a one-dimensional domain $\Omega \subset \mathbb{R}$:

$$-\nabla \cdot (D(x)\nabla\varphi(x)) + \mu_a(x)\varphi(x) = q(x), \; x \in \; \Omega, \tag{4.4}$$

$$2D(x)\frac{\partial\varphi(x)}{\partial n} + \varphi(x) = 0, \; x \in \partial\Omega. \tag{4.5}$$

The domain limits are $[-25, 25]$, the source is defined as $q(x) = \exp(-(\frac{x}{2})^2)$. The grid step h was decreased using the expression $h(\mu) = 2^{-\mu}, \; \mu = \{0, 1, 2, 3\}$. Note that both method's bases correspond to the ones shown in Fig. 4.2. The $\mathbb{L}^2$ and $\mathbb{W}^{1,2}$ errors are computed between the numerical solutions and the analytic reference solution. Two situations were studied, in which: 1) additional nodes where introduced in FEM, and 2) FEM was forced to have the same number of nodes as the Tensor B-spline method. The plots of errors are shown in Fig. 4.3.

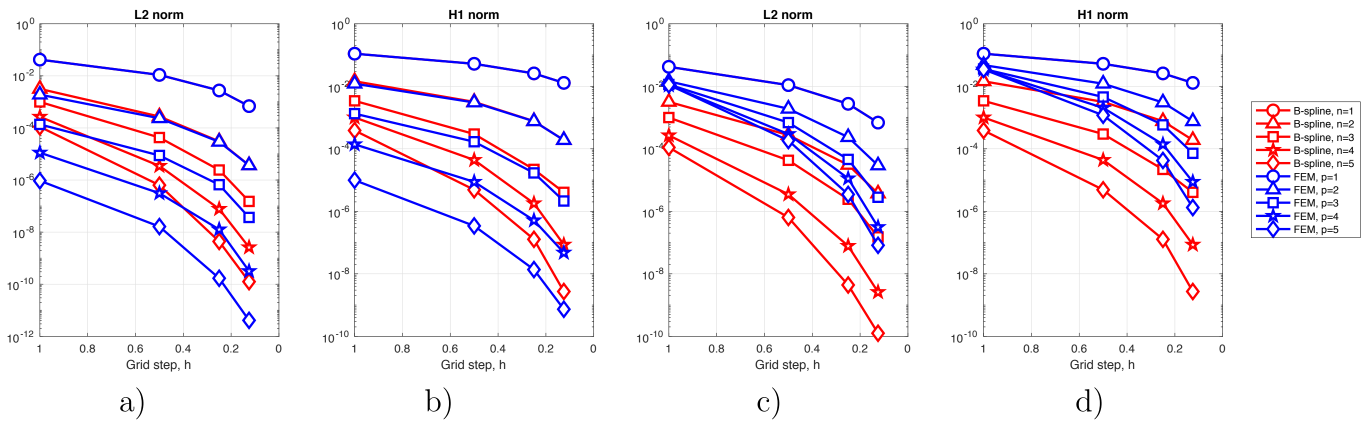


Figure 4.3: $\mathbb{L}^2$ and $\mathbb{W}^{1,2}$ (depicted as $\mathbb{H}^1$) norms of errors between the reference solution and B-spline and the reference and FEM solution. FEM uses additional nodes (a), (b). FEM uses the same number of nodes as B-splines use (c), (d).

Table 4.1: Number of operations for Fig. 4.3.

	B-spline					FEM (with additional nodes)					FEM (same number of nodes)				
h	n=1	n=2	n=3	n=4	n=5	p=1	p=2	p=3	p=4	p=5	p=1	p=2	p=3	p=4	p=5
1	251	465	665	895	1095	251	701	1351	2201	3251	251	351	433	529	651
0.5	501	915	1315	1745	2145	501	1401	2701	4401	6501	501	701	892	1101	1301
0.25	1001	1815	2615	3445	4245	1001	2801	5401	8801	13001	1001	1401	1783	2201	2601
0.125	2001	3615	5215	6845	8445	2001	5601	10801	17601	26001	2001	2801	3592	4401	5201

From Fig. 4.3 one can observe that a high-order FEM with additional nodes is more accurate than a high-order Tensor B-spline, but at the same time, the number of operations for a high-order FEM grows dramatically (Table 4.1). With the same number of nodes, a high-order Tensor B-spline is more accurate while requiring only slightly more operations.

4.5 Comparison with FEM: Two-dimensional Case

We implemented a Tensor B-spline forward ODT solver in MATLAB. The ODT forward problem solves the Diffusion PDE with Robin boundary condition (3.4). Performance critical code parts were implemented in C and MathOberon [77] as MEX files. The regular structure of the solver algorithm permitted extensive code vectorization including efficient use of AVX vector instructions. The solver was tested on a Intel® Core™ i7-2720QM 2.2 GHz CPU with 8 GByte of DDR3 1333 MHz memory.

4.5.1 Error Measures

For a reference solution $\varphi_{i,j}^{ref}$ and a test solution $\hat{\varphi}_{i,j}$ the normalized discrete l_2 norm is defined as

$$||e||_2' = \frac{||\varphi^{ref} - \hat{\varphi}||_{l_2}}{||\varphi^{ref}||_{l_2}} = \frac{\sqrt{\sum_{i,j} |\varphi_{i,j}^{ref} - \hat{\varphi}_{i,j}|^2}}{\sqrt{\sum_{i,j} |\varphi_{i,j}^{ref}|^2}}, \tag{4.6}$$

and signal-to-noise ratio

$$SNR = 10log_{10}\left(\frac{\sum_{i,j}(\varphi_{i,j}^{ref})^2}{\sum_{i,j}(\varphi_{i,j}^{ref} - \hat{\varphi}_{i,j})^2}\right). \tag{4.7}$$

For a given linear system solution estimate $\mathbf{x}$, the relative residual norm is defined as

$$||r||'_2 = \frac{||\mathbf{b} - Op(\mathbf{x})||_{l_2}}{||\mathbf{b}||_{l_2}}, \tag{4.8}$$

where $\mathbf{b}$ is the right-hand side and $Op(\cdot)$ is the linear system operator.

4.5.2 Solver Verification

We verified our Tensor B-spline solver implementation by comparison with reference FEM solutions on very dense meshes, as analytical solutions are not generally available for the given PDE and the associated boundary conditions on domains with arbitrary shapes [68]. FEM is valid as a reference because it is known to converge to the true solution with decreasing element size [25]. Fig. 4.4 shows solver convergence curves on a rectangular domain of size 1 $cm \times 1$ cm with constant diffusion and absorption coefficients. Each value of the normalized l_2 error norm $||e||'_2$ along an abscissa axis from left to right corresponds to halving the grid step (equally in each dimension). The convergence rate $r = log_2(||e_h||'_2/||e_{0.5h}||'_2)$ conforms to the theoretical one ($r_{theor} \approx n+1$) for linear and cubic B-spline as long as the reference solution is sufficiently precise, consistent with the finding in [20] on page 74.

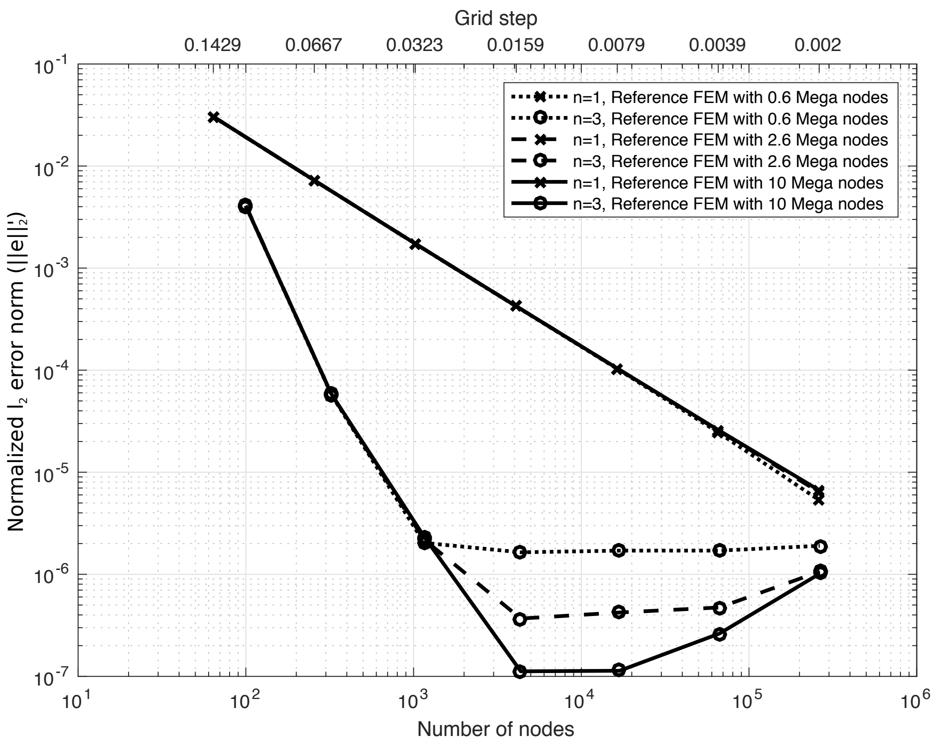


Figure 4.4: Convergence of linear and cubic B-spline solver solution (photon density distribution in the domain including boundary) on a rectangular domain of size 1 $cm \times 1$ cm, $D = 0.33$ cm, $\mu_a = 0.01$ cm^{-1}.

Our results confirm that with the increasing node count, the B-spline solver solution converges to the reference solution, with a significantly higher convergence rate for cubic versus linear splines. The error for the cubic spline improves up to a critical number of nodes, beyond which it does not decrease further. The apparent rebound of the error in Fig. 4.4 is an artifact related to the inaccuracy of the reference; it vanishes progressively with further refinement of the reference. On non-rectangular domains, a similar behavior was observed. Further verification of the B-spline solver was achieved by comparison of its performance versus FEM, as shown below, also using very dense FEM solutions as the reference.

4.5.3 Comparison of B-spline Method versus state-of-the art

Accuracy and computational performance of our solver were compared to standard FEM from Matlab's PDE Toolbox and the Toast++ framework. We used square and circular domains with analytically defined domain boundaries and applied spatially varying diffusion and absorption coefficients as shown in Fig. 4.5, Table 4.2. B-spline coefficients of the optical properties and the source were obtained by spline interpolation [7]. Key results compared to FEM were: a) with a given number of nodes, the B-spline solver converges to a significantly more accurate solution, b) for achieving a prespecified error, the B-spline solver requires significantly fewer computations, c) with a given computational budget, the B-spline solver converges to a significantly more accurate solution. Results using Toast++ were very close to the Matlab FEM, and are therefore not shown.

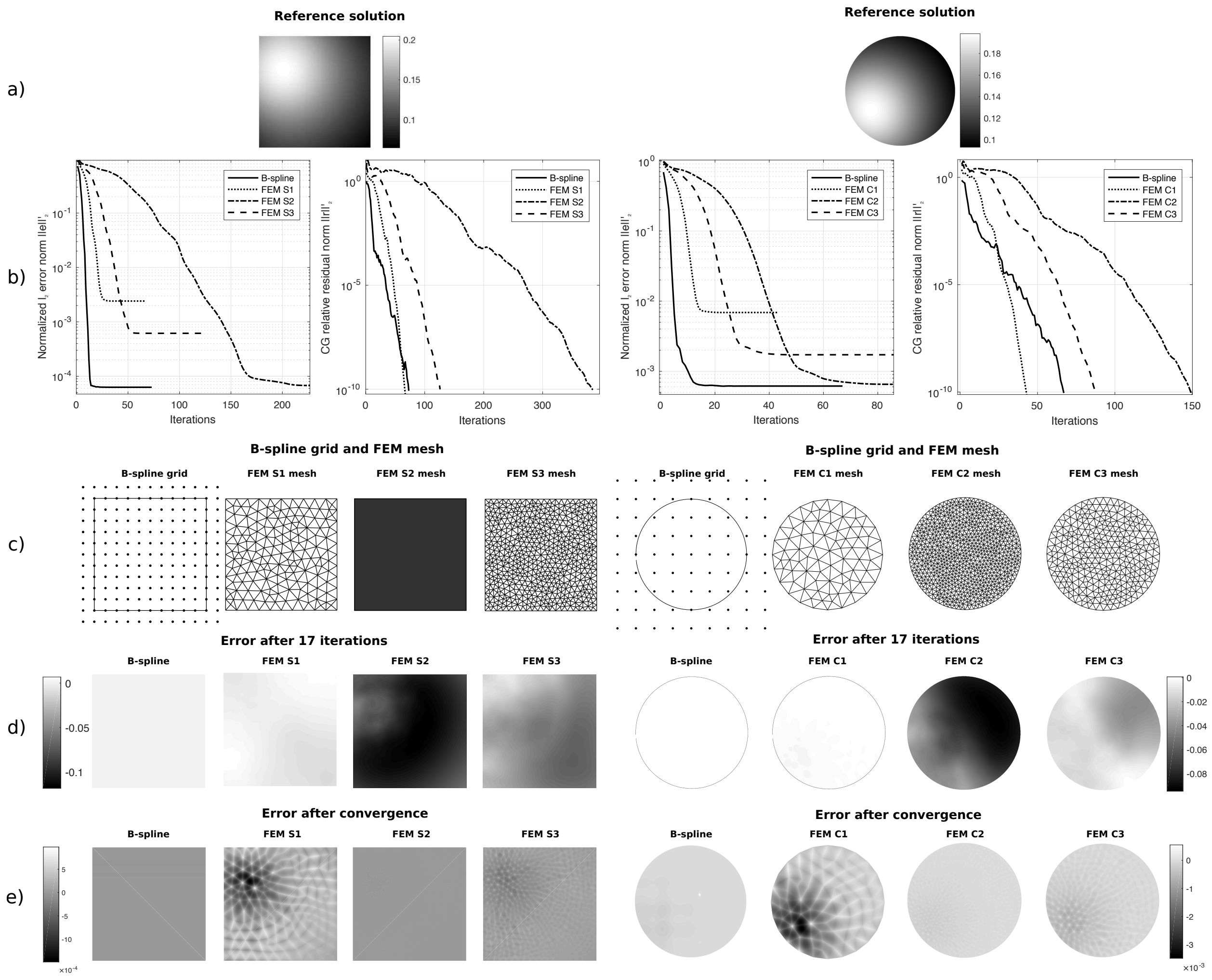


Figure 4.5: B-spline-based solver (cubic B-splines) comparison with linear Matlab FEM on square ($1\ cm \times 1\ cm$) and circular domains ($1\ cm$ diameter). Source function $Q(x,y) = exp(-10(x-0.3)^2)exp(-10(y-0.2)^2)$. Diffusion coefficient $D(x,y) = 0.33cos(0.3x)cos(0.3y)/2+0.33$, absorption coefficient $\mu(x,y) = 0.01sin(0.3x)sin(0.3y)/2+0.01$, refractive index-mismatch term $\gamma = 1$. (a) Reference solutions for square domain (2558465 nodes) and circular domain (2082817 nodes), (b) Normalized l_2 error norm $||e||_2'$ and CG relative error norm $||r||_2'$, (c) B-spline grid and FEM meshes, (d) Error $\hat{\varphi} - \varphi^{ref}$ after 17 iterations, (e) Error $\hat{\varphi} - \varphi^{ref}$ after CG convergence.

4.5.4 Tensor B-spline Solver Performance

High degree of basis at limited computational cost

The nature of the B-spline basis, including compact support and shift invariance, allows increasing the order of approximation at a limited computational cost. For a polynomial degree > 1, discretization of the PDE by B-splines leads to sparse linear systems with fewer unknowns and a lower number of non-zeros compared to polynomials in FEM. A change of the basis from linear to cubic increases the number of non-zeros by a factor of ~ 20 in FEM, while only an increase by a factor of ~ 6 is observed from linear to cubic B-spline (Fig. 4.6, “nnz, system matrix” column in the table). This is because FEM requires the insertion of extra nodes in contrast to B-splines.

Impact of domain boundary representation

The flexibility of the parametric, B-spline based boundary representation used here permits tuning solver performance. Accuracy depends on the spline degree and the number of parametric nodes [9]. Linear B-splines proved to yield sufficiently high accuracy at low complexity in our case, but higher degree splines are feasible in principle. In contrast to FEM, where the quality of boundary approximation is mesh-dependent, we can control boundary representation independent from the domain discretization.

Table 4.2: Tensor B-spline method comparison with FEM.

	Square					Circle			
	B-spline	FEM S1 (same N_u)	FEM S2 (same $\|\|e\|\|'_2$)	FEM S3 (same FLOP)		B-spline (same N_u)	FEM C1 (same $\|\|e\|\|'_2$)	FEM C2 (same FLOP)	FEM C3
N_u	169	170	5421	614		81	81	866	321
nnz(A)	6241	1104	37493	4140		2601	513	5896	2145
FLOP/it	12313	2038	69565	7666		5121	945	10926	3969
Fixed error $\|\|e\|\|'_2 = 10^{-2}$									
Iters	9	21	108	37		5	14	40	25
FLOP	110817	42798	7513020	283642		25605	13230	437040	99225
Fixed CG relative residual norm $\|\|r\|\|'_2 = 10^{-10}$									
Iters	72	66	384	124		66	42	149	87
FLOP	886536	134508	26712960	958250		337986	39690	1627974	345303
$\|\|e\|\|'_2$	0.000063	0.002414	0.000067	0.000616		0.000876	0.009658	0.000850	0.002618

N_u - number of unknowns, $nnz(A)$ - number of non-zeros in the system matrix,

FLOP/it - floating-point operations per iteration, Iters - numer of iterations

Computational complexity of integration and solver initialization

The computational cost for solver initialization is dominated by computation of the tensor kernel coefficients of the basis functions intersecting the domain boundary (here $\sim 10\%$ of nodes). This can be done by standard 2-D numeric quadratures as in [25, 46]. We found, however, that integration based on the Divergence Theorem, combined with B-spline-based approximation of the parametric integrands (look-up table complexity) significantly reduces computational complexity. For example, to achieve a relative error $< 10^{-14}$ over a single grid cell intersected by a single boundary segment, using a 7×7-point 2-D Gauss quadrature rule for the cubic B-spline requires ~ 25.9 million operations, while our integration algorithm requires a 7-point 1-D parametric quadrature with a complexity of $\sim$ 6.6 million operations and does not require the generation of quadratures adapted to the geometry of each and every boundary cell. The initialization of our solver requires computing the tensor kernels only once. To solve the inverse problem iteratively, the precomputed tensor kernels can be reused at every iteration with an additional cost of about 3920 operations for cubic B-spline ($96n^3 + 128n^2 + 56n + 8$, n is spline degree) per node per iteration for recomputing the system operator, about 1000 times less than the standard integration approach, used in [46]. Thus, this tensor kernel approach can reduce the overall computational complexity of the ODT forward problem solutions.

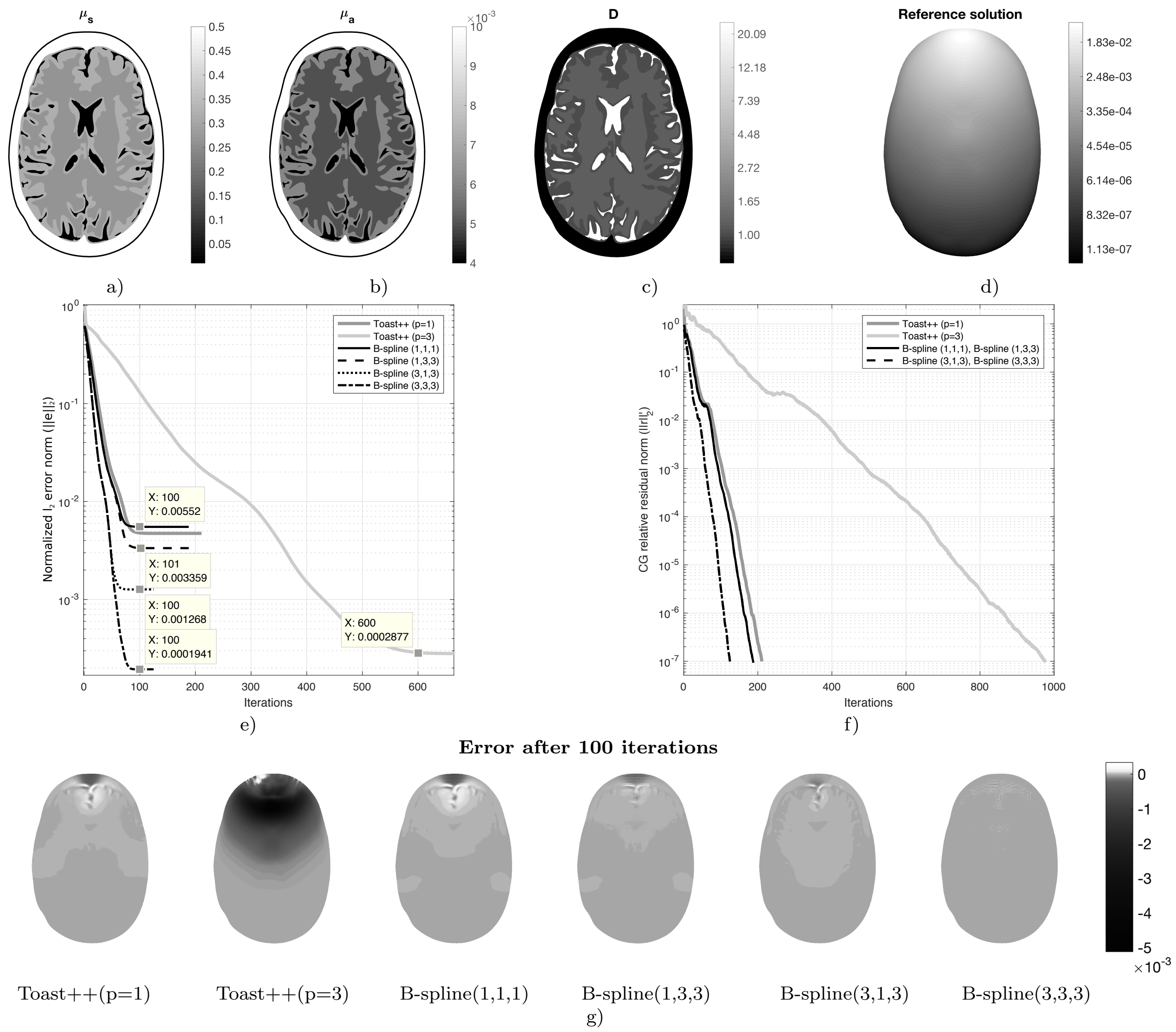


Figure 4.6: B-spline-based solver comparison with Toast++ on a brain phantom of size 180×125 mm. Light source is modeled as a collimated source [78] of Gaussian shape $q(x_1, x_2) = [1/(2\pi\sigma^2)]exp(-[(x_1 - x_1^0)^2 + (x_2 - x_2^0)^2]/(2\sigma^2))$, with $\sigma = 8$ ('Isotropic' source type with 'Gaussian' profile in Toast++). Refractive index mismatch term $\gamma = 2.74$ that corresponds to the skin-air boundary relative refractive index $n = 1.4$ reported in the literature [78]. (a) Reduced scattering distribution μ_s', mm^{-1}, (b) Absorption distribution μ_a, mm^{-1}, [79] (c) Diffusion distribution $D = 1/(3[\mu_a + \mu_s'])$, (d) Reference FEM solution φ^{ref} (5678977 nodes), (e) Normalized l_2 error norm $||e||_2'$ vs iterations, (f) CG relative residual norm $||r||_2'$ vs iterations, (g) Absolute errors $\hat{\varphi} - \varphi^{ref}$ after 100 iterations.

Efficient and flexible mesh-free basis

The shift-invariant B-spline basis does not require the potentially expensive mesh generation needed in FEM. It allows efficient sampling of the involved spline representations [7], in contrast to FEM, where more costly mesh-dependent sampling is required. The B-spline degree to represent solution, diffusion and absorption coefficients, and light source can be chosen independently (Fig. 4.6). This gives added flexibility for tuning the solver towards speed or accuracy, a feature of particular benefit for the on-the-fly solving strategy. Computational efficiency also benefits from the multiresolution B-spline properties that permit simple filtering-based geometric tensor multigrid solving schemes [8] not requiring the generation of scale-dependent meshes implied by FEM [80]. Filtering-based analytical differentiation and integration of B-spline representations permit efficient implementation of the inverse problem regularization strategies based on squared l_2 norms of differential operators [8] and sparse regularizers [81].

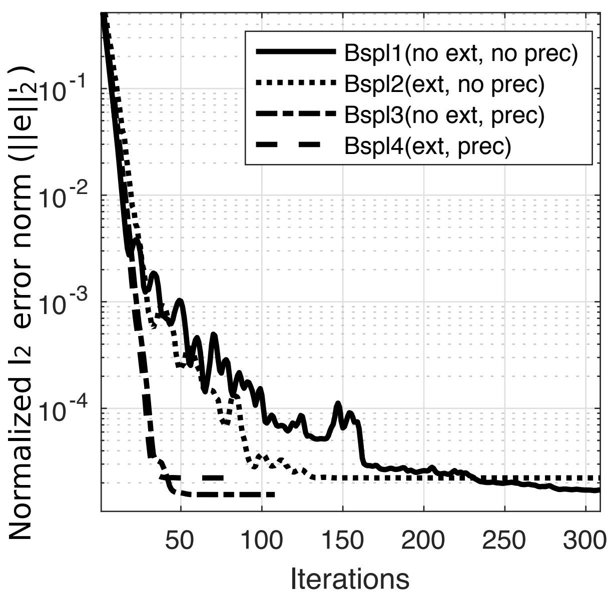
Table 4.3: Tensor B-spline Method comparison with FEM.

Method	degree	N_u	nnz, system matrix	FLOP/ iteration	$\|\|e\|\|_2'$	SNR, dB	Iter-s ($\|\|r\|\|_2' = 10^{-7}$)	CPU time (CG), s	Memory [MB], system matrix (double prec.)
FEM (Toast)	p=1	10282	71368	132454	0.004734	46.50	210	0.27	1.2
FEM (Toast)	p=3	91630	1552294	3012958	0.000281	71.04	975	15.3	24.4
B-spline	$n_b = 1$ $n_p = 1$ $n_s = 1$	10202 (125x98)	90493	170766	0.005515	45.17	187	0.11	1.8
B-spline	$n_b = 1$ $n_p = 3$ $n_s = 3$	10202 (125x98)	90493	170766	0.003347	49.51	187	0.11	1.8
B-spline	$n_b = 3$ $n_p = 1$ $n_s = 3$	10652 (125x98)	503024	995396	0.001268	57.94	124	0.17	9.6
B-spline	$n_b = 3$ $n_p = 3$ $n_s = 3$	10652 (125x98)	503024	995396	0.000194	74.24	124	0.17	9.6

Summary on methods, where p - degree of polynomial basis function in Toast++, degrees of B-spline basis functions for solution, parameters and source are n_b, n_p and n_s, N_u - number of unknowns, nnz, sys. matrix - number of non-zeros in the system matrix

4.5.5 Comparisons on Medical Synthetic Data

We performed comparisons using the MRI brain phantom from [82]. Domain boundaries are represented by parametric splines fitted as an active contour [9] to the phantom. Typical absorption and scattering coefficients were assigned to gray and white matter, bone, and water areas. These images of optical properties were pre-filtered with a Gaussian filter to avoid aliasing and assure non-negativity and were transformed to the B-spline domain by direct filtering [7]. Results computed using our solver performed with several B-spline degrees for the solution, coefficients, and source terms were compared to the results obtained by Matlab FEM and by Toast++. The results shown in Fig. 4.6 document a significant advantage in the accuracy and computational performance of our solver.



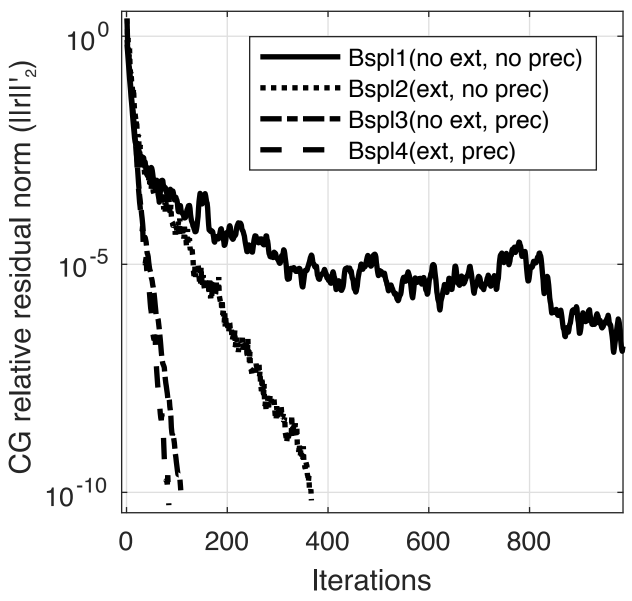


Figure 4.7: B-spline-based solver with/without extension and with/without preconditioner. Solutions obtained on circular domain of radius 25 mm. Reference solution computed by FEM with 8.8 million nodes. Source is a Gaussian of width 4 placed at (24,0). Diffusion and absorption coefficients are smooth, spatially variable functions. a) Normalized l_2 norm of error $||e||_2'$ vs iterations and CG relative residual norm $||r||_2'$ vs iterations for grid of size 21x21.

Table 4.4: A summary of results presented in Fig. 4.7.

Grid	Method	ext.	precond.	N_u	$\|\|e\|\|_2'$	SNR, dB
16x16	Bspl1	no	no	300	1.40e-03	57.09
	Bspl2	yes	no	232	2.59e-03	51.74
	Bspl3	no	yes	300	1.40e-03	57.09
	Bspl4	yes	yes	232	2.59e-03	51.74
21x21	Bspl1	no	no	475	3.47e-04	69.18
	Bspl2	yes	no	387	6.36e-04	63.93
	Bspl3	no	yes	475	3.47e-04	69.18
	Bspl4	yes	yes	387	6.36e-04	63.93
32x32	Bspl1	no	no	996	5.00e-05	86.02
	Bspl2	yes	no	864	7.90e-05	82.03
	Bspl3	no	yes	996	4.99e-05	86.02
	Bspl4	yes	yes	864	7.90e-05	82.03

4.5.6 Solver Convergence

Tensor B-spline solver error and CG residual are presented in Fig. 4.7. The use of the extension for the B-spline basis [45] led to a major improvement of convergence, but we found that a simple Jacobi preconditioner provided even better results. The extension procedure slightly degraded the l_2 error, which could be explained by the effective reduction of the number of nodes. Spikes in the CG residual convergence curves that we observed with the Jacobi preconditioner at high grid density could be overcome by a simple thresholding scheme applied to the system operator coefficients, with a threshold of 10^{-7} in our setup. Even after convergence of the solution error norm $||e||_2'$, the CG relative residual norm $||r||_2'$ still decreases steadily. This nonlinear behaviour is found because the two measures correspond to different mathematical spaces; such behaviour is observed in other applications of CG, a Krylov subspace method [8].

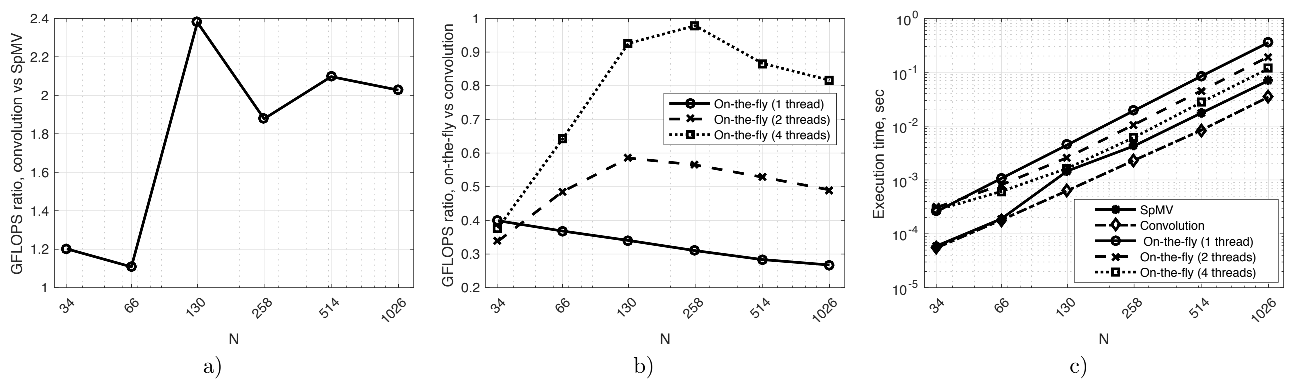


Figure 4.8: Performance plots of sparse matrix-vector multiplication (SpMV), convolution algorithm and on-the-fly algorithm with multiple threads. Here N is a number of nodes along one side of square grid, size of sparse system matrix is $N^2 \times N^2$, size of the system tensor is $7 \times 7 \times N \times N$. (a) Convolution speed-up compared to sparse matrix-vector multiplication (SpMV), (b) On-the-fly performance compared to convolution (c) Execution time for SpMV, convolution and on-the-fly algorithms.

4.5.7 Performance of the System Operator Evaluation

System operator evaluation, performed iteratively by CG, is a critical performance factor for the Tensor B-spline solver. In a standard implementation, this is often done with standard sparse matrix libraries. However, we have shown that the full exploitation of the problem structure can improve performance [15], a finding confirmed in the current work. Fig. 4.8 (a) shows that our spatially variant filtering algorithm for evaluation of the system operator is as twice as fast as the sparse matrix-vector multiplication-based equivalent. A comparison of operation count/CPU clock with CPU time showed that the achieved performance is limited by the memory bottleneck. The tensor structure induced an alternative, on-the-fly version for system operator evaluation based on a decomposition of the system operator (Fig. 3.2). Such an on-the-fly, matrix-free, tensor kernel-based algorithm trades memory transfers for additional CPU load and was found to be highly parallelizable (cf. Fig. 4.8 (b)) and well suited for parallel hardware e.g., SIMD and GPU. The reduction in memory requirement for the system operator renders this algorithm variant very attractive for computing platforms such as FPGA.

4.6 Comparison with FEM: Three-dimensional Case

Compared to 2-D, real-world 3-D problems present new challenges for efficient numerical solving as 1) in FEM, the meshing process becomes challenging and mesh-free solutions may offer advantages [37] but require suited boundary handling algorithms [46], 2) the computation cost of solving increases sharply with increased dimensionality, grid size and higher order of basis functions [8].

We have implemented a three-dimensional Tensor B-spline forward ODT solver in MATLAB. Performance-critical code parts were implemented in C as MEX files using AVX vector instructions and multithreading. The solver was tested and compared with the standard FEM solver from the Toast++ framework [28] on Intel Core i7-2760QM 2.4 GHz with 16GB DDR3 1333 MHz memory. The computations were performed using double precision arithmetic. For the sake of simplicity, in all experiments we used $n_b = n_p = n_s = n$. All comparisons were made in respect to reference solutions computed using an FEM on dense meshes using the strategy proposed in [17].

4.6.1 Error Measures

For a reference volume image $\varphi^{ref}_{i,j,k}$ and a test volume image $\hat{\varphi}_{i,j,k}$ absolute error is defines as

$$\varepsilon_a = \sqrt{\sum_{i,j,k} (\varphi^{ref}_{i,j,k} - \hat{\varphi}_{i,j,k})^2}, \tag{4.9}$$

and relative error is defined as

$$\varepsilon_r = \frac{\sqrt{\sum_{i,j,k} (\varphi^{ref}_{i,j,k} - \hat{\varphi}_{i,j,k})^2}}{\sqrt{\sum_{i,j,k} (\varphi^{ref})^2_{i,j,k}}}. \tag{4.10}$$

4.6.2 Solver Verification

Tensor B-spline solver feasibility, correctness, high accuracy and performance is confirmed by the obtained results presented in Fig. 4.9, Fig. 4.10 and Table 4.6, and Table 4.7.

4.6.3 Advantages of the Tensor B-spline Methodology

Tensor-product B-splines, when used as basis functions in a Ritz-Galerkin formulation of a PDE, result in a very regular data structure that 1) is sparse due to the compact support of B-splines, 2) separable due to the tensor-product basis of B-splines, 3) regular due to the shift-invariance of B-spline, and therefore 4) has a convolutional structure. As a consequence, the Ritz-Galerkin formulation enjoys a regular tensor decomposition structure that is a key factor of efficient computations and memory usage.

Computationally Efficient High-order Basis

The tensor decomposition of a Ritz-Galerkin formulation of a PDE represents the system operator as coefficients contracted with small separable dense kernels in the domain and non-separable sparse kernels on the domain boundary (Fig. 3.3). The separability of domain kernels results in a fewer number of operations, as explained in Table 4.5, where $O_{nonsep} = 4M^6 - 1$, $O_{sep} = 24M^4 - 10M^3 - 1$, $M = 2n + 1$.

Table 4.5: Number of operations for separable and non-separable algorithms.

n	O_{nonsep}	O_{sep}	Ratio, non-sep/sep
1	2915	1673	1.7424
2	62499	13749	4.5457
3	470595	54193	8.6837
4	2125763	150173	14.1554
5	7086243	338073	20.9607

The comparison shows a substantial benefit from separability with the increase of the B-spline degree. For example, in the case of a cubic basis, the separable algorithm requires almost nine times fewer operations than its non-separable equivalent.

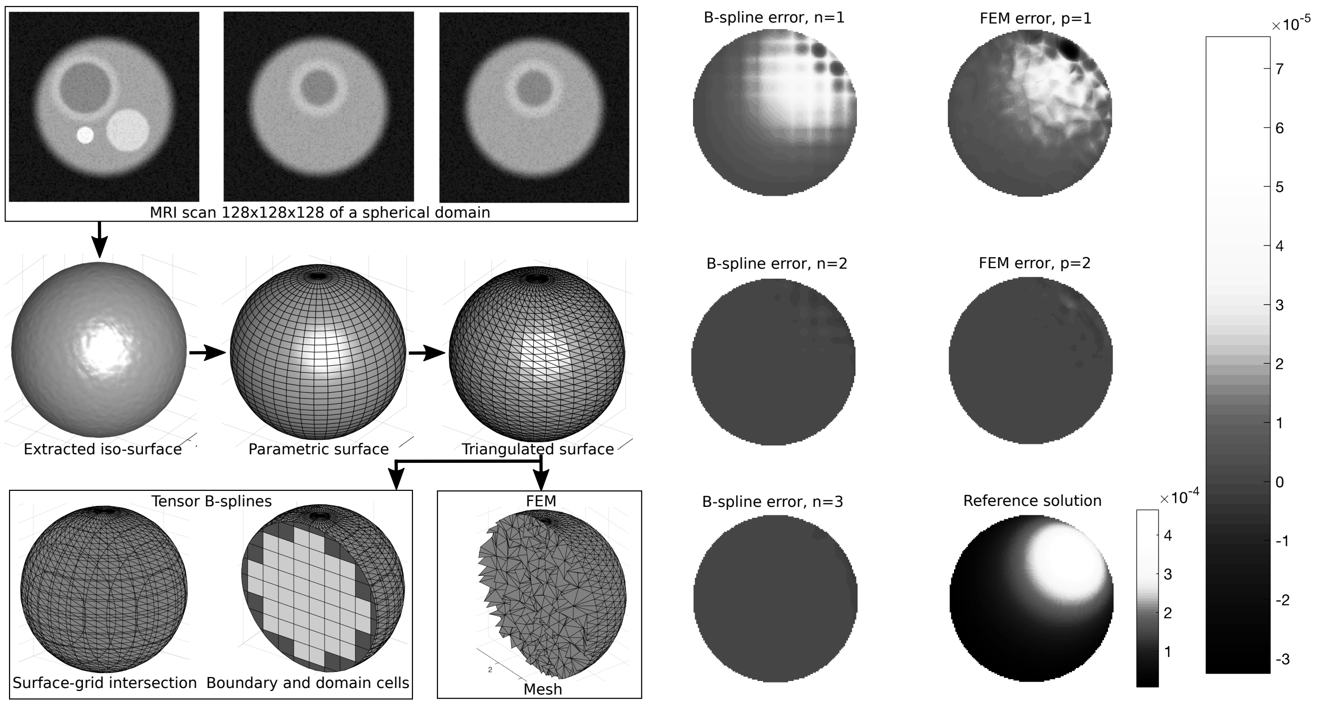


Figure 4.9: B-spline solver performance and comparison with FEM solver on a spherical domain. MRI scan used only for domain surface extraction, and smooth spatially variable absorption μ_a and reduced scattering μ_s' coefficients were assigned with mean values 0.01 mm^{-1} and 1.0 mm^{-1} respectively ($\mu_a = 0.01 sin(0.1x_1)sin(0.1x_2)sin(0.1x_3)/2 + 0.01$, $\mu_s' = cos(0.1x_1)cos(0.1x_2)cos(0.1x_3)/2 + 1.0$).

Mesh-free Integration

The proposed integration method based on the Divergence Theorem enables integration over complex non-convex geometry of the boundary without 3-D meshing. The boundary discretization is independent of the domain discretization, and fine boundary discretizations combined with moderate domain grids are possible (see Fig. 4.10).

Benefits of Tensor Decomposition

The advantage of Tensor B-spline methodology is that the tensor decomposition that comes literally for free enables suppression of memory requirements that is a key factor in solving large-scale problems. The benefit from this decomposition increases with the growth of problem size and degree of basis functions (Table 4.10).

Computational Strategy: 3-D convolution

The results show that a 3-D convolution computational strategy, even in a single-threaded implementation, outperforms the sparse matrix-vector multiplication (SpMV) algorithm (Matlab 2017) and achieves even higher performance with multiple threads (Table 4.8).

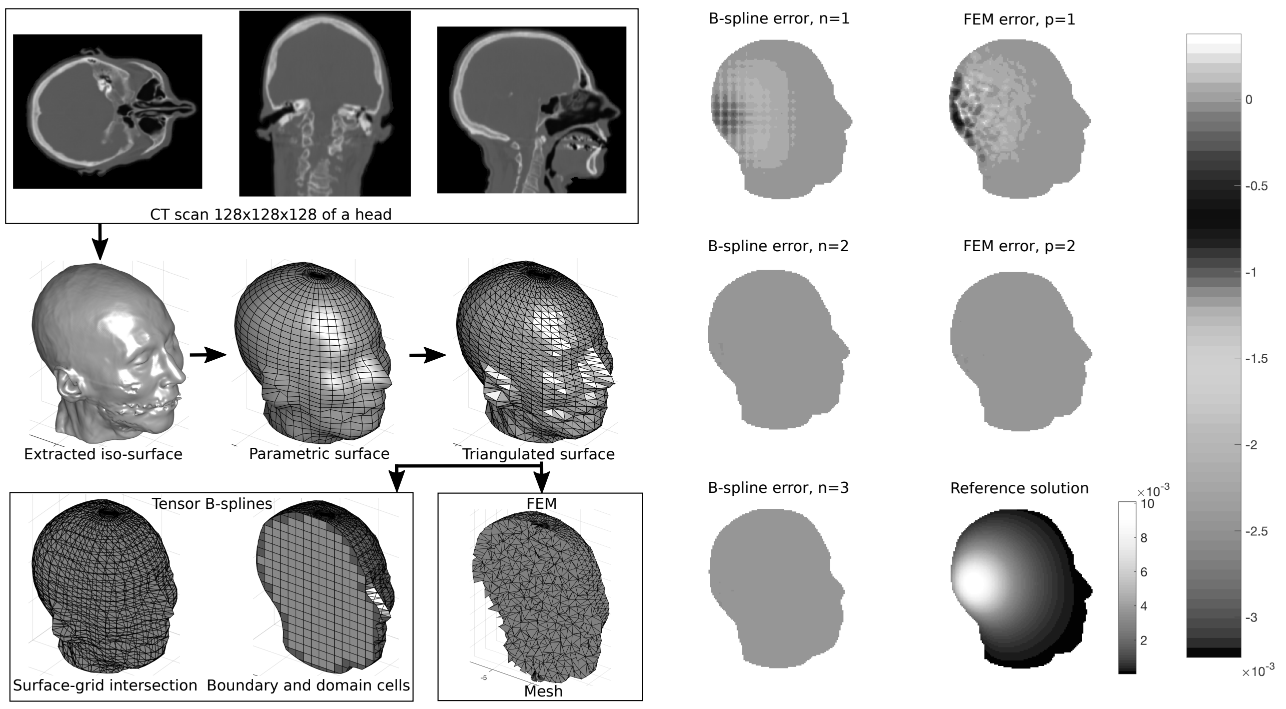


Figure 4.10: B-spline solver performance and comparison with FEM solver on a head domain. CT scan used only for domain surface extraction, and smooth spatially variable absorption μ_a and reduced scattering μ_s' coefficients were assigned with mean values 0.01 mm^{-1} and 1.0 mm^{-1} respectively ($\mu_a = 0.01sin(0.1x_1)sin(0.1x_2)sin(0.1x_3)/2 + 0.01$, $\mu_s' = cos(0.1x_1)cos(0.1x_2)cos(0.1x_3)/2 + 1.0$).

Table 4.6: B-spline solver performance and comparison with FEM solver on a spherical domain.

Method	degree	number of unknowns (nodes)	number of non-zeros, sys. mat.	FLOP/iter	absolute error, ε_a	relative error, ε_r	Iterations of CG*	CPU time (CG*), seconds	Memory [MB], system matrix (double prec.)
FEM	p=1	986	14002	27018	0.01477	0.07186	180	0.0582	0.22118
	p=2	7494	206076	404658	0.00122	0.00592	300	0.4895	3.20165
	p=3	24706	1157470	2290234	N/A	N/A	N/A	N/A	17.85009
B-spline	n=1	984	21070	41156	0.01631	0.07938	23	0.0182	0.32902
	n=2	1482	126504	251526	0.00111	0.00542	100	0.0675	1.94161
	n=3	1778	370490	739202	0.00047	0.00230	500	0.48	5.66680

*- CG is without a preconditioner.

Table 4.7: B-spline solver performance and comparison with FEM solver on a head domain.

Method	degree	number of unknowns (nodes)	number of non-zeros, sys. mat.	FLOP/iter	absolute error, ε_a	relative error, ε_r	Iterations of CG*	CPU time (CG*), seconds	Memory [MB], system matrix (double prec.)
FEM	p=1	3725	52685	101645	0.15512	0.05359	153	0.13	0.83234
	p=2	28156	769276	1510396	0.00441	0.00152	500	3.61	11.95304
	p=3	92663	4318495	8544327	N/A	N/A	N/A	N/A	66.6020
B-spline	n=1	3704	85168	166632	0.10341	0.03573	200	0.29	1.32783
	n=2	4901	471739	938577	0.00731	0.00253	50	0.09	7.23557
	n=3	5551	1328223	2650895	0.00394	0.00143	500	1.5	20.31

*- CG is without a preconditioner.

Table 4.8: System operator evaluation performance comparison: 3-D convolution vs sparse matrix-vector multiplication (SpMV).

n	Grid size	GFLOPS	CPU time, s	GFLOPS	CPU time, s
SpMV (Matlab)					
1	32x32x32	1.4	0.0011		
	64x64x64	1.3	0.0097		
2	32x32x32	1.4	0.0050		
	64x64x64	1.3	0.043		
3	32x32x32	1.4	0.013		
	64x64x64	1.4	0.109		
3-D convolution					
		1 thread		4 threads	
1	32x32x32	1.7	0.00076	2.0	0.00065
	64x64x64	1.7	0.00689	2.8	0.0042
2	32x32x32	1.8	0.00281	2.8	0.0018
	64x64x64	1.8	0.0269	3.7	0.014
3	32x32x32	2.3	0.00491	3.3	0.003
	64x64x64	2.3	0.054	4.0	0.031

Table 4.9: Performance and scalabily of the On-the-fly algorithm*.

		1 thread		2 threads		4 threads	
n	Grid size (L1xL2xL3)	time, s	GFLOPS	time, s	GFLOPS	time, s	GFLOPS
1	32x32x32	0.009	6.4	0.005	12	0.003	19.4
	64x64x64	0.08	5.6	0.04	10.8	0.03	18.4
	128x128x128	0.79	4.6	0.4	9.2	0.22	16.6
	256x256x256	6.5	4.5	3.2	9.0	2.0	15
2	32x32x32	0.08	5.6	0.04	11.2	0.03	17.8
	64x64x64	0.88	4.1	0.47	7.8	0.26	14.2
	128x128x128	8.25	3.6	4.1	7.1	2.6	11.4
	256x256x256	81.8	2.9	36.5	6.5	22.1	10.7
3	32x32x32	0.17	10.3	0.09	19.5	0.06	28
	64x64x64	2.2	6.5	1.2	12	0.7	21
	128x128x128	22.5	5.1	11.7	10	7.56	15.3
	256x256x256	225.2	4.1	104.5	8.8	69.3	13.3

* - the boundary tensor kernels are recomputed and contracted with the coefficient tensors $\mathscr{D}$ and $\mathscr{M}$ according to 3.3(b). The computational load is dominated by the domain tensor kernel computations.

Table 4.10: Comparison of memory requirements (double precision) for SpMV, convolution and on-the-fly computational algorithms.

n	MxMxM	Grid size L1xL2xL3	SpMV memory*	Convolution memory**	On-the-fly memory***	
					30% of boundary kernels	50% of boundary kernels
1	3x3x3	32x32x32	13.40 MB	7.25 MB	2.26 MB	2.94 MB
		64x64x64	110.70 MB	58.00 MB	18.10 MB	23.50 MB
		128x128x128	$\approx$ 928 MB	464.00 MB	144.80 MB	188.00 MB
		256x256x256	$\approx$ 7.5 GB	3.62 GB	1.13 GB	1.47 GB
		512x512x512	$\approx$ 58 GB	29.00 GB	9.05 GB	11.75 GB
2	5x5x5	32x32x32	56.50 MB	31.75 MB	5.94 MB	9.06 MB
		64x64x64	478.40 MB	254.00 MB	47.50 MB	72.50 MB
		128x128x128	$\approx$ 4 GB	1.98 GB	380.00 MB	580.00 MB
		256x256x256	$\approx$ 32 GB	15.88 GB	2.97 GB	4.53 GB
		512x512x512	$\approx$ 254 GB	127.00 GB	23.75 GB	36.25 GB
3	7x7x7	32x32x32	146.10 MB	86.25 MB	14.11 MB	22.69 MB
		64x64x64	1.24 GB	690.00 MB	112.90 MB	181.50 MB
		128x128x128	$\approx$ 11 GB	5.39 GB	903.20 MB	1.42 GB
		256x256x256	$\approx$ 87 GB	43.12 GB	7.06 GB	11.34 GB
		512x512x512	$\approx$ 690 GB	345.00 GB	56.45 GB	90.75 GB

* - the memory is needed to store 2-D sparse system matrix A, $\mathscr{C}$ and $\mathscr{Q}$

** - the memory is needed to store 6-D array $\mathscr{P}_{k_1k_2k_3l_1l_2l_3}$, $\mathscr{C}$ and $\mathscr{Q}$

*** - the memory is needed to store 3-D arrays $mask$, $\mathscr{D}$, $\mathscr{M}$, $\mathscr{C}$ and $\mathscr{Q}$

Computational Strategy: On-the-fly algorithm

The results show that the "on-the-fly" algorithm has better CPU utilization than the 3-D convolution algorithm and the SpMV algorithm (Table 4.9). For example, for cubic B-splines on a grid of $64 \times 64 \times 64$, the "on-the-fly" algorithm shows 21 GFLOPS, while only 4 GFLOPS with the convolution algorithm and 1.4 with the SpMV algorithm are achieved. Despite our sub-optimal implementation, the "on-the-fly" algorithm provides good parallelization and CPU utilization. As expected, for a fixed problem size, an increase in the number of threads results in shorter execution times, and higher GFLOPS numbers. A decrease of GFLOPS with the increase of the grid size in Table 4.9 can be explained by the limited size of the CPU cache memory used in the computing hardware.

Efficient Memory Usage

The 3-D convolution algorithm needs two times less memory than the SpMV, however, the "on-the-fly" algorithm has the lowest memory requirements (Table 4.10). To achieve that, the boundary kernels are computed and convolved with coefficients without explicit storage of 6-D tensor kernels. It was found that the sparsity of boundary kernels reduces the size of the resulting tensor at least by a factor of two. Although the memory consumption depends on the percentage of boundary kernels, even in the case of 50% boundary kernels, the problems up to $256 \times 256 \times 256$ can be solved with cubic B-splines on desktop computers equipped with 16 GB of memory.

4.6.4 Tensor B-spline Solver Comparison with FEM

We compared the properties of the 3-D Tensor B-spline ODT forward solver with the state-of-the-art FEM solver from the Toast++ framework. The support of cubic basis functions in FEM is not implemented in any of the ODT frameworks, however we provide some estimated numbers for this case. For 3-D mesh generation we used the "gmsh" software [83]. The reference, FEM, and B-spline solutions were compared on a grid of size $128 \times 128 \times 128$. The light source was modelled as a collimated source [78] of Gaussian shape $q(\mathbf{x}) = (1/(\sqrt{(2\pi)^3}\sigma^3))exp\{-\,||\mathbf{x} - \mathbf{x}_0|\,|^2/(2\sigma^2)\}$, where $\sigma = 2$ ("Isotropic" source type with "Gaussian" profile in Toast++), $\mathbf{x}_0$ defines the position of the source. The range of values of absorption and reduced scattering coefficients in the presented simulations corresponds to realistic optical properties of biological tissues [79]. The results are presented in Fig. 4.9, Fig. 4.10, Table 4.6, and Table 4.7.

Spherical domain

Fig. 4.9 and Table 4.6 show the results of the comparison on a spherical domain. The reference solution was obtained by quadratic FEM on a mesh with 171910 nodes. The domain bounding box is $[-5, 5] \times [-5, 5] \times [-5, 5]$. The grid size is $11 \times 11 \times 11$ for $n = 1$ and $13 \times 13 \times 13$ for $n = 2, 3$. The grid step is $h = 1$ in all dimensions. The source is placed at $\mathbf{x}_0 = [-5, -5, -5]$.

Human head domain

Fig. 4.10 and Table 4.7 show the results of the comparison on a human head domain. The reference solution was obtained by quadratic FEM on a mesh with 135548 nodes. The domain bounding box is $[-10, 10] \times [-10, 10] \times [-10, 10]$. The grid size is $21 \times 21 \times 21$ for $n = 1$ and $23 \times 23 \times 23$ for $n = 2, 3$. The grid step is $h = 1$ in all dimensions. The source is placed at $\mathbf{x}_0 = [-7, -4, 0]$.

From Fig. 4.9 and Fig. 4.10 one can observe that the most significant error values are concentrated around the light source position and resemble the pattern of the underlying discretization that is regular in case of B-splines and unstructured in case of FEM.

In our comparison, both the Tensor B-spline solver and the FEM solver had the same number of unknowns for the case of linear basis functions; the order of basis was gradually increased (Table 4.6, 4.7). The comparison showed that:

1. The Tensor B-spline solver requires fewer floating point operations per iteration in comparison to the FEM solver when high-order ($n, p > 1$) basis functions are used. For example, with cubic basis functions the Tensor B-spline solver and the FEM solver require 0.74 MFLOP/iteration and 2.3 MFLOP/iteration respectively (column "FLOP/iter" in Table 4.6, 4.7). This result is explained by a sparser system operator (system matrix) of the Tensor B-spline solver rather than of the FEM solver. Indeed, the growth of the number of non-zeros in the system operator of the Tensor B-spline solver is slower than in the system operator of the FEM solver. In particular, in the case when a linear basis was extended to a cubic basis the number of non-zeros increased by a factor of 17.5 for the Tensor B-spline solver and by 83 for the FEM solver (column "number of non-zeros" in the Table 4.6, 4.7).

2. The Tensor B-spline solver provides solutions of equivalent or higher accuracy (columns "absolute error" and "relative error" in Table 4.6, 4.7) while requiring fewer iterations.

3. The Tensor B-spline solver requires less memory space for storing the system operator (system matrix) then the FEM solver when a high-order basis ($n, p > 1$) is applied (last column in Tabels 4.6, 4.7).

As seen from Tables 4.6 and 4.7, column 3, the increase of basis degree results in the rapid growth of the number of unknowns for FEM, while for Tensor B-splines the number of unknowns grows only slightly. In the case of FEM, this is explained by the insertion of nodes for each element of the mesh, when the B-spline solver requires the addition of a limited number of nodes at the boundary only. As observed from our experiments (e.g., Table III and our results in [17]) and supported by the results in [20], an increase of the B-spline degree can be more beneficial in terms of both the accuracy and the computational cost than a mere increase of the grid size.

A limited number of grid nodes in the presented examples was chosen to allow a simple didactic comparison with FEM. However, for practical problems significantly larger grid sizes could be required. Table 4.9 and 4.10 document that the approach is feasible and performant for more than 16 million grid nodes. Increasing the number of grid nodes will improve accuracy as described in [20] and verified experimentally by us in [17].

4.7 High-performance Solution of Large-scale Problems

The performance of the system operator as well as the performance of the solution in three-dimensions was evaluated on the heterogeneous workstation. The computation of the system operator is stated to be the most critical part of the Tensor B-spline solver. The memory consumption depends on the problem size and on the computational strategy. In this section, we perform the evaluation of the system operator performance and conjugate iterative solver on a problem with a large number of nodes.

4.7.1 Target Platform for Numerical Computations

A simplified diagram of the architecture of the workstation is shown in Fig. 4.11. The workstation's CPU AMD EPYC 7401P has four non-uniform memory access (NUMA) nodes (shown in dark gray). Each node is connected to its own random-access memory (RAM) domain and to the other nodes. There are 128 GB ($4 \times 2 \times 16$ GB) of RAM in total. Each node contains a multi-core processor with floating-point SIMD units and several levels of cache memory. Three nodes are connected to GPUs AMD Radeon Vega Frontier Edition. Each GPU has 4096 stream processors (shown in light gray) and 16 GB of RAM.

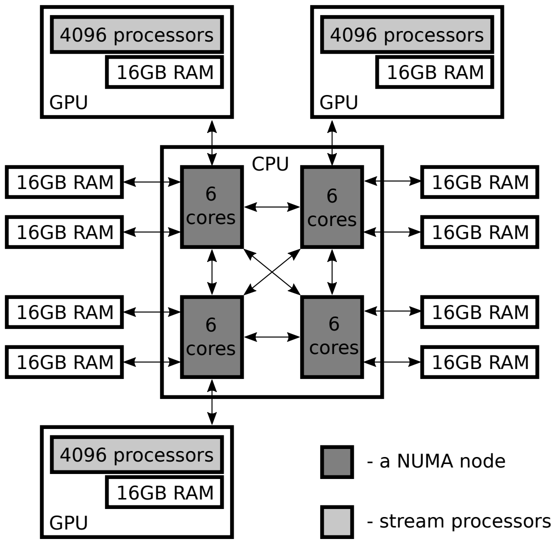


Figure 4.11: The target high-performance workstation consisting of the AMD EPYC 7401P 2.0 GHz 24 cores CPU, 128GB RAM, and three AMD Radeon Vega Frontier Edition 16 GB RAM GPUs.

4.7.2 Multi-threaded Implementation

The platform's massive parallelism and non-uniform memory access could challenge the efficient implementation of a numerical algorithm. However, the Tensor B-spline method allows us to use data parallelism in a very intuitive way. Fig. 4.12 depicts a possible pattern of data distribution that was applied to the CPU-based computations in this paper. The rectangular domain of size $L1 \times L2 \times L3$ is divided into rectangular sub-domains (chunks) of size $L1 \times L2 \times L3/4$. The chunks are bound to NUMA nodes and further divided into blocks of size $L1/M \times L2/M \times L3/4/N$ in order to be processed independently and simultaneously by threads involving SIMD instructions. One can adjust the number of threads per node (N) as well as the number of blocks per node (M) to tune the performance and CPU load. The majority of computations is represented by filtering of input data (see Fig. 3.3) performed with the use of fused multiply-add (FMA) SIMD instructions.

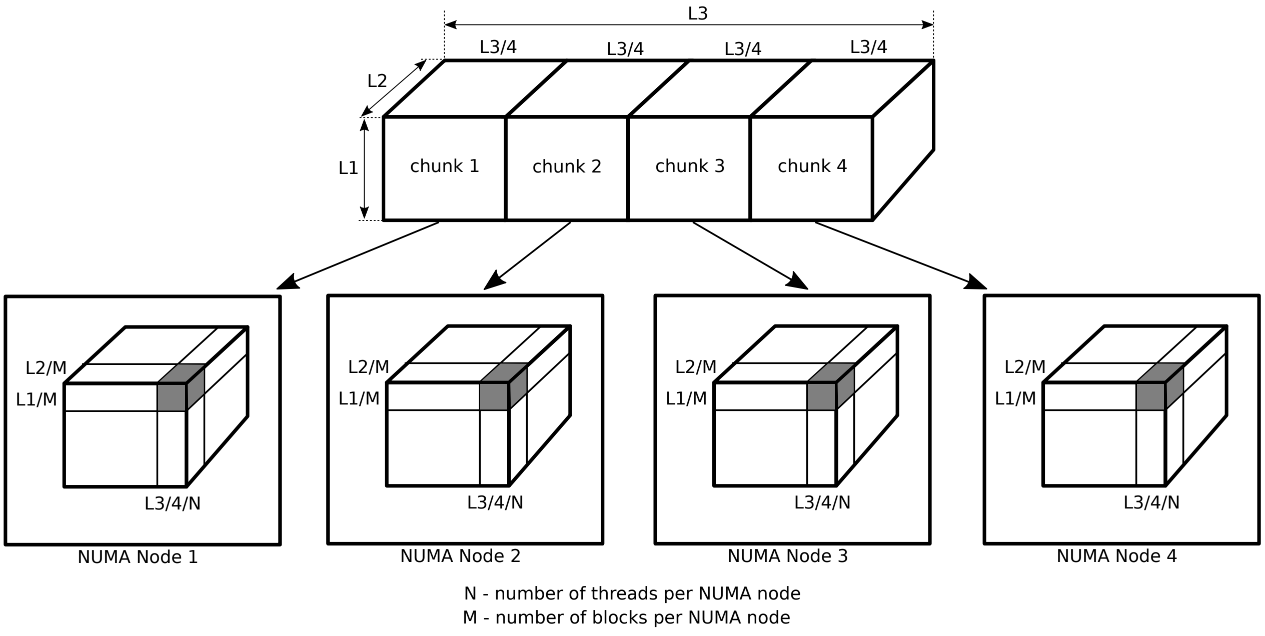


Figure 4.12: The pattern of data distribution for CPU-based computations.

The workstation runs 64-bit Linux Ubuntu 16.04.5 LTS. We used GCC 8.1.0 compiler, POSIX threads for parallelization, and "libnuma" library for memory management and threads binding to NUMA nodes, AVX instructions for vectorized floating point computations. The GPU-based computations were performed using an Active Oberon tensor runtime with OpenCL support.

4.7.3 NUMA-optimized Conjugate Gradient

The main steps of a multi-threaded conjugate gradient solver with Jacobi preconditioner are presented below. Note that synchronization procedures (barriers) are not shown for the sake of simplicity. The visual representation of "copy blocks" $CB(\ldots)$ and "sum blocks" $SB(\ldots)$ procedures are shown in Fig. 4.13 and Fig. 4.14. The computation of system operator $SysOp(\ldots)$ is performed with "on-the-fly" strategy according to the Fig. 3.2 or Fig. 3.3.

We define $M = 2n + 1$, where n is the B-spline degree. Terms of domain kernels decompositions $\hat{w}_{km}$, $\hat{f}_{km} \in \mathbb{R}^{M\times M}$, and $\hat{r}_j \in \mathbb{R}^M$. Boundary kernels $\overline{\mathfrak{w}}_{\mathbf{km}} \in \mathbb{R}^{M\times M\times M\times M\times M\times M}$, $\overline{\mathfrak{r}}_{\mathbf{j}}$, $\overline{\mathfrak{h}}_{\mathbf{j}} \in \mathbb{R}^{M\times M\times M}$. Tensor $MASK_n \in \mathbb{Z}^{L_1\times L_2\times L_3/N}$ contains types of B-spline kernels, tensors C_n, K_n, Q_n, $U_n \in \mathbb{R}^{L_1\times L_2\times L_3/N}$ contains PDE solution, coefficient, source, values of Dirichlet BC. Tensors $SYSOP_n$, $DIAG_n$, RHS_n, $RESID_n$, $P \in \mathbb{R}^{L_1\times L_2\times L_3/4}$ are used for intermediate results.

CG Initialization for n-th NUMA node

1. initialize $RHS_n := 0$, $SYSOP_n := 0$, $DIAG_n := 0$, $C_n := 0$;
2. compute diagonal $DIAG_n := diag(MASK_n, K_n, \hat{w}, \hat{f}, \hat{r}, \overline{\mathfrak{w}}_n, \overline{\mathfrak{r}}_n, \overline{\mathfrak{h}}_n)$;
   sum blocks $SB(node, thread, DIAG_n)$;
   inverse diagonal values;
3. compute right-hand side $RHS_n = rhs(MASK_n, Q_n, U_n, \hat{r}, \overline{\mathfrak{r}}_n, \overline{\mathfrak{h}}_n)$;
   copy blocks $CB(node, thread, RHS_n)$;
4. compute right-hand side norm $rhsNorm := \sqrt{\sum_n ||(RHS_n||}$;
5. compute the system operator
   $RESID_n = SysOp(MASK_n, C_n, K_n, U_n, \hat{w}, \hat{f}, \hat{r}, \overline{\mathfrak{w}}_n, \overline{\mathfrak{r}}_n, \overline{\mathfrak{h}}_n)$;
   copy blocks $CB(node, thread, RESID_n)$;
6. compute residual $RESID_n := RHS_n - RESID_n$;
7. compute norm of the residual $residNorm := \sqrt{\sum_n ||RESID_n||}$;
8. compute preconditioner $P_n := RESID_n DIAG_n$;
   $P_n := RHS_n$;
   copy blocks $CB(node, thread, P_n)$;
9. compute $\rho := \sum_n RESID_n P_n$.

CG Iterations

1. compute the system operator
   $SYSOP_n = SysOp(MASK_n, P_n, K_n, U_n, \hat{w}, \hat{f}, \hat{r}, \overline{\mathfrak{w}}_n, \overline{\mathfrak{r}}_n, \overline{\mathfrak{h}}_n)$;
2. compute $\alpha := \frac{1}{\rho} \sum_n (P_n \cdot SYSOP_n)$;

3. compute $C_n := \alpha P_n$;
4. compute $RESID_n = RESID_n - \alpha SYSOP_n$;
5. compute norm of the residual $residNorm := \sqrt{\sum_n ||(RESID_n||}$;
   $\rho_{-1} = \rho$;
6. application of the preconditioner $SYSOP_n := RESID_n DIAG_n$;
7. compute $\rho := \sum_n RESID_n SYSOP_n$;
8. update $P_n := (\rho/\rho_{-1})P_n + SYSOP_n$;
   copy blocks $CB(node, thread, P_n)$.

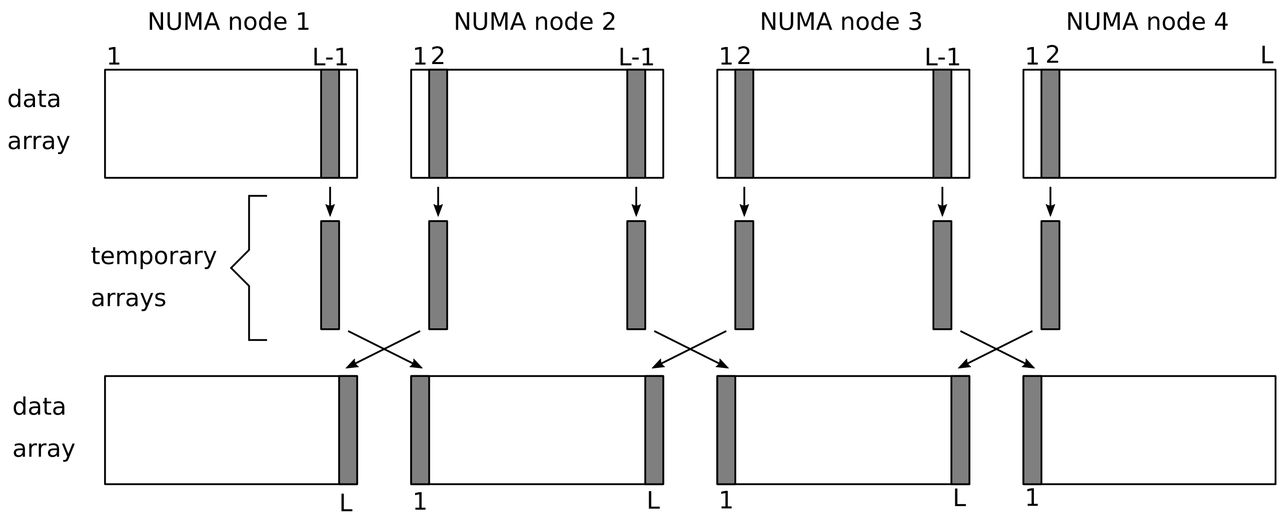


Figure 4.13: Simplified visual diagram of the "copy blocks" $CP(node, tread, P_n)$ procedure for $n = 1$. For three-dimensional problems blocks are two-dimensional.

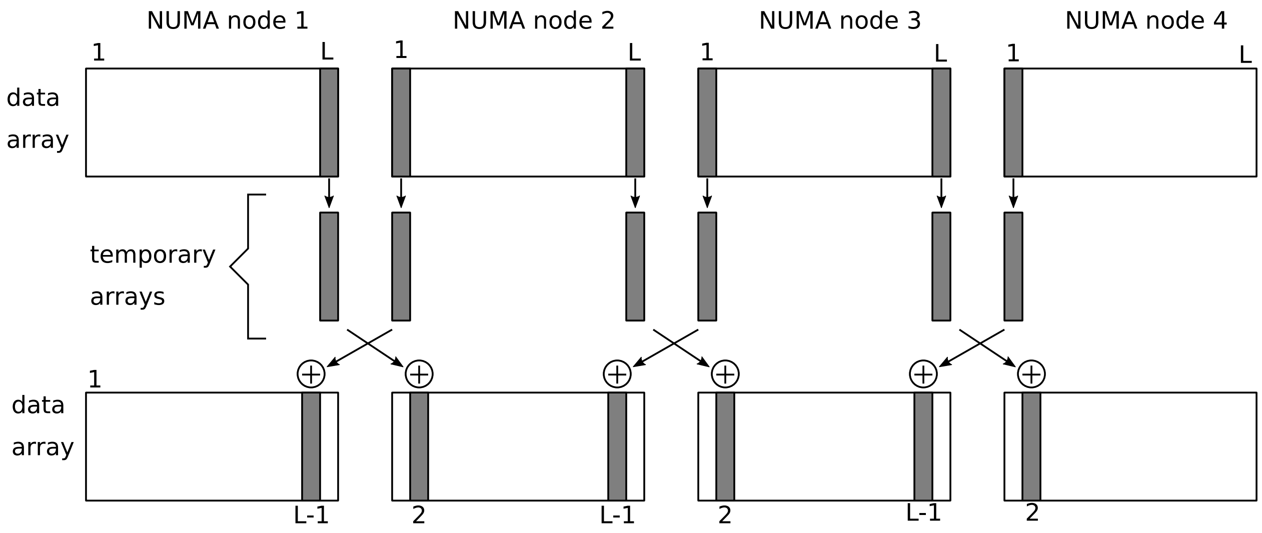


Figure 4.14: Simplified visual diagram of the "sum blocks" $CB(node, tread, DIAG_n)$ procedure for $n = 1$. For three-dimensional problems blocks are two-dimensional.

4.7.4 System Operator Performance

Multi-Core Scalability

The critical property of any numerical algorithm is its scalability since it allows us to understand the potential performance that can be achieved on different platforms and even supercomputers. The scalability of the method is presented in Table 4.11, showing the performance of the on-the-fly algorithm for different numbers of exploited NUMA nodes and threads.

Table 4.11: On-the-fly algorithm scalability performance on the AMD EPYC 7401P CPU. Grid of size 240x240x960, cubic splines.

NUMA nodes	threads	time, s	GFLOPS
	4	84.93	36
1	8	72.43	42
	12	50.90	60
	8	48.10	64
2	16	31.67	96
	24	26.00	117
	12	27.78	110
4	24	17.55	174
	48	14.69	208

CPU Performance

The Tensor B-spline method shows a significant increase in performance when more and more NUMA nodes and threads are used. Table 4.12 shows the performance of the on-the-fly algorithm for different B-spline degrees, grids, and precisions.

Table 4.12: On-the-fly algorithm performance on the AMD EPYC CPU only.

			double precision			single precision		
n	Grid $(L_1 \times L_2 \times L_3)$	GFLOP	MEM R/W, GB	time, s	GFLOPS	MEM R/W, GB	time, s	GFLOPS
1	240x240x960	97	2.11/0.42	1.07	91	1.06/0.21	0.87	111
2	240x240x960	781	2.16/0.44	6.93	113	1.08/0.22	3.65	213
3	240x240x960	3054	2.22/0.44	15.11	202	1.11/0.22	9.87	309
1	480x480x960	388	8.38/1.68	3.81	102	4.19/0.84	3.72	104
2	480x480x960	3124	8.52/1.70	21.75	144	4.26/0.85	14.14	221
3	480x480x960	12214	8.66/1.73	59.81	204	4.33/0.87	40.10	305
1	960x960x960	1552	33.37/6.67	14.34	108	16.69/3.34	13.36	116
2	960x960x960	12496	33.79/6.76	83.85	149	16.89/3.38	57.05	219
3	960x960x960	48857	34.21/6.84	237.90	205	17.10/3.42	163.76	298
1	1200x1200x1200	3031	65.02/13.00	28.70	106	32.51/6.50	26.62	114
2	1200x1200x1200	24406	65.67/13.13	172.94	141	32.83/6.57	111.65	219
3	1200x1200x1200	95424	66.32/13.26	474.72	201	33.16/6.63	324.12	294

Table 4.12 shows that the Tensor B-spline method achieved 200 GFLOPS in double precision and 300 GFLOPS in single precision when cubic basis functions are used. Despite the very large problem sizes, the on-the-fly strategy provides conservative memory usage; thus, the problem of 1.7 billion nodes requires only 66.3 GB to read and 13.26 GB to write. The amount of used memory almost does not depend on the B-spline degree.

GPU Performance

Table 4.13 depicts GPU performance of the on-the-fly algorithm for single precision.

Table 4.13: AMD RADEON Vega GPU performance for the on-the-fly algorithm.

			One GPU		Two GPUs		Three GPUs	
n	Grid	Memory,GB	time	GFLOPS	time	GFLOPS	time	GFLOPS
3	72x72x72	0.00695	68 ms	212	71 ms	204	-	-
3	144x144x144	0.0556	510 ms	227	512 ms	226	-	-
3	258x258x258	0.3198	3.821 s	243	1.912 s	485	1.333 s	711
3	522x522x522	2.649	21.079 s	352	10.54 s	703	7.4486 s	1055

4.7.5 Large-Scale Solution of the Heat Equation

In the last example, we present the solution of the heat equation in steady state with Mixed BC (Dirichlet and Neumann) in a volume domain $\Omega \subset \mathbb{R}^3$:

$$\begin{aligned} -\nabla \cdot (k(\mathbf{x})\nabla\varphi(\mathbf{x})) &= g(\mathbf{x}),\ \mathbf{x} \in \Omega, && (4.11)\\ \varphi(\mathbf{x}) &= \varphi_0,\ \mathbf{x} \in \partial\Omega_1, && (4.12)\\ \nabla\varphi(\mathbf{x}) \cdot \mathbf{n} &= 0,\ \mathbf{x} \in \partial\Omega_2, && (4.13) \end{aligned}$$

where $\varphi(\mathbf{x})$ is the temperature, $k(\mathbf{x})$ is the thermal conductivity, and $g(\mathbf{x})$ is the rate of heat generation.

The domain of numerical computations is obtained by segmentation of computed tomography (CT) scan of a human leg. Skin, blood vessels, surrounding tissue, and the internal volume of the leg were separated based on gray-scale values, holes were closed and the air is added around the skin. The resulting domain size is $600 \times 600 \times 2400$ (≈ 0.8 billions of unknowns). The grid step (resolution) is $h = 0.3$ mm. The visualization of the leg's skin and arteria is shown in Fig. 4.15.

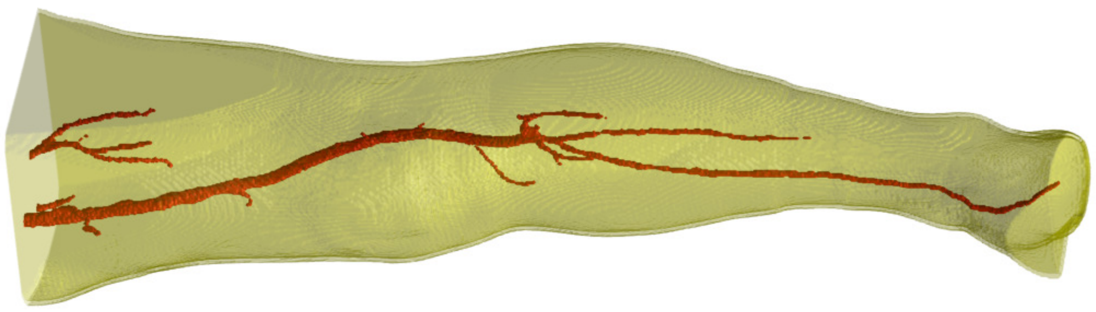


Figure 4.15: The volumetric visualization of the leg's skin and arteria obtained from a CT dataset.

For the numerical computations, a data distribution pattern was used as depicted in Fig. 4.12. There are 98.5 % of domain kernels and 1.5 % of boundary kernels. A linear B-spline basis was used. The solution was computed in double precision. The Dirichlet BC was approximated with Robin BC using the boundary penalty method [63]. The Dirichlet BC sets the temperature on the boundary to 20 $°C$. The advantage of this method is that it does not require the basis functions to vanish on the boundary. The solver uses the parallel implementation of the on-the-fly algorithm of the system operator because neither sparse matrices nor 6-D tensors would fit into the 128 GB of RAM available. A parallel version of the conjugate gradient algorithm with the Jacobi preconditioner was used. In order to speed-up the convergence on such a large grid, the solver was initialized with a solution obtained on a reduced grid of size $120 \times 120 \times 480$. The solution is presented in Fig. 4.16 as maximum-projections images. Each solution was computed in 4 h 4 min for 1000 conjugate gradient iterations.

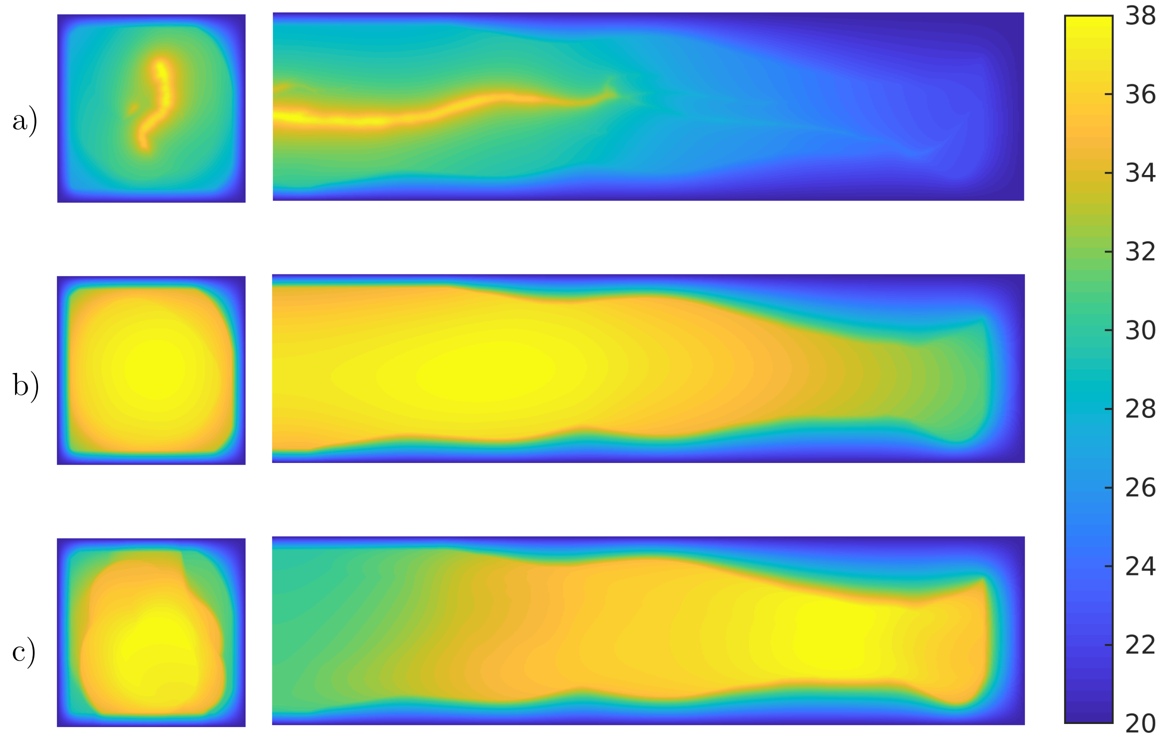


Figure 4.16: The solution of the heat equation in the leg domain. The color bar shows the temperature values in $°C$. The source of heat is a) arteria, b) tissue, c) skin.

4.8 Discussion

This chapter describes the implementation details of the Tensor B-spline method, the evaluation of one-, two- and three- dimensional solvers, and the comparison with the state-of-the-art FEM in application to the forward problem of the optical diffusion tomography.

A general requirement for numerical PDE solvers is the accurate representation of the underlying continuous mathematical model, and from a practical point of view, it is crucial to utilize the computational resources available as efficiently as possible. Both aspects can be addressed successfully if the solution strategy used is flexible enough to adapt to the intrinsic problem structure, and if the algorithms used can be adapted easily to the architecture of the hardware.

4.8.1 Tensor-based High-Performance Computational Strategies

An essential aspect of the Tensor B-spline method is the pervasive application of tensors starting from the PDE formulation and keeping a tensor structure through the solver implementation. This approach preserves the multidimensional data structure in comparison to the classic matrix approach when dimensions are merged and an inherently multidimensional formulation is folded into a two-dimensional matrix structure. Keeping the tensor structure at the implementation stage allows less memory consumption in comparison to the sparse matrix structure where indices of non-zero elements have to be stored.

The data structure of a tensor is more regular in comparison to a sparse matrix; therefore, tensors allow more efficient memory access and use of CPU vector instructions (higher flop/byte ratio). This is confirmed by the "tensor convolution" strategy that requires less memory and is faster than SpMV. Runtime computation of the system operator ("on-the-fly" strategy) reduces the space requirements dramatically. The "on-the-fly" strategy does not compute the multidimensional system operator, but operates on the data and small kernels and needs fewer memory transfers.

Matrix-Free Approach for High-order B-splines

In the case of cubic B-spline basis functions the system operator is represented by a symmetric block-band diagonal matrix of width seven, where each block is symmetric and a band diagonal of width seven. A standard sparse format can be used for efficient storage and matrix-vector multiplication of the system matrix. The drawback of this approach is in high memory space requirement, which grows exponentially with grid size and non-regular memory access that limits the performance of numerical computations (Table 4.10). As we show here, our decomposed tensor-based computational algorithm significantly reduces storage requirements, permits efficient block-wise memory access, and is well-parallelizable. The significant reduction of memory space, however, comes at the cost of a somewhat larger number of computations. The latter, as we showed, can be amortized by using multicore CPUs and GPUs equipped with vector instructions. Moreover, SpMV operations are known for their low computational efficiency [72] [73]. Tensor B-spline methods allow us to use tensor-based computational kernels (3-D convolutions and on-the-fly) that are more efficient than SpMV [17, 18]. This strategy allows solving large-scale problems on inexpensive off-the-shelf hardware and scales naturally to high performance computing on very large systems.

4.8.2 Tensor B-spline Method Comparison with FEM

Regular B-spline Basis

Tensor B-splines do not require meshing and give more sparsity in the system operator in comparison to high-order FEM. In FEM, the increase of the degree of polynomial basis functions leads to significant growth of the number of coefficients (Table 4.6, and Table 4.7) and therefore, the number of non-zeroes in the system matrix. B-splines naturally enable parallelization of computations and structural computational algorithms that can be implemented efficiently on CPU, GPU as we showed, and, potentially, on FPGA. In the Tensor B-spline approach, the solution is defined in the continuous domain and can be evaluated with high efficiency using simple algorithms e.g,. by digital filtering. Due to the irregular geometry of the mesh, the parallelization of computations in FEM is non-trivial, and special methods have to be applied. FEM solution must be resampled numerically if it is to be evaluated on a regular grid.

Efficient High-Order B-spline Basis

The use of high-order approximations is beneficial because faster convergence rates can be achieved [27, 84, 85]. For a specified level of accuracy, high-order methods are typically more efficient than low-order methods. Even more important, high-order Tensor B-splines are usually more efficient than high-order polynomial FEM basis, see [17, 18]. Further, such high-order B-splines require less memory and operations per iteration, and often, the convergence rate is faster, too.

4.8.3 B-splines Approximations

The source term, material properties, and the solution of ODT PDE are consistently expressed as B-spline representations of different orders. We find that the application of cubic B-splines is advantageous in terms of both accuracy and computational performance when compared to lower degree splines. Moreover, the excellent approximation properties of B-splines enable us to use regular grids with a relatively small number of nodes, translating to high computational performance while still allowing to achieve accurate results.

4.8.4 PDE Coefficients representation

Splines are well-known for their excellent interpolation properties [7] that we used for an accurate representation of the coefficients D and μ_a. The accuracy of such representations tends to improve with the increase of B-spline degree and the grid density. We experimentally verified that the use of higher-degree approximations of the coefficients typically leads to significant improvements of the solution accuracy, which was particularly studied in [17] compared to FEM where usually 0-th or 1-st order approximations are used. The increase of the computational cost associated with the use of higher degree representations of the coefficients is moderate, due to the separability of the tensor kernels (see Fig. 3.3 and Table 4.5).

4.8.5 Solving Large-Scale Problems

We have presented a parallel high-performance implementation for multi-core CPU with shared memory and GPUs, and we show numerical results for a large-scale problem consisting of 0.8 billion nodes. We demonstrated that tensor B-spline methods are indeed capable of solving such large-scale problems within reasonable time limits, whereas standard FEM methods run into severe memory problems. Recent advances in element-based methods for large-scale weather forecasting problems [6] suggest a huge potential application field for such tensor B-spline methods. Tensor B-spline methodology can be used for numerical computation of high-performance problems like fluid simulation or weather forecast. The method relaxes the curse of dimensionality without the application of multivariate analysis, has good parallelization properties and allows efficient memory access.

4.8.6 Extended and WEB-Splines

The concept of extended and weighted-extended B-splines for solving PDEs [45], [46] was introduced to reduce instability of the linear system caused by B-splines with only small overlap with the boundary. We examined this approach here and found that it indeed improves stability and convergence, at the price of decreasing regularity of the linear system structure, and increasing computational and memory cost of the solver. Application of a simple preconditioner to the iterative solver leads to similar benefits while keeping the solving algorithm significantly simpler; therefore, we did not use extended B-splines in our final implementation. In our method, like in any mesh-free method where domain discretization does not conform to the domain boundary, the difficulties in the application of Dirichlet boundary conditions arise. To overcome this, the WEB-method [45] can be applied at the cost of more numerical computations and a partial loss of structure regularity due to the multiplication by a weighting function. It is typical for mesh-free methods that integration procedures on the domain boundary are costly. In our work we developed a Divergence theorem-based method that is enabled by the B-spline separability property and, we believe, provides a good trade-off in terms of simplicity, accuracy and numerical cost in comparison to other methods like [86]. Due to tiny contributions of boundary basis functions into the domain, solver convergence instability is potentially possible. According to our experiments, the stability of the solver convergence can be improved by the application of the multigrid solver.

4.8.7 Application to the Optical Diffusion Tomography

The Finite Element Method is state-of-the-art for ODT algorithms. FEM can achieve high precision by combining a domain-adapted mesh with a suited degree of the polynomial basis, where higher degrees often lead to improved approximation quality. This comes at the price of loss of regularity in data structures and integration procedures, the need for creating and maintaining a good mesh and a significant increase of the linear system size. We found that the meshless Tensor B-spline methodology, even with a limited number of grid nodes, is highly competitive with state-of-the-art FEM solvers applied to ODT, even when they are using higher order basis functions. The forward solver implemented here is compatible with ODT frameworks like Toast++, EIDORS so that it can be used as a high-performance plugin.

4.8.8 Integration into Existing Frameworks

The numerical solution of the forward problem PDEs in state-of-the-art image reconstruction frameworks like Toast++ [28], NIRFAST [29] and EIDORS [30], etc. [31] is based on the use of the Finite Element Method (FEM) [25]. These frameworks might benefit from the Tensor B-spline Solver.

4.8.9 Problem-Specific Hardware

Despite ubiquitous utilization of general-purpose CPUs and GPUs, recent studies suggest domain-specific hardware as a future solution for efficient computations [87]. An example of this kind is Google's Tensor Processing Unit containing 65,536 8-bit MAC matrix multiply units with a peak throughput of 92 TeraOps/second. The specialized computing units of such hardware rely on specific properties of the data structure and data flow, as well as on specific data locality patterns [88]. One interesting aspect of tensor B-spline methods is that they may be naturally deployed on such domain-specific hardware, a combination which – at least in our view – has a huge potential in future high-performance computing problems. On the other hand, tensor B-spline methods allow us to solve large-scale problems solution using limited computing resources. This might also suggests the use of these methods in embedded systems with their typical restrictions in terms of on-chip memory and flops-per-watt ratio.

4.8.10 Relevance for Clinical and Biomedical Imaging Applications

Future clinical and biomedical applications of ODT will be made feasible by progress in image acquisition hardware, reconstruction algorithms, and suitable clinical scenarios. Development in ODT has a high potential for the future clinical benefit because it does not require ionizing radiation, radioactive isotopes or physically large machines built upon low-temperature superconducting magnets. Optical imaging is also amenable to clinical molecular imaging using diagnostic, receptor-targeting nanosystems that are currently developed [89], [90]. Progress in ODT technology as reported here may also contribute to lean algorithm implementations running on compact imaging systems and contribute to improved imaging as required for the Personalized Medicine of the Future, and may furthermore render this imaging technique also applicable in healthcare systems with limited resources. Optical tomographic imaging technologies are also increasingly used in biomedical imaging because they enable noninvasive, longitudinal imaging in organ- and cell-specific fashion.

4.8.11 Application to Other Fields of Science and Engineering

In this chapter, an application of the Tensor B-spline methodology was shown based on the examples of the Diffusion and Heat PDEs. Many problems rely on the variant of Diffusion PDEs with certain boundary conditions. However, the methodology has a much broader range of potential applications. Variational formulations of many PDEs share common building blocks like integrals over products of basis functions and their gradients. The Tensor B-spline methodology can be applied to the mechanical problems and problems in physics and biology. Examples of potential applications are elasticity, sound propagation, and fluid dynamics. In the previous chapter we have described the method's approach to solve time-dependent and convection diffusion PDEs, however the evaluation of numerical results will need further work.

Summary

Tensor B-spline method	Advantages	Disadvantages
B-spline representations	high-quality interpolation, approximation, exact differentiation	
B-spline minimal support for a given degree	systems of equations with high sparsity	
Tensor product B-splines	separability, tensor decompositions, 1-D filtering along dimensions, much less computations compared to non-separable bases especially for high-degree basis	less flexible than non-separable bases
B-spline digital filtering algorithms	efficient computations of B-spline coefficients	
tensor-product grid	mesh-free	non-conforming grid for arbitrarily-shaped domain, special handling on domain boundary
tensor decomposition	tensor B-spline kernels, efficient in-domain computations	
tensor structure	preserves spatial coherence that is lost in matrix formulations, highly sparse matrix-free computations, low memory consumption, parallelism and scalability	
integration using Divergence Theorem and B-spline properties	efficient integration on boundaries	

This work was motivated by the search for a numerical method to solve large PDEs that is simple and generic, benefits from recent progress in signal processing algorithms, employs high-degree basis functions in an efficient fashion, does not require domain meshing but handles arbitrarily-shaped domain boundaries, allows to solve large-scale problems on off-the-shelf computers, has efficiently parallelizable algorithms and is scalable to high performance computing. To this end, we employed a B-spline signal processing framework together with computational tensor algebra. As a result, a novel Tensor B-spline method for numerical solution of PDEs is proposed here. The method permits simple and efficient high-order approximations, benefits from the decomposed tensor structure of a discrete PDE formulation, is mesh-free, supports domains with non-convex boundaries of arbitrary shapes and provides efficient matrix-free parallel algorithms for the numerical solution of PDEs.

The use of a B-spline signal processing framework permits a spectrum of efficient filter-based techniques for PDE terms representation and processing. These includes B-spline-based interpolation or approximation, differentiation, integration of the PDE term functions. A computational tensor algebra approach revealed the decomposed tensor nature of the PDE discrete formulation. The tensor formulation of the PDE exposes structure patterns amenable to processing by small kernels, and therefore induces highly performing solving algorithms. We introduced tensor B-spline kernels that are important building-blocks of the method that significantly contribute to the method's computational efficiency. We introduced a decomposed tensor-based storage format that requires less memory then sparse matrix format. Its extreme case of the "on-the-fly" contractions of decomposed B-spline kernels leads to a large reduction in required memory size and allows large-scale solutions with limited memory resources.

Due to the shift-invariant B-spline kernel structure, integration is extremely efficient inside the domain. The integration over the domain surface and its surroundings is performed with the proposed integration method based on the Divergence Theorem and B-spline properties. This integration method effectively reduces the dimensionality of integration and thereby provides efficient integration scenarios. The integration method also benefits from the B-spline-based representation of the domain boundary. We found that the approximation of the Dirichlet boundary condition with the boundary penalty method can be successfully used in the Tensor B-spline method.

The Tensor B-spline solver was successfully applied to the forward problem of Optical Diffusion Tomography. An extensive comparison with state-of-the-art FEM on synthetic and real medical data showed the advantages of our new method. We built high-performance multicore CPU- and GPU-based implementations of the Tensor B-spline solver. The method's parallel implementations exhibit good scalability, that makes it applicable to high performance computing with very large numbers of CPU cores or GPU accelerators. These properties were proved in the method's application to a large-scale problem, where the solution consisting of 0.8 billion of unknowns was obtained on on a 24 CPU / 3 GPU hybrid workstation.

The decomposed tensor aspects of the Tensor B-spline method are used efficiently on large-scale problems. Some limitations are introduced on coarse grids or domains with complicated boundaries where the number of non-separable boundary computations increases. We have proposed a boundary handling approach that can help to overcome these limitations.

Our novel method can solve a variety of PDEs types, can be incorporated in the existing medical imaging frameworks as alternative to standard FEM. The Tensor B-spline method might find applications in many fields, where an efficient, mesh-free, parallel, and scalable numerical method for solving PDEs is needed.

Appendix

Verification of the Bilinear Form

We define two smooth scalar fields $f(\mathbf{x}) = f(x_1, x_2)$ and $g(\mathbf{x}) = g(x_1, x_2)$ are functions on the two-dimensional real space $\mathbf{x} \in \mathbb{R}^2$ that have continuous 1-st derivative. Let us also define the domain $\Omega \subset \mathbb{R}^2$ with boundary $\partial\Omega$.

We have to compute

$$a = \int_\Omega \nabla f(x_1, x_2) \cdot \nabla g(x_1, x_2) dx_1 dx_2 \tag{4.14}$$

over the domain Ω.

We approximate functions $f(x_1, x_2)$ and $g(x_1, x_2)$ by splines of degree n

$$\begin{aligned} \hat{f}(x_1, x_2) &= \sum_{k_1,k_2} c_{k_1,k_2} \beta^n(x_1/h_1 - k_1)\beta^n(x_2/h_2 - k_2), \\ \hat{g}(x_1, x_2) &= \sum_{k_1,k_2} m_{k_1,k_2} \beta^n(x_1/h_1 - k_1)\beta^n(x_2/h_2 - k_2), \end{aligned} \tag{4.15}$$

where c_{k_1,k_2} and m_{k_1,k_2} are coefficients of tensor-product B-spline functions $\beta^n_{\mathbf{k},\mathbf{h}}(\mathbf{x})$. The coefficients can be found by recursive 1-D filtration along the scalar fields dimensions.

Substituting (4.15) into (4.14) we get:

$$a = \mathcal{C}^{\mathbf{k}} \mathcal{M}^{\mathbf{l}} \int_\Omega \nabla \beta^n_{\mathbf{k}}(\mathbf{x}) \cdot \beta^n_{\mathbf{l}}(\mathbf{x}) d\mathbf{x} = \mathcal{C}^{\mathbf{k}} \mathcal{M}^{\mathbf{l}} \mathcal{W}_{\mathbf{kl}} = \mathcal{C}^{k_1 k_2} \mathcal{M}^{l_1 l_2} \mathcal{W}_{k_1 k_2 l_1 l_2} \tag{4.16}$$

Using B-spline properties we compute gradients $\nabla \hat{f}(x_1, x_2)$ and $\nabla \hat{g}(x_1, x_2)$

$$\frac{\partial \hat{f}(x_1, x_2)}{\partial x_1} = \frac{1}{h_1} \sum_{k_1,k_2} c_{k_1,k_2} \left[\beta^{n-1}\left(\frac{x_1}{h_1} - k_1 - \frac{1}{2}\right) - \beta^{n-1}\left(\frac{x_1}{h_1} - k_1 + \frac{1}{2}\right)\right] \beta^n\left(\frac{x_2}{h_2} - k_2\right), \tag{4.17}$$

$$\frac{\partial \hat{f}(x_1, x_2)}{\partial x_2} = \frac{1}{h_2} \sum_{k_1,k_2} c_{k_1,k_2} \beta^n\left(\frac{x_1}{h_1} - k_1\right) \left[\beta^{n-1}\left(\frac{x_2}{h_2} - k_2 - \frac{1}{2}\right) - \beta^{n-1}\left(\frac{x_2}{h_2} - k_2 + \frac{1}{2}\right)\right], \tag{4.18}$$

$$\frac{\partial \hat{g}(x_1, x_2)}{\partial x_1} = \frac{1}{h_1} \sum_{k_1,k_2} m_{k_1,k_2} \left[\beta^{n-1}\left(\frac{x_1}{h_1} - k_1 - \frac{1}{2}\right) - \beta^{n-1}\left(\frac{x_1}{h_1} - k_1 + \frac{1}{2}\right)\right] \beta^n\left(\frac{x_2}{h_2} - k_2\right), \tag{4.19}$$

$$\frac{\partial \hat{g}(x_1, x_2)}{\partial x_2} = \frac{1}{h_2} \sum_{k_1,k_2} m_{k_1,k_2} \beta^n\left(\frac{x_1}{h_1} - k_1\right) \left[\beta^{n-1}\left(\frac{x_2}{h_2} - k_2 - \frac{1}{2}\right) - \beta^{n-1}\left(\frac{x_2}{h_2} - k_2 + \frac{1}{2}\right)\right] \tag{4.20}$$

and integral over Ω explicitly:

$$a = \int_\Omega \nabla \hat{f}(x_1, x_2) \cdot \nabla \hat{g}(x_1, x_2) dx_1 dx_2 = \int_\Omega \frac{\partial \hat{f}(\mathbf{x})}{\partial x_1} \frac{\partial \hat{g}(\mathbf{x})}{\partial x_1} + \frac{\partial \hat{f}(\mathbf{x})}{\partial x_2} \frac{\partial \hat{g}(\mathbf{x})}{\partial x_2} dx_1 dx_2. \tag{4.21}$$

Computation of Tensor B-spline Kernels

Consider the computation of tensor kernel $\mathfrak{r}$ in two-dimensional case. In the domain

$$\widehat{\mathfrak{r}}_{j_1 j_2} = \int_\Omega \beta^{n_s}_{\mathbf{j},\mathbf{1}}(\mathbf{x})\beta^{n_b}_{\mathbf{0},\mathbf{1}}(\mathbf{x})d\mathbf{x} \quad = \beta^{n_s+n_b+1}(j_1)\beta^{n_s+n_b+1}(j_2), \tag{4.22}$$

on boundary

$$\overline{\mathfrak{r}}_{j_1 j_2 l_1 l_2} = \oint_a^b F_1[u_1(t), j_1, l_1] f_2[u_2(t), j_2, l_2]\frac{du_2(t)}{dt}dt, \tag{4.23}$$

$$F_1(t, j_1, l_1) = \int_{-\infty}^t \beta^{n_s}(x - j_1)\beta^{n_b}(x - l_1)dx, \tag{4.24}$$

$$f_2(t, j_2, l_2) = \beta^{n_s}(t - j_2)\beta^{n_b}(t - l_2). \tag{4.25}$$

Tri-product tensor kernels $\mathfrak{w}_{\mathbf{km}}$ and $\mathfrak{f}_{\mathbf{km}}$ can be computed using analytical recursions based on B-spline properties.

Some Geometrical Procedures

Line-line Intersection in 3-D

First line contains points $\mathbf{x}_1 = [x_1, y_1, z_1]$ and $\mathbf{x}_2 = [x_2, y_2, z_2]$. Second line contains points $\mathbf{x}_3 = [x_3, y_3, z_3]$ and $\mathbf{x}_4 = [x_4, y_4, z_4]$. The intersection of these two lines can be found directly by solving system of equations:

$$\mathbf{x} = \mathbf{x}_1 + (\mathbf{x}_2 - \mathbf{x}_1)s \tag{4.26}$$

$$\mathbf{x} = \mathbf{x}_3 + (\mathbf{x}_4 - \mathbf{x}_3)t \tag{4.27}$$

Lines have to be not skew. We solve for

$$s = \frac{(\mathbf{c} \times \mathbf{b}) \cdot (\mathbf{a} \times \mathbf{b})}{|\mathbf{a} \times \mathbf{b}|^2}, \tag{4.28}$$

where

$$\mathbf{a} = \mathbf{x}_2 - \mathbf{x}_1 \tag{4.29}$$

$$\mathbf{b} = \mathbf{x}_4 - \mathbf{x}_3 \tag{4.30}$$

$$\mathbf{c} = \mathbf{x}_3 - \mathbf{x}_1 \tag{4.31}$$

$$\tag{4.32}$$

The point of intersection

$$\mathbf{x} = \mathbf{x}_1 + \mathbf{a}s. \tag{4.33}$$

Line-plane Intersection in 3-D

A line is represented by a parametric from

$$\mathbf{l}_a + (\mathbf{l}_b - \mathbf{l}_a)t, \ t \in \mathbb{R}, \tag{4.34}$$

where, $\mathbf{l}_a = (x_a, y_a, z_a), \ \mathbf{l}_b = (x_b, y_b, z_b)$.

A plane is represented as

$$\mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v, \;\; u, v \in \mathbb{R}, \tag{4.35}$$

where $\mathbf{p}_k = (x_k, y_k, z_k), \;\; k = 0, 1, 2.$

To find line-plane intersection point a parametric equation

$$\mathbf{l}_a + (\mathbf{l}_b - \mathbf{l}_a)t = \mathbf{p}_0 + (\mathbf{p}_1 - \mathbf{p}_0)u + (\mathbf{p}_2 - \mathbf{p}_0)v \tag{4.36}$$

has to be solved.

Bibliography

[1] S. R. Arridge, "Optical tomography in medical imaging," *Inverse problems*, vol. 15, no. 2, p. R41, 1999.

[2] M. Cheney, D. Isaacson, and J. C. Newell, "Electrical impedance tomography," *SIAM review*, vol. 41, no. 1, pp. 85–101, 1999.

[3] M. Soleimani and W. R. Lionheart, "Nonlinear image reconstruction for electrical capacitance tomography using experimental data," *Measurement Science and Technology*, vol. 16, no. 10, p. 1987, 2005.

[4] H. Hallez, B. Vanrumste, R. Grech, J. Muscat, W. De Clercq, A. Vergult, Y. D'Asseler, K. P. Camilleri, S. G. Fabri, S. Van Huffel, *et al.*, "Review on solving the forward problem in eeg source analysis," *Journal of neuroengineering and rehabilitation*, vol. 4, no. 1, p. 46, 2007.

[5] Y.-G. Lv and J. Liu, "Measurement of local tissue perfusion through a minimally invasive heating bead," *Heat and Mass Transfer*, vol. 44, no. 2, p. 201, 2007.

[6] S. Marras, J. F. Kelly, M. Moragues, A. Müller, M. A. Kopera, M. Vázquez, F. X. Giraldo, G. Houzeaux, and O. Jorba, "A review of element-based galerkin methods for numerical weather prediction: Finite elements, spectral elements, and discontinuous galerkin," *Archives of Computational Methods in Engineering*, vol. 23, no. 4, pp. 673–722, 2016.

[7] M. Unser, "Splines: A perfect fit for signal and image processing," *IEEE Signal processing magazine*, vol. 16, no. 6, pp. 22–38, 1999.

[8] O. V. Morozov, M. Unser, and P. Hunziker, "Reconstruction of large, irregularly sampled multidimensional images. a tensor-based approach," *IEEE transactions on medical imaging*, vol. 30, no. 2, pp. 366–374, 2011.

[9] P. Brigger, J. Hoeg, and M. Unser, "B-spline snakes: a flexible tool for parametric contour detection," *IEEE Transactions on image processing*, vol. 9, no. ARTICLE, pp. 1484–1496, 2000.

[10] M. Unser, A. Aldroubi, and M. Eden, "B-spline signal processing. i. theory," *IEEE transactions on signal processing*, vol. 41, no. 2, pp. 821–833, 1993.

[11] M. Unser, A. Aldroubi, and M. Eden, "B-spline signal processing. ii. efficiency design and applications," *IEEE transactions on signal processing*, vol. 41, no. 2, pp. 834–848, 1993.

[12] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009.

[13] A. Cichocki, D. Mandic, L. De Lathauwer, G. Zhou, Q. Zhao, C. Caiafa, and H. A. Phan, "Tensor decompositions for signal processing applications: From two-way to multiway component analysis," *IEEE Signal Processing Magazine*, vol. 32, no. 2, pp. 145–163, 2015.

[14] N. D. Sidiropoulos, L. De Lathauwer, X. Fu, K. Huang, E. E. Papalexakis, and C. Faloutsos, “Tensor decomposition for signal processing and machine learning,” *IEEE Transactions on Signal Processing*, vol. 65, no. 13, pp. 3551–3582, 2017.

[15] O. Morozov and P. Hunziker, “Solving tensor structured problems with computational tensor algebra,” *arXiv preprint arXiv:1001.5460*, 2010.

[16] O. Morozov, *Spline-and tensor-based signal reconstruction: from structure analysis to high-performance algorithms to multiplatform implementations and medical applications.* PhD thesis, University_of_Basel, 2015.

[17] D. Shulga, O. Morozov, and P. Hunziker, “A tensor b-spline approach for solving the diffusion pde with application to optical diffusion tomography,” *IEEE transactions on medical imaging*, vol. 36, no. 4, pp. 972–982, 2017.

[18] D. Shulga, O. Morozov, and P. Hunziker, “Solving 3-d pdes by tensor b-spline methodology: A high performance approach, applied to optical diffusion tomography,” *IEEE Transactions on Medical Imaging*, 2018.

[19] D. Shulga, O. Morozov, V. Roth, F. Friedrich, and P. Hunziker, “Tensor b-spline numerical methods for pdes: a high-performance alternative to fem,” *arXiv preprint arXiv:1904.03057*, 2019.

[20] K. Höllig, *Finite element methods with B-splines.* SIAM, 2003.

[21] J. D. Hoffman and S. Frankel, *Numerical methods for engineers and scientists.* CRC press, 2001.

[22] G. B. Arfken and H. J. Weber, “Mathematical methods for physicists,” 1999.

[23] Y. A. Cengel, S. Klein, and W. Beckman, *Heat transfer: a practical approach*, vol. 141. McGraw-Hill New York, 1998.

[24] R. J. LeVeque, *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, vol. 98. Siam, 2007.

[25] O. C. Zienkiewicz, R. L. Taylor, and R. L. Taylor, *The finite element method*, vol. 3. McGraw-hill London, 1977.

[26] O. C. Zienkiewicz, R. L. Taylor, and J. Z. Zhu, *The finite element method: its basis and fundamentals.* Elsevier, 2005.

[27] B. Guo and I. Babuška, “The hp version of the finite element method,” *Computational Mechanics*, vol. 1, no. 1, pp. 21–41, 1986.

[28] M. Schweiger and S. Arridge, “The toast++ software suite for forward and inverse modeling in optical tomography,” *Journal of biomedical optics*, vol. 19, no. 4, pp. 040801–040801, 2014.

[29] H. Dehghani, M. E. Eames, P. K. Yalavarthy, S. C. Davis, S. Srinivasan, C. M. Carpenter, B. W. Pogue, and K. D. Paulsen, “Near infrared optical tomography using nirfast: Algorithm for numerical model and image reconstruction,” *International Journal for Numerical Methods in Biomedical Engineering*, vol. 25, no. 6, pp. 711–732, 2009.

[30] N. Polydorides and W. R. Lionheart, “A matlab toolkit for three-dimensional electrical impedance tomography: a contribution to the electrical impedance and diffuse optical reconstruction software project,” *Measurement Science and Technology*, vol. 13, no. 12, p. 1871, 2002.

[31] J. Vorwerk, C. Engwer, S. Pursiainen, and C. H. Wolters, “A mixed finite element method to solve the eeg forward problem,” *IEEE transactions on medical imaging*, vol. 36, no. 4, pp. 930–941, 2017.

[32] D.-T. Lee and B. J. Schachter, “Two algorithms for constructing a delaunay triangulation,” *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980.

[33] P.-O. Persson and G. Strang, “A simple mesh generator in matlab,” *SIAM review*, vol. 46, no. 2, pp. 329–345, 2004.

[34] D. S. Lo, *Finite Element Mesh Generation.* CRC Press, 2014.

[35] K.-J. Bathe, “The amore paradigm for finite element analysis,” *Advances in Engineering Software*, vol. 130, pp. 1–13, 2019.

[36] V. P. Nguyen, T. Rabczuk, S. Bordas, and M. Duflot, “Meshless methods: a review and computer implementation aspects,” *Mathematics and computers in simulation*, vol. 79, no. 3, pp. 763–813, 2008.

[37] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs, “Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement,” *Computer methods in applied mechanics and engineering*, vol. 194, no. 39, pp. 4135–4195, 2005.

[38] P. Kaufmann, S. Martin, M. Botsch, and M. Gross, “Flexible simulation of deformable models using discontinuous galerkin fem,” *Graphical Models*, vol. 71, no. 4, pp. 153–167, 2009.

[39] C. Engwer, J. Vorwerk, J. Ludewig, and C. H. Wolters, “A discontinuous galerkin method to solve the eeg forward problem using the subtraction approach,” *SIAM Journal on Scientific Computing*, vol. 39, no. 1, pp. B138–B164, 2017.

[40] R. J. LeVeque, *Finite volume methods for hyperbolic problems*, vol. 31. Cambridge university press, 2002.

[41] R. Eymard, T. Gallouët, and R. Herbin, “Finite volume methods,” *Handbook of numerical analysis*, vol. 7, pp. 713–1018, 2000.

[42] P. K. Banerjee and R. Butterfield, *Boundary element methods in engineering science*, vol. 17. McGraw-Hill London, 1981.

[43] T. Belytschko, Y. Krongauz, D. Organ, M. Fleming, and P. Krysl, “Meshless methods: an overview and recent developments,” *Computer methods in applied mechanics and engineering*, vol. 139, no. 1-4, pp. 3–47, 1996.

[44] S. De and K. Bathe, “The method of finite spheres,” *Computational Mechanics*, vol. 25, no. 4, pp. 329–345, 2000.

[45] K. Höllig, C. Apprich, and A. Streit, “Introduction to the web-method and its applications,” *Advances in Computational Mathematics*, vol. 23, no. 1, pp. 215–237, 2005.

[46] K. Höllig and J. Hörner, "Programming finite element methods with weighted b-splines," *Computers & Mathematics with Applications*, vol. 70, no. 7, pp. 1441–1456, 2015.

[47] N. Nguyen-Thanh, H. Nguyen-Xuan, S. P. A. Bordas, and T. Rabczuk, "Isogeometric analysis using polynomial splines over hierarchical t-meshes for two-dimensional elastic solids," *Computer Methods in Applied Mechanics and Engineering*, vol. 200, no. 21, pp. 1892–1908, 2011.

[48] V. P. Nguyen, C. Anitescu, S. P. Bordas, and T. Rabczuk, "Isogeometric analysis: an overview and computer implementation aspects," *Mathematics and Computers in Simulation*, vol. 117, pp. 89–116, 2015.

[49] N. Nguyen-Thanh, K. Zhou, X. Zhuang, P. Areias, H. Nguyen-Xuan, Y. Bazilevs, and T. Rabczuk, "Isogeometric analysis of large-deformation thin shells using rht-splines for multiple-patch coupling," *Computer Methods in Applied Mechanics and Engineering*, vol. 316, pp. 1157–1178, 2017.

[50] I. Schoenberg, "Cardinal interpolation and spline functions," *Journal of Approximation theory*, vol. 2, no. 2, pp. 167–206, 1969.

[51] C. De Boor, "On calculating with b-splines," *Journal of Approximation theory*, vol. 6, no. 1, pp. 50–62, 1972.

[52] R. H. Bartels, J. C. Beatty, and B. A. Barsky, *An introduction to splines for use in computer graphics and geometric modeling.* Morgan Kaufmann, 1995.

[53] M. Unser, A. Aldroubi, and M. Eden, "The l/sub 2/-polynomial spline pyramid," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, no. 4, pp. 364–379, 1993.

[54] J.-C. Baritaux, S. C. Sekhar, and M. Unser, "A spline-based forward model for optical diffuse tomography," in *2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro*, pp. 384–387, IEEE, 2008.

[55] A. Cichocki, R. Zdunek, A. H. Phan, and S.-i. Amari, *Nonnegative matrix and tensor factorizations: applications to exploratory multi-way data analysis and blind source separation.* John Wiley & Sons, 2009.

[56] H. Si, "Tetgen, a delaunay-based quality tetrahedral mesh generator," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 2, p. 11, 2015.

[57] J. R. Shewchuk, "Delaunay refinement algorithms for triangular mesh generation," *Computational geometry*, vol. 22, no. 1-3, pp. 21–74, 2002.

[58] D. Lukkassen, "A short introduction to sobolev-spaces and applications for engineering students," 2004.

[59] I. M. Smith, D. V. Griffiths, and L. Margetts, *Programming the finite element method.* John Wiley & Sons, 2013.

[60] P. J. Davis and P. Rabinowitz, *Methods of numerical integration.* Courier Corporation, 2007.

[61] V. L. Rvachev and T. I. Sheiko, "R-functions in boundary value problems in mechanics," *Appl. Mech. Rev*, vol. 48, no. 4, pp. 151–188, 1995.

[62] F. Züger, *Solution of Non-Homogeneous Dirichlet Problems with FEM*. PhD thesis, Master Thesis, Zurich University, 2013.

[63] J. W. Barrett and C. M. Elliott, "Finite element approximation of the dirichlet problem using the boundary penalty method," *Numerische Mathematik*, vol. 49, no. 4, pp. 343–366, 1986.

[64] C. Çelik and M. Duman, "Crank–nicolson method for the fractional diffusion equation with the riesz fractional derivative," *Journal of computational physics*, vol. 231, no. 4, pp. 1743–1750, 2012.

[65] J. Stam, "Stable fluids.," in *Siggraph*, vol. 99, pp. 121–128, 1999.

[66] L. Cammoun, C. A. Castaño-Moraga, E. Muñoz-Moreno, D. Sosa-Cabrera, B. Acar, M. Rodriguez-Florido, A. Brun, H. Knutsson, and J. Thiran, "A review of tensors and tensor signal processing," in *Tensors in Image Processing and Computer Vision*, pp. 1–32, Springer, 2009.

[67] S. Mousavi and N. Sukumar, "Numerical integration of polynomials and discontinuous functions on irregular convex polygons and polyhedrons," *Computational Mechanics*, vol. 47, no. 5, pp. 535–554, 2011.

[68] A. P. Gibson, J. C. Hebden, and S. R. Arridge, "Recent advances in diffuse optical imaging," *Physics in Medicine and Biology*, vol. 50, no. 4, p. R1, 2005.

[69] S. R. Arridge, J. P. Kaipio, V. Kolehmainen, M. Schweiger, E. Somersalo, T. Tarvainen, and M. Vauhkonen, "Approximation errors and model reduction with an application in optical diffusion tomography," *Inverse Problems*, vol. 22, no. 1, p. 175, 2006.

[70] Y. Zhai and S. A. Cummer, "Fast tomographic reconstruction strategy for diffuse optical tomography," *Opt. Express*, vol. 17, pp. 5285–5297, Mar 2009.

[71] V. Ntziachristos, J. Ripoll, L. V. Wang, and R. Weissleder, "Looking and listening to light: the evolution of whole-body photonic imaging," *Nature biotechnology*, vol. 23, no. 3, pp. 313–320, 2005.

[72] E. Saule, K. Kaya, and Ü. V. Çatalyürek, "Performance evaluation of sparse matrix multiplication kernels on intel xeon phi," in *International Conference on Parallel Processing and Applied Mathematics*, pp. 559–570, Springer, 2013.

[73] A. Elafrou, G. Goumas, and N. Koziris, "Performance analysis and optimization of sparse matrix-vector multiplication on modern multi-and many-core processors," in *Parallel Processing (ICPP), 2017 46th International Conference on*, pp. 292–301, IEEE, 2017.

[74] K. Hou, W.-c. Feng, and S. Che, "Auto-tuning strategies for parallelizing sparse matrix-vector (spmv) multiplication on multi-and many-core processors," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 713–722, IEEE, 2017.

[75] F. C. Kunter and S. S. Seker, "3d web-splines solution to human eye heat distribution using bioheat equation," *Engineering Analysis with Boundary Elements*, vol. 35, no. 4, pp. 639–646, 2011.

[76] F. C. Kunter and S. S. Seker, "Radially symmetric weighted extended b-spline model," *Applied Mathematics and Computation*, vol. 217, no. 24, pp. 10305–10316, 2011.

[77] F. Friedrich, J. Gutknecht, O. Morozov, and P. Hunziker, “A mathematical programming language extension for multilinear algebra,” *Proc. Kolloqium über Programmiersprachen und Grundlagen der Programmierung, Timmendorfer Strand*, 2007.

[78] M. Schweiger, S. Arridge, M. Hiraoka, and D. Delpy, “The finite element method for the propagation of light in scattering media: boundary and source conditions,” *Medical physics*, vol. 22, no. 11, pp. 1779–1792, 1995.

[79] S. L. Jacques, “Optical properties of biological tissues: a review,” *Physics in Medicine & Biology*, vol. 58, no. 11, p. R37, 2013.

[80] M. Geveler, D. Ribbrock, D. Göddeke, P. Zajac, and S. Turek, “Efficient finite element geometric multigrid solvers for unstructured grids on gpus,” in *Second Int. Conf. on Parallel, Distributed, Grid and Cloud Computing for Engineering*, p. 22, 2011.

[81] M. Freiberger, C. Clason, and H. Scharfetter, “Total variation regularization for nonlinear fluorescence tomography with an augmented lagrangian splitting approach,” *Applied optics*, vol. 49, no. 19, pp. 3741–3747, 2010.

[82] M. Guerquin-Kern, L. Lejeune, K. P. Pruessmann, and M. Unser, “Realistic analytical phantoms for parallel magnetic resonance imaging,” *IEEE Transactions on Medical Imaging*, vol. 31, no. 3, pp. 626–636, 2012.

[83] C. Geuzaine and J.-F. Remacle, “Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities,” *International Journal for Numerical Methods in Engineering*, vol. 79, no. 11, pp. 1309–1331, 2009.

[84] W. F. Mitchell and M. A. McClain, “A comparison of hp-adaptive strategies for elliptic partial differential equations,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 1, p. 2, 2014.

[85] W. F. Mitchell, “How high a degree is high enough for high order finite elements?,” *Procedia Computer Science*, vol. 51, pp. 246–255, 2015.

[86] M. Joulaian, S. Hubrich, and A. Düster, “Numerical integration of discontinuities on arbitrary domains based on moment fitting,” *Computational Mechanics*, vol. 57, no. 6, pp. 979–999, 2016.

[87] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pp. 1–12, IEEE, 2017.

[88] W. Qadeer, R. Hameed, O. Shacham, P. Venkatesan, C. Kozyrakis, and M. A. Horowitz, “Convolution engine: balancing efficiency & flexibility in specialized computing,” in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 24–35, ACM, 2013.

[89] P. Brož, S. M. Benito, C. Saw, P. Burger, H. Heider, M. Pfisterer, S. Marsch, W. Meier, and P. Hunziker, “Cell targeting by a generic receptor-targeted polymer nanocontainer platform,” *Journal of controlled release*, vol. 102, no. 2, pp. 475–488, 2005.

[90] K. Liu, X. Wang, V. Ntziachristos, S. Marsch, and P. Hunziker, “Polymeric nanosystems for near-infrared multispectral photoacoustic imaging: Synthesis, characterization and in vivo evaluation,” *European Polymer Journal*, vol. 88, pp. 713–723, 2017.