

DISSERTATION

Concepts for Scientific Computing

ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines Doktors der technischen Wissenschaften

eingereicht an der Technischen Universität Wien
Fakultät für Elektrotechnik und Informationstechnik
von

RENÉ HEINZL



Wien, im August 2007

Kurzfassung

Wissenschaftliches Rechnen beschäftigt sich bereits seit langer Zeit mit Konzepten bezüglich der Konvergenz von diskreten Näherungsverfahren von partiellen Differenzialgleichungen und deren effizienter Umsetzung in computerimplementierten Anwendungen. Die große Breite von physikalischen Modellen, welche z.B. in der Halbleiter-Bauelementmodellierung auftreten, und ihre unterschiedlichen numerischen Modellierungsarten zusammen mit der sehr stark schwankenden Komplexität der Modelle erschweren die Entwicklung von Simulationsanwendungen beträchtlich. Dazu wurde in den letzten Jahrzehnten eine Reihe von numerischen Lösungsverfahren entwickelt, um diese Modellierung in verschiedenen Gebieten so exakt wie möglich zu gestalten, und die effiziente Überführung in die digitale Welt des Computers zu ermöglichen.

Unterstützt wurde diese Entwicklung durch die stark steigende Rechenleistung moderner Computersysteme. Dadurch können zwar immer komplexere Modelle berechnet werden, jedoch wird schlussendlich wieder die komplette Rechenleistung benutzt. Deshalb ist es notwendig, auch im Anwendungsbereich die neuesten Konzepte und Techniken einzusetzen, um damit die Rechenzeiten auch für komplexe Modelle im Rahmen zu halten.

Diese Dissertation beschäftigt sich deshalb mit verschiedenen Konzepten für wissenschaftliches Rechnen und dem Umstand, dass bereits eine Vielzahl an Konzepten existiert. Jedoch sind bis zum heutigen Zeitpunkt weder zufriedenstellende und allgemeine Schnittstellen zur Anwendungsentwicklung noch einfach wiederverwendbare Module verfügbar, welche in einem breiten Maßstab eingesetzt werden können. Deshalb werden hier Konzepte gezeigt, welche sich auf die Umsetzung und Entwicklung von hochverfügbaren und performanten Komponenten zur Anwendungsentwicklung beziehen. Dazu wird eine allgemein topologische Schnittstelle für Datenstrukturen und eine eingebettete funktionale mathematische Spezifizierungssprache entwickelt. Diese Konzepte werden anschließend zu einer generischen Simulationsumgebung zusammengefügt und eine konsistente Basis von Algorithmen abgeleitet, welche im Prinzip auf jeder topologisch beschreibbaren Struktur arbeiten können. Dazu werden nicht nur die theoretischen Konzepte präsentiert, sondern auch die entsprechenden Programmierparadigmen gezeigt, welche die Anforderungen der Konzepte bestmöglich wiedergeben.

Der letzte Teil beschäftigt sich mit einer Auswahl an Anwendungen basierend auf verschiedenen numerischen Berechnungsmethoden. Diese sollen zeigen, dass die hier gezeigten Konzepte tatsächlich eine effiziente Umsetzung erlauben. Eine Analyse des Laufzeitverhaltens dieser Anwendungen zeigt anschließend, dass diese Konzepte nicht nur eine effiziente Beschreibung von mathematischen Modellen erlauben, sondern auch sehr leistungsstarke Anwendungen erzeugen, welche sogar manuell optimierte Anwendungen übertreffen.

Abstract

Scientific computing has traditionally been concerned with concepts related to numerical issues, such as the convergence of discrete approximations to partial differential equations and the computational efficiency of software implementations of numerical methods. The great diversity of physical models, e.g., semiconductor device models, makes the development of simulation applications extremely challenging. Each of the phenomena can be described by differential equations of varying complexity. The development of several numerical solution techniques has been necessary in order to best model the underlying physics and to accommodate the mathematical peculiarities of each of these equations when transferring them to the discrete world of digital computing. While computer performance is steadily increasing, the additional complexity of current mathematical models is easily outgrowing this gain in computational power. It is therefore of utmost importance to employ the latest techniques of software development to obtain high performance and to thereby ensure adequate simulation times, especially for complex problems.

Originating in the field of technology computer aided design, an important and complex area of scientific computing, this dissertation is motivated by the fact that though a great number of separate concepts have been developed during the last decades, up to now no common interfaces nor widely reusable modules for application design have emerged which can be used in a broad variety of areas. Thus the concepts for scientific computing given here are focused on components for generic and high performance library-centric application design. Therefore, this dissertation introduces a common generic data model and an embedded functional specification language to provide effective representations across a very wide range of application areas. Finally, a generic scientific simulation environment that operates on virtually any data is presented. Not only are generic concepts introduced, but also the corresponding programming paradigms which model the necessary requirements for an orthogonal extension and enhancement are expounded.

A concluding section presents various applications based on a wide variety of numerical calculation schemes and illustrates that the concepts demonstrated are indeed functional. It is also shown that even manually tuned high performance applications are outperformed by the generic scientific simulation environment.

Acknowledgments

Most of all I want to thank Prof. Selberherr for his support in many areas. The working environment, from the hardware to the various people at the Institute for Microelectronics, is more than fabulous. Most importantly I want to thank him for supporting my ideas from the beginning. I would like to thank Prof. Grasser for giving me the opportunity to work at the institute and for his support in various areas and for giving me the opportunity to learn a lot. I thank Prof. Purgathofer for his participation in the examining committee on such short notice.

My special thanks go to Philipp Schwaha who is one of the most wonderful and brilliant people I have ever worked with and from whom I have learned so much. I thank him for all the valuable discussions and for always believing in our work, for his company on all the travels around the world, and for making the hard days much easier. I also want to thank our former colleague Michael Spevak who introduced me to his world of mathematics and to his special view of different topics. Many parts of my own work were influenced by these two people, and so a deep gratitude will always be with you two. I want to express my appreciation to Andreas Hössinger who introduced me to the C++ world at the institute, which was the entry point to this work. Only chronologically last, I thank Franz Stimpfl for his support and fitting so well into the generic group at the institute.

My thanks also go to Hubert Enichlmayer and Rainer Minixhofer (AMS) for valuable insights, real world examples, and support from the company side; to Tatsumi-san and Kimura-san (Sony) for giving me the chance to work on the forefront of technology; to Peter Fleischmann, Steven Cea and Stefan Halama (Intel) for the great opportunity to work on the cutting edge and providing real examples; to Markus Schordan from the Institute for Computer Languages for valuable support in the field of compiler optimizations and insights into compiler technology.

I want to thank Werner Benger for introducing me to the fiber bundle concept and for being a great source of inspiration in so many areas, Yaakoub El-Khamra, and the rest of the group at the Center for Computation & Technology at Louisiana State University. Finally, I want to thank Joyce Visne for her never ending help and support.

And most importantly I want to thank my parents for giving me the chance to see and become who I am.

Contents

1	Introduction	1
1.1	Scientific Computing	1
1.2	Technology Computer Aided Design	2
1.3	Motivation	3
1.4	Organization	3
I	Theoretical Concepts	5
2	Mathematical Concepts	6
2.1	Introduction	6
2.2	Physical Fields and Electromagnetics	7
2.3	Topological Toolkit	11
2.3.1	Order Theory and Sets	13
2.3.2	Cell Topology	15
2.3.3	Complex Topology	16
2.4	Computational Topology	17
2.4.1	Chains	17
2.4.2	Cochains	19
2.4.3	Homology and Cohomology	20
2.4.4	Duality between Chains and Cochains	21
2.5	Fiber Bundles	22
2.6	Fiber Bundles and Chains/Cochains	27
2.7	Discrete Electromagnetics	29
3	Numerical Discretization Schemes	32
3.1	Generic Discretization Concepts	33
3.1.1	Domain Discretization	33
3.1.2	Topological Equations	34
3.1.3	Constitutive Relation Discretization	35

3.2	Finite Volumes	36
3.2.1	Basic Concepts	36
3.3	Finite Elements	39
3.3.1	Basic Concepts for a Galerkin Method	39
3.4	Finite Differences	42
3.4.1	Basic Concepts	42
3.4.2	Analysis of the Finite Difference Method	45
3.4.3	Further Analysis	46
 II Practical Concepts		49
4	Overview	50
5	Programming Concepts	52
5.1	Data Model for Scientific Computing	52
5.2	Evolution of Programming Paradigms	53
5.2.1	Object-Orientation Programming	55
5.2.2	Functional Programming	57
5.2.3	Generic Programming	58
5.2.4	Meta-Programming	59
5.3	Domain-Specific Embedded Language	60
5.4	Concept Development and Related Work	61
5.4.1	STAP, SCAP	62
5.4.2	Minimos-NT	62
5.4.3	AMIGOS, FEDOS	62
5.4.4	Wafer-State Server	63
5.4.5	Boost Graph Library	63
5.4.6	Computational Geometry Algorithm Library	63
5.4.7	Grid Algorithms Library	63
5.4.8	Further Related Work	64
6	Application Concepts	65
6.1	Generic Data Structures	65
6.2	Boundary Operation	69
6.3	Data Structure Storage Mechanism	70
6.4	Separation of Base and Fiber Space	71
6.5	Library-Centric Software Design	74

III	Applied Concepts	76
7	Overview	77
8	A Generic Scientific Simulation Environment	79
8.1	The Generic Topology Library	81
8.1.1	The 0-Cell Complex	83
8.1.2	The 1-Cell Complex	84
8.1.3	The 2-Cell Complex	85
8.1.4	Higher-Dimensional Cell Complexes	88
8.1.5	Data Access	89
8.2	The Generic Functor Library	90
8.2.1	Traversal Operators	90
8.2.2	Data Accessors	91
8.2.3	Arithmetic Functors	92
8.2.4	Domain-Specific Embedded Language	92
8.3	Additional Generic Components	94
8.3.1	Generic Solver Interface, GSI	94
8.3.2	Generic Orthogonal Range Queries	95
8.3.3	Generic File Interface, GFI	95
8.3.4	Finite Element Components	96
8.4	Performance Analysis	98
8.4.1	GTL's Performance	98
8.4.2	GFL's Performance	101
9	Generic Application Design	103
9.1	Visual Programming	103
9.2	Wave Equation	105
9.3	Diffusion Simulation	107
9.3.1	The Equations	107
9.3.2	Transformation into the GSSE	107
9.3.3	Results	108
9.4	Device Simulation	111
9.4.1	The Equations	111
9.4.2	Transformation into the GSSE	111
9.4.3	Results	112
10	Smart Analysis Package	116
10.1	Structure Modeling	116

10.2	Basic Equations and Discretization	118
10.2.1	Capacitance Analysis	118
10.2.2	Resistance Analysis	119
10.2.3	Finite Element Solution Procedure	119
10.2.4	Transformation into GSSE	120
10.3	Applications	121
10.3.1	Capacitance Analysis	121
10.3.2	Resistance Analysis for Cu-DD Architecture	123
10.4	Performance Analysis	125
10.5	Application Design Concepts	126
11	Summary and Outlook	128
A	Common Mathematical Terms	129
B	STL Iterator Analysis	132
C	Cell Properties of Discretization Schemes	134
	Bibliography	136
	Own Publications	146
	Curriculum Vitae	150

List of Tables

5.1	Language comparison.	60
6.1	Classification scheme based on the dimension of cells.	68
6.2	Classification scheme based on the dimension of cells and the cell topology.	68
6.3	Classification scheme based on the dimension of cells, the cell topology, and the complex topology.	69

List of Figures

2.1	Top: internal orientation. Bottom: external orientation for geometric objects in three dimensions.	8
2.2	Left: Integers form a chain, totally ordered by \leq . Middle: Incomparable items forming an anti-chain. Right: The power-set of $\{a, b, c\}$ ordered by \leq as a partially ordered set. . .	14
2.3	Cell topology of a simplex cell in two dimensions.	15
2.4	Cell topology of a cuboid cell in two dimensions.	15
2.5	Cell topology of a 3-simplex cell.	16
2.6	Complex topology of a simplex cell complex.	16
2.7	Complex topology of a cuboid cell complex.	16
2.8	Representation of a 1-chain with boundary (left) and a 2-chain with boundary (right). . . .	18
2.9	Cochain complex with the corresponding coboundary operator: $\mathcal{K}^1 \xrightarrow{\delta} \delta\mathcal{K}^1 \xrightarrow{\delta} \delta\delta\mathcal{K}^1 = 0$.	20
2.10	A graphical representation of (co)homology for a three-dimensional cell complex.	20
2.11	Illustration of cycles A, B, C and a boundary C . A, B are not boundaries.	21
2.12	Left: a fiber bundle with the homeomorphism f . Right: A homeomorphism into $U_b \times F$, which does not preserve the projection, thus not revealing a fiber bundle [1].	22
2.13	Zero section of a vector bundle [2].	23
2.14	A hierarchy of concepts with partial specialization. The most general form is represented by a sheaf concept. The concept of fiber bundles is obtained by using fibers with a certain dimension. If the fiber space satisfies linear vector space properties, the concept of a vector bundle is derived. Finally, by confining the dimension of the base and fiber space, a tangent bundle is obtained.	25
2.15	Illustration of the correlation of tangent space and cotangent space.	26
2.16	The identification of the concept of fiber bundles and the chain and cochain concept as dual spaces.	27
2.17	A representation of the intrinsic fiber bundles of the respective skeleton base spaces.	27
2.18	Overview of the mathematical concepts which have been introduced for the separation of a physical field domain into a base space and fiber space related to the concept of fiber bundles. The topological toolkit provides concepts to describe the internal structure of the cells of the cell complex to enable a generic data structure specification as well as generic traversal mechanisms. Concepts from computational topology such as chain and cochain complexes to obtain vector space structures within the tangent and cotangent space, are presented. Generic data access mechanisms are based on discrete field representations by cochains with their corresponding geometrical orientation and dimension.	31

3.1	Discretization concept for the primary and secondary cell complex with consistent orientation.	33
3.2	Time stepping for the discretization concepts.	34
3.3	Space-time stepping for the discretization concepts.	34
3.4	Finite volume requirements for a primary and secondary cell complex.	36
3.5	Primary control volumes used in the finite volume method. Left: cell-centered. Right: vertex-centered.	37
3.6	Finite volume and the rendered cell average value within each cell for a one-dimensional cell complex.	37
3.7	Geometrical interpretation of the basis of the expanded solution variable u_h for finite elements for a one-dimensional cell complex.	40
3.8	Finite difference requirements for the mesh.	42
3.9	Different geometric interpretations of the first-order finite difference approximation related to forward, backward, and central difference approximation.	43
3.10	Approximation of two-dimensional mixed derivatives.	44
3.11	Boundary treatment for the finite difference scheme. The necessary grid points are not available for all different types of approximation schemes.	44
5.1	A simplex representation of a data model and the corresponding interfaces.	52
5.2	Application design based on the object-oriented programming paradigm. Each of the inner blocks has to be implemented, resulting in an implementation effort of $\mathcal{O}(D \cdot A)$, where D stands for the number of data structures and A for the number of algorithms.	56
5.3	Application design based on the generic programming paradigm. Only the topmost and leftmost parts have to be implemented, which reduces the implementation effort to $\mathcal{O}(D + A)$	58
6.1	Complex topology of a singly linked list.	66
6.2	Complex topology of a doubly linked list.	66
6.3	Complex topology of a tree.	66
6.4	Complex topology of an array.	67
6.5	Complex topology of a graph.	67
6.6	Local cell complex (left), normally called mesh, and a 2-cell representation (right). Vertices are marked with filled circles.	67
6.7	Global cell complex (left), normally called grid, and a 2-cell representation (right).	68
6.8	Boundary operator applied onto a 3-simplex poset.	70
6.9	Connection matrix C_j^i for a cell complex.	71
6.10	Left: a fiber bundle over a 0-cell complex. Right: a fiber bundle over a 2-cell complex.	72
6.11	Illustration of an index space depth of zero (left) and one (right). The base space is modeled by a 0-cell complex.	73
8.1	Building blocks of the GSSE.	80

8.2	The generic topology library of the GSSE.	81
8.3	Example of the connection matrix C_j^i for a 3-simplex cell complex.	82
8.4	Representation of a 0-cell complex with a topological structure equivalent to a standard container.	83
8.5	Representation of a 1-cell complex with cells (edges, C) and vertices (V).	84
8.6	Incidence relation and traversal operation.	85
8.7	The cell poset of a 4-simplex as implemented by the GTL.	88
8.8	Renderings of a 4-simplex (left) and a 4-cube (right).	88
8.9	The generic functor library of the GSSE.	90
8.10	Traversal methods induced by the incidence relation. The rows illustrate traversal schemes of the same base element, whereas columns depict traversal schemes of the same traversal element.	91
8.11	Topological traversal of vertices.	98
8.12	0-cell traversal on the P4 (left) and the AMD64 (right); the units are iterations per second.	99
8.13	0-cell traversal on the G5 (left) and the IBM (right); the units are iterations per second	99
8.14	Topological traversal of cells for a one-dimensional cell complex.	100
8.15	Incidence traversal for the BGL and the GTL approach on the P4 (left) and the AMD64 (right); the units are iterations per second	100
8.16	Incidence traversal for the BGL and the GTL approach on the G5 (left) and the IBM (right); ; the units are iterations per second	100
8.17	Evolution of compiler enhancements on the P4 for the DAXPY benchmark. Left: GCC 4.0.4; middle: GCC 4.1.2; right: GCC 4.2.1	101
8.18	Evolution of compiler enhancements on the AMD X2 for the DAXPY benchmark. Left: GCC 4.0.4; middle: GCC 4.1.2; right: GCC 4.2.1	101
8.19	Best compiler performance for the Intel Core (left) and the AMD X2 (right), both using the GCC 4.2.1.	102
8.20	Best compiler performance for the G5 (left) and the IBM (right), both using the system compiler.	102
9.1	Potential of a PN diode during different stages of the Newton iteration. From initial (left) to the final result (right).	104
9.2	Visualization of non-converging process. Here the potential is illustrated where small oscillations can be observed from left to right.	104
9.3	Illustration of a complete breakdown of the solution procedure (from left to right).	104
9.4	The left figure depicts Faraday's law by the corresponding projection onto a finite cell, whereas the right figure illustrates Amperé's law.	105
9.5	Illustration of the x -component of \mathbf{E} with a harmonic oscillating source in the x - y plane at two different time steps.	106
9.6	Wave equation with a harmonic oscillating source in the x - y plane where the source is switched of (left figure). The y -component of \mathbf{E} is depicted on the right side.	106
9.7	Two-dimensional structured grid with an initial doping profile (concentration in cm^{-1}).	108

9.8	Two-dimensional diffusion simulation for a structured grid after 200 time steps.	109
9.9	Two-dimensional diffusion simulation for a structured grid after 2000 time steps.	109
9.10	Three-dimensional device structure with an initial phosphorus doping profile (concentration in cm^{-1}).	110
9.11	Three-dimensional diffusion simulation for a device structure with an initial doping profile. Two subsequent simulation steps are depicted.	110
9.12	Domain for a two-dimensional PN-junction diode.	113
9.13	Netto doping concentration of the two-dimensional PN diode. Donors are given in red, acceptors are blue; units are given in parts per cubic meter.	113
9.14	Two-dimensional diode with potential distribution for equilibrium mode; units given in Volt.	114
9.15	Two-dimensional diode with potential distribution for reverse mode; units given in Volt.	114
9.16	Two-dimensional diode with potential distribution for forward mode; units given in Volt.	114
9.17	Two-dimensional diode with charge distribution for reverse mode; units given in parts per cubic meter.	115
9.18	Two-dimensional diode with charge distribution for forward mode; units given in parts per cubic meter.	115
10.1	A comparison between the LAYGRID and VGM syntax.	117
10.2	Resulting structure from the given input specification.	117
10.3	A structure suitable for capacitance simulation, created by VGM.	121
10.4	The offset of the second layer of contact 1. Right: the corresponding mesh.	121
10.5	Illustration of isosurfaces of the potential distribution.	122
10.6	Interconnect structure for resistance analysis.	123
10.7	Interconnect structure for resistance analysis.	123
10.8	A comparison between the LAYGRID and VGM syntax.	124
10.9	Comparison of the structures created and the corresponding meshes for LAYGRID (left) and VGM (right). With the VGM approach different ratios between the interconnect line (red) and the covering layer (blue) can be easily modeled. Here a ratio between the thickness of the interconnect line and the covering layer is given by $1/10000$. As can be seen, the thin layer does not impose additional points for the interconnect line.	124
10.10	Resistance progression related to the spatial expansion of the via part.	125
10.11	Temperature distribution due to self-heating in a tapered interconnect line with cylindrical vias.	125
10.12	Comparison of the finite element assembly times in equations per millisecond, GSSE and SAP.	126
A.1	A relation R of a set A and a set B	129
A.2	The image of a under f	130
A.3	The kernel of f	131

C.1 Geometrical interpretation of the basis functions for the (left) finite volume method and (right) for the finite element method. The finite volume method describes consistent crisp cells which can be interpreted as a cell complex, whereas the finite element method uses spread cells which do not conform with the properties of a consistent cell complex. 135

Notation

Due to the wide variety of material in this work drawn from different fields in scientific computing, some conflicts or ambiguities in the notation cannot be avoided. The following table summarized the used notations:

\mathbb{D}^n	n -dimensional closed unit ball $\{x \in \mathbb{R}^n \mid \ x\ \leq 1\}$
\mathbb{S}^n	n -dimensional unit sphere $\{x \in \mathbb{R}^{n+1} \mid \ x\ = 1\}$
$\langle a, b \rangle$	scalar product ς of a, b
\mathbb{F}	field, e.g., \mathbb{R}
\mathcal{V}	vector space
\mathcal{V}^*	dual vector space
(X, \mathcal{T})	topological space
\mathfrak{M}	manifold
Ω	domain, subset of \mathbb{R}^n
Γ	boundary of a domain $\partial\Omega$
\mathfrak{K}	cell complex
\mathcal{L}	linear operator
$\mathcal{T}_{\mathcal{P}}$	tangent space
$\mathcal{T}_{\mathcal{P}}^*$	cotangent space
O	(globally) ordered set
P	partially ordered set (poset)
$(B \times F, B, pr_1)$	trivial fiber bundle
(F, B, pr_1)	fiber bundle
$\mathcal{T}(\mathfrak{M})$	tangent bundle
C_j^i	connection matrix
τ_p^i	i -th p -cell
c_p	p -chain
\mathcal{C}^p	p -cochain
C_*	chain complex
C^*	cochain complex
∂	boundary operator, partial derivative
δ	coboundary operator
E	electric field
H	magnetic field
D	electric flux
B	magnetic flux
J	current flux density vector
A	magnetic vector potential
ε	dielectric permittivity
μ	magnetic permeability
ϱ	electrical charge density
ϕ	basis function for finite elements
u	generic solution quantity

Chapter 1

Introduction

Scientific computing, a common term for various tasks in computational science, is concerned with concepts which close the gap between theoretical and experimental science. This development is supported by the exponential advancement of the brute force number-crunching possibilities of modern computer systems, which have proved Moore's prediction in 1965 that computational power would double every 18 months.

Hidden in the shadow of the number crunching, various other paradigms have emerged since then, especially in the field of scientific computing. One of them can be attributed to the rigorous shift of the interaction between users and computer systems. In the early days of computer operation, a stack of cards had to be used to access the resources available. Through the advent of terminals and the even more drastic paradigm change of multi-terminals on one screen, the possibilities of computing have completely changed. Nowadays we can connect interactively from almost any location in any part of the world to our computer systems. Work stations even offer complete virtual realities with real-time calculation and multi-dimensional visualization capabilities. However, what can still be observed is that software is implemented repeatedly in almost the same way as it was decades ago, sometimes with only minor adaptations or a small change in the paradigms used. This repetition is also noticeable in the area of computer languages, with the development of a great number of different languages, all of which are currently not sufficient for the tasks of scientific computing [3–7] with the exception of a few outstanding developments [8, 9].

The Institute for Microelectronics has a long history in the field of Technology Computer-Aided Design (TCAD), the process of gaining insights into procedures and mechanisms in the area of semiconductor manufacturing and advancement. Since this history goes back to the roots of computer simulation in the field of semiconductor research, the institute is able to track the various changes in the field of computational science. Currently the scientific community, and even more the TCAD community, relies on either domain-specific closed applications or on components tied very closely to particular representations which were often built for very specific purposes and then tossed into a huge and monolithic toolkit. To this date, data structures and algorithms are implemented in a heavily application-specific way, making their reuse practically impossible. From an engineering perspective, interactions between theoretical scientific computing concepts and their effective and efficient practical application are required, thereby shifting the focus to high performance, library-centric application design.

1.1 Scientific Computing

Scientific computing is understood here as the collection of concepts which deal with large equation systems by utilizing discretized partial differential equations from different fields of physics as well as

the computational efficiency of software implementations of these numerical methods. Different types of partial differential equations and their discretization schemes, such as the finite element method or the finite volume method, have to be considered for diverse types of problems. Types of topological cell complexes, different dimensions, and various solving strategies, all with their appropriate algorithmic description, have to be considered during application development.

The development of software for the numerical simulation of disparate physical phenomena, such as electromagnetic wave propagation, heat transfer, mechanical deformation, fluid flow, and quantum effects, is not straight-forward and is even today very complex. Early software in the field of scientific computing consisted of a monolithic single application to deal with special problems. Due to the direct solution process, these applications perform exceptionally well, as they are written by domain experts and tuned manually. At this time, the domain expert, software designer, programmer, tester, and end-user is one person, a situation which complicates and slows down the complete development process. The reuse of components or the extension of such an application written by a domain expert usually requires a large investment of time to get accustomed to the special domain and the internal mechanisms of the application. The field becomes more complex, if couplings of various phenomena and different discretization schemes are used to obtain a solution. Each of these schemes has its merits and shortcomings and is more or less suited for different classes of equations. Therefore a multitude of software applications and tools, which provide methods and libraries for the solution of very specific problem classes, has been developed. However, they are mostly specialized for a certain type of underlying mathematical model, resulting in a solution process which is highly predictable. Only in the recent past have environments for various problems at hand been developed and published, all with their advantages and disadvantages. However, it is important to note that applications not developed with interoperability in mind impose restrictions on possible solution methods which can not be foreseen at the beginning of program development.

1.2 Technology Computer Aided Design

TCAD is the application of computational methods and software tools for the analysis and design of integrated semiconductor devices and their fabrication processes. The computational methods deal with the assembly of large equation systems by utilizing discretized partial differential equations from different fields of physics. Special constraints related to high flexibility for the modeling are imposed [10]. The great diversity of physical phenomena present in semiconductor devices themselves, and in the processes involved in their manufacture, make the field of TCAD extremely challenging. Each of the phenomena can usually be described by differential equations of varying complexity. The development of several different discretization schemes has been necessary in order to best model the underlying physics and to accommodate the mathematical peculiarities of each of these equations while transferring them to the discrete world of digital computing.

With the steady evolution of software tools and languages, the shift of programming paradigms can be clearly observed. During its first years our institute used only one- and two-dimensional data structures, due to the limitations of computer resources. The imperative programming paradigm was sufficient for this type of task [11]. With the improvement of computer hardware and the rise of the object-oriented programming paradigm, the shift to more complex data structures was possible. The development of complex applications for process [12] and device [13] simulation is an example of this evolutionary path. More complexity is added when modeling requires a change in the underlying topological data structure, usually from structured grids to unstructured meshes; the use of alternative linear or nonlinear solver mechanisms [14, 15]; different types of quantities, e.g., vectorial quantities; or a change in the discretization scheme. Not only does application development in TCAD require the utilization of up-to-date data structures and algorithms, but it also demands an overall high performance.

1.3 Motivation

The motivation for reviewing and developing a variety of concepts for scientific computing is derived from the need for flexibility in high performance applications in the field of scientific computing, especially in TCAD. With a growing number of simulation tools and various requirements on the underlying data structures, the question arises: which part of an application can be reused if it is properly implemented. This dissertation therefore introduces, analyzes, and unifies not only theoretical but also practical concepts. Different programming paradigms and typically used idioms common to many different kinds of applications in the field of scientific computing are introduced. The nature of dealing with mathematical models, with the inherent coupling of topological traversal and functional description, complicates the use of libraries which are not designed in an orthogonal library-centric way. Significant advances have been made here since the advent of the C++ standard template library (STL [16, 17]) or the Java library collection [6]. The effort of application development is thereby greatly reduced. Particularly the formal and abstract interface specification of the generic programming approach, implemented by means of static parametric polymorphism in C++, offers a formal way of specification to the user, which the compiler can verify. Reusability and orthogonality of the libraries is thereby accomplished without sacrificing overall performance. This is a necessary constraint because the additional complexity of simulation models easily outgrows the steadily increasing computational power. The overview given in Section 5.4 shows that, so far, no approach can efficiently deal with the mathematical formulation of physical phenomena within a programming language.

The concepts presented here also demonstrate how to deal with problems arising in scientific computing. The method of programming applications and then patching them to fit the requirements of examples is becoming more and more of a handicap. The required models, physical processes, and simulation methodologies are expanding in an almost exponential manner. And as previous decades have shown, the software engineering community has lagged behind most of the time.

Therefore, this work does not introduce another general-purpose kernel but instead provides and develops a comprehensive set of generic components and algorithms for scientific computing. The two basic mechanisms are, on the one hand, a generic data structure specification language and, on the other hand, a functional equation specification language. Based on these building blocks, the complete range of application design, from small generic components to complete applications, is possible. The focus of developing an application can be shifted to identifying generic components, using already existing modules, and implementing the remaining functionality.

A possible implementation is presented here by the generic scientific simulation environment, which exhibits formally defined interfaces as well as an overall high performance comparable to all manually tuned applications including classical high performance Fortran applications.

1.4 Organization

Part I is based on the gradual shift of several concepts into an abstract form for scientific computing. The starting point in Section 2.2 is given by a brief review of basic treatment of physical fields and in particular Maxwell's equations. The thereby given continuous formulations are then the basic motivation to introduce concepts and mechanism for the transition to the finite regime of computational methods. A basic topological toolkit (Section 2.3) is therefore introduced to enable structural properties of discrete spaces and is based on the work of Jänich [2] for common topological terms, Berti [18] for cell complex theory, and Zomorodian [19] for the computational topology (Section 2.4). Concepts of fiber bundles (Section 2.5) are adopted to enable a separation for the concepts given based on the work of Butler [20, 21]. Algebraic topology [22] and differential forms [23–25] are established to analyze various discretization

schemes [26–30], given in Section 3, to numerically deal with the given continuous problems. The discrete mapping is then given by concepts related to chains and cochains related to physical field theories and follows mainly the works of Tonti [24, 31, 32], Mattiussi [33–36], and Hyman [37]. Furthermore, the formal handling of field theory and the application of algebraic topology to identify the underlying space describe the nature of different discretization methods.

Part II deals with the transformation of the presented theoretical concepts into generic and reusable software components. An important step, the C++ STL, is presented and analyzed as the beginning of a systematic development towards a reusable software component catalog [38]. Various programming paradigms are presented in Section 5.2 and their related advantages and disadvantages [39–45] and performance issues [46] are illustrated. To enable efficient algorithm specification, concepts for domain-specific languages [45] are introduced in Section 5.3, whereas the focus of application design is shifted to the field of active libraries [47–50]. Section 5.4 reviews various works and assembles a set of requirements for generic software components. A topological generalization of data structures and a related common specification mechanism is then developed and introduced in Section 6. This part is heavily influenced by the works of Berti [18, 51]. Utilizing the fiber bundle model a data model for application design is developed, based on concepts from scientific visualization [1] but with a special focus on application design. Finally, the remainder of Section 6 identifies necessary requirements for generic library-centric application design in the field of scientific computing.

Based on these theoretical concepts and practical concepts, Part III introduces a set of generic components that operate on any data represented in the given models, represented by a generic scientific simulation environment, given in Section 8. A performance analysis is presented in Section 8.4 to highlight that multi-paradigm library-centric application design can easily compete with highly specialized applications. Concept-based programming and iterative refinement of algorithms is inspired by the work of Stepanov [16]. Various applications present the applicability and feasibility of the presented approach of the generic scientific simulation environment, given in Section 9 and Section 10.

Part I

Theoretical Concepts

Chapter 2

Mathematical Concepts

2.1 Introduction

One of the most fundamental concepts in scientific computing is the numerical and discrete treatment of the transformation of physical field problems into mathematical problems and then into the finite and discrete area of computational feasible algorithms and data structures. The reason for the following sections and definitions is therefore to extract the relevant concepts for scientific computing based on mathematical concepts to provide:

- a proven foundation
- a roadmap for application design
- great flexibility and longevity of resulting software

The foundation and roadmap for application design are initiated by the separation of two important concepts based on the topic of algebraic topology. First, space or, in a more general way, space-time has to be discretized into simple computable blocks, e.g., into a collection of cells describable by the theory of cell complexes and their topological structure. An algebraic description is then presented by the concept of chains. Second, concepts from (co)homology are introduced to handle the different types of physical quantities with their respective dimension and orientation by algebraic structures and thus make computer resources available for simulation. Selected concepts from topology and fiber bundles are then applied to specify the transition between continuous space and the concept of differential forms and the finite regime of computer simulation in a formal way. Topology, chains, and the fiber model, which are introduced in this chapter, are axiomatic studies. This results in the fact that a large number of definitions is required.

The contribution of this section is the preparation of basic formal mechanisms for the methodology of discretization schemes introduced subsequently. Structural information and guidelines for applied numerical analysis can thus be obtained. The result of this chapter is a foundation for a formal language used in the following chapters, which builds a framework to introduce the common concepts hidden under the surface of a variety of physical and mathematical modeling concepts, commonly represented by equations of different types, thereby enabling new possibilities for specification and optimization of developed algorithms.

2.2 Physical Fields and Electromagnetics

This section is dedicated to motivate the subsequent theoretical concepts, introducing a mapping of physical fields to a concise computational framework. One of the fundamental concepts of mathematical physics is that of a field, a spatial distribution of mathematical objects representing a physical quantity [35]. Most of the currently known physical theories build on a common structure [31] emphasizing the following:

- The existence of a natural association of many physical quantities with geometric objects in space-time.
- The necessity to consider the geometric object to which physical quantities are associated as oriented.
- The existence of two kinds of orientation for these geometric objects.
- The primacy and priority of global physical quantities associated with geometric objects over the corresponding densities.

Therefore, it is necessary to equip the inherently discrete parts of a computer implementation with as much information as possible about the continuous to be projected physical entity. Thereby not only the dimension but also the orientation and information regarding a primary and secondary quantity have to be transferred accordingly.

Here, Maxwell's equations [52] are briefly introduced, which are extraordinarily important in a special field of scientific computing. It was shown [24, 31] that most of the physical theories can be treated similarly, sometimes called the Tonti-diagram, due to the intrinsic nature and structure of the physical concepts. Here, the classic theory of electromagnetics and the macroscopic electric and magnetic phenomena including their interactions are introduced. The following four space-time dependent vector fields are considered:

- the electric field denoted by \mathbf{E} [V/m]
- the magnetic field denoted by \mathbf{H} [A/m]
- the electric flux \mathbf{D} [As/m^2]
- the magnetic flux \mathbf{B} [Vs/m^2]

The sources of electromagnetic fields are electric charges and currents described by

- the charge density ϱ [As/m^3]
- the current density \mathbf{J} [A/m^2]

where the SI units denote meter m , seconds s , Amperé A , and Volt V . The equations related to electromagnetics in integral formulation read:

$$\int_{\partial S} \mathbf{E} \cdot d\mathbf{l} = -\frac{d}{dt} \int_S \mathbf{B} \cdot d\mathbf{S} \quad \text{Faraday's law} \quad (2.1)$$

$$\int_{\partial V} \mathbf{B} \cdot d\mathbf{S} = 0 \quad \text{Gauß's law for magnetic charge} \quad (2.2)$$

$$\int_{\partial S} \mathbf{H} \cdot d\mathbf{l} = \frac{d}{dt} \int_S \mathbf{D} \cdot d\mathbf{S} + \int_S \mathbf{J} \cdot d\mathbf{S} \quad \text{Amperé's law} \quad (2.3)$$

$$\int_{\partial V} \mathbf{D} \cdot d\mathbf{S} = \int_V \varrho dV \quad \text{Gauß's law for electric charge} \quad (2.4)$$

where V represents a volume, S a surface, and l a line. If S is the boundary of a volume V in Amperé's law then the boundary of S vanishes ($\partial\partial V = 0$). The field vector can then be eliminated from Amperé's and Gauß's law to obtain a statement about charge conservation:

$$0 = \frac{d}{dt} \int_V \varrho dV + \int_{\partial V} \mathbf{J} \cdot d\mathbf{S} \quad (2.5)$$

If the surfaces and volumes are stationary with respect to the inertial reference frame of field vectors, the local differential versions of the Maxwell equations can be used:

$$\text{curl}(\mathbf{E}) = -\partial_t \mathbf{B} \quad (2.6)$$

$$\text{div}(\mathbf{B}) = 0 \quad (2.7)$$

$$\text{curl}(\mathbf{H}) = \partial_t \mathbf{D} + \mathbf{J} \quad (2.8)$$

$$\text{div}(\mathbf{D}) = \rho \quad (2.9)$$

The differential versions of Maxwell equations are much less general than the integral laws. The differential laws assume differentiability of the field vectors with respect to spatial coordinates and time. However, there can be discontinuities in fields at material interfaces, so that it is not possible to deduce the proper interface conditions from the differential laws. Therefore, the global integral laws have to be used to derive interface conditions which give relations for the fields at material or media interfaces:

$$\mathbf{n} \times (\mathbf{E}^a - \mathbf{E}^b) = 0, \quad (2.10)$$

$$\mathbf{n} \cdot (\mathbf{B}^a - \mathbf{B}^b) = 0, \quad (2.11)$$

$$\mathbf{n} \times (\mathbf{H}^a - \mathbf{H}^b) = \mathbf{K} \quad (2.12)$$

$$\mathbf{n} \cdot (\mathbf{D}^a - \mathbf{D}^b) = \sigma_s \quad (2.13)$$

where the superscripts refer to values of the field variables corresponding to one of the sides of the interface. It is important to highlight that the source quantities, e.g., \mathbf{K} , do not really exist, but rather represent a way of modeling current distributions in the limit of zero skin depth [53].

The integral formulation contains geometric dimensional attributes such as volume or line objects, where the corresponding physical quantities can be calculated. Also, this formulation contains additional topological information, such that an area has to be the surface of a given volume. This information is not given by the differential expression. For the transition to the finite regime of the computer additional information has to be processed to maintain the given structure of physical fields.

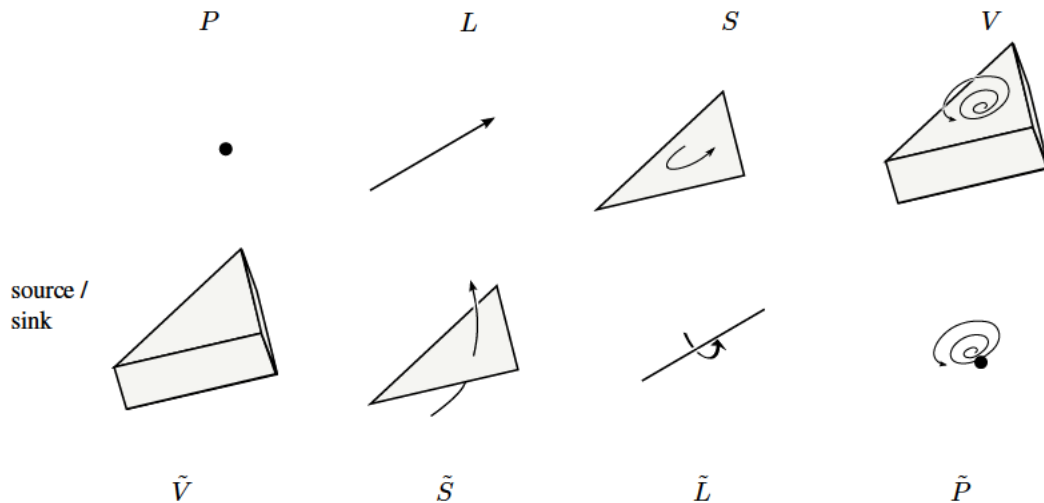


Figure 2.1: Top: internal orientation. Bottom: external orientation for geometric objects in three dimensions.

Both of these formulations do not contain the important information of the given orientation of the underlying geometrical objects, which is required to assign signs to the given quantities. An example regarding this orientation is the direction of \mathbf{D} through an area, a so-called external orientation, because the direction arrow used is not part of the area itself. Another example is the electric field associated with an internally

oriented line segment, which means that the direction arrow is directly associated with the object under consideration. See Figure 2.1 for an overview, where the top line depicts internally oriented objects and the bottom line shows the externally oriented objects with their corresponding dimension. P states a zero-dimensional object, a point with a source sink orientation, L a one-dimensional object (line) with a direction, S a two-dimensional object (area) with a circulation direction, and finally rotation direction for a three-dimensional object (volume) V . The corresponding externally oriented objects are given by $\tilde{V}, \tilde{S}, \tilde{L}, \tilde{P}$. As can be seen, the internal orientation can be described by directions within the given objects, whereas the external orientation requires an additional direction outside the object.

The transition of this modeling step to the regime of the computer demands an additional approximation, the reformulation of the given integral problem in discrete terms with additional orientation information. Based on the given configuration of geometrical objects, the following table associates each physical quantity with the corresponding dimension, orientation, and time attribute. In the following table, $L \times I$ states that the line L is actually multiplied by a time interval I , where T means discrete time frames.

dimension	physical quantity	internally oriented object	externally oriented object
1D	$\mathbf{E} \rightarrow (L \times I)$	line	
2D	$\mathbf{B} \rightarrow (S \times T)$	area	
2D	$\mathbf{J} \rightarrow (\tilde{S} \times I)$		area
3D	$\varrho \rightarrow (\tilde{V} \times T)$		volume
1D	$\mathbf{H} \rightarrow (\tilde{L} \times I)$		line
2D	$\mathbf{D} \rightarrow (\tilde{S} \times T)$		area
0D	$\varphi \rightarrow (P \times I)$	point	
1D	$\mathbf{A} \rightarrow (L \times T)$	line	

The necessity of a primary and secondary cell complex can also be easily seen here. The requirement of housing the global physical quantities of a problem implies that both objects with internal orientation and objects with external orientation must be available. Hence, two logically distinct meshes must be defined, one with internal and the other with external orientation.

For general field problems the given Maxwell's equations are not sufficient to determine the electromagnetic field since there are six independent equations in twelve unknowns $\mathbf{E}, \mathbf{B}, \mathbf{D}, \mathbf{H}$. To obtain a consistent equation system, so-called constitutive laws have to be introduced additionally. Then, the given system of equations can be identified by two different classes, whereas the first class represents the structure of a given physical problem [32].

Definition 1 (Structural law) *Structural laws are conservation, balance, and equilibrium laws.*

They state a balance of global quantities whose validity does not depend on metrical or material properties and is therefore invariant for very general transformations. This gives them topological significance and the name topological equation was used [33] to express the idea of invariance under arbitrary transformations. The second class represents a constitutive law which acts as link between different orientations of a structural law [31].

Definition 2 (Constitutive law) *Constitutive laws link local field representations.*

For the simple case of linear isotropic media, the constitute laws are given by:

$$\mathbf{D} = \varepsilon \mathbf{E} \tag{2.14}$$

$$\mathbf{B} = \mu \mathbf{H} \tag{2.15}$$

Unlike topological laws, constitutive law representations imply the recourse to metrical concepts. This is not apparent in Equation 2.14 because of the use of vectors to represent field quantities, which tends to hide the geometric details of the theory. This leads to the important distinction of topological laws which are intrinsically discrete [24] and constitutive relations which admit only approximate discrete renderings. For a topological equation the discrete or global version appears as the fundamental one, with the differential statement resulting from it if additional requirements are fulfilled.

The final step towards a complete discrete representation is the association of all given quantities by their corresponding geometrical object. Starting with the global expression

$$\int_{\partial V} \mathbf{B} \cdot d\mathbf{S} = 0 \quad (2.16)$$

which can be written with the complete dimensional information as:

$$\int_T \int_{\partial V} \mathbf{B} = \int_T \int_V 0 \quad (2.17)$$

This expression can be stated by global quantities only and in a four-dimensional space-time where also a four-dimensional geometrical depiction can be given, where Φ^2 represents the associated quantity on a $\partial V \times T$ object:

$$\Phi^2(\partial V \times T) = 0(V \times T) \quad (2.18)$$

It must be mentioned that this expression does not use any material properties and is therefore not an approximation, hence a representation of physical quantities by their intrinsic discrete nature. As an introductory example and a static time frame the expression can be rewritten as:

$$\Phi^2(\partial V) = 0(V) \quad (2.19)$$

By using a, e.g., three-dimensional cell τ_3 , and the corresponding boundary of this cell $\partial\tau_3$, the expression yields:

$$\Phi^2(\partial\tau_3) = 0(\tau_3) \quad (2.20)$$

To highlight the relation between the cell and the physical quantity, the following pairing $\langle \cdot, \cdot \rangle$ can be stated:

$$\langle \partial\tau_3, \Phi^2 \rangle = \langle \tau_3, \mathbf{0}^3 \rangle \quad (2.21)$$

which lists equivalences of global physical quantities, but still compares to different dimensional quantities, the two-dimensional boundary of a cell $\partial\tau_3$ and the associated Φ^2 , and the three-dimensional $\mathbf{0}^3$ on a three-dimensional cell τ_3 . Another operator, the so-called coboundary operator acts similarly to the boundary operator and can then be used to obtain the following expression:

$$\langle \tau_3, \delta\Phi^2 \rangle = \langle \tau_3, \mathbf{0}^3 \rangle \quad (2.22)$$

As it can be seen, the geometrical dimensions of the given cells correspond to each other, and the final expression states the identity of the two quantities $\delta\Phi^2 = \mathbf{0}^3$.

The following sections now introduce the necessary theoretical concepts to develop a complete and concise framework for this topic. On the one hand, the topological and geometrical concepts are given to handle the underlying discretization of space. On the other hand, concepts required for handling physical quantities and their corresponding operations are introduced. The final goal of this section is to transfer the given physical quantities to the finite regime of the computer without loss of information regarding their dimension, orientation, and pairing with the corresponding geometrical object.

2.3 Topological Toolkit

To model space and time within the regime of a finite computer representation, the continuous domain has to be projected onto finite domains or cells. This chapter describes how these cells can be introduced and manipulated algebraically¹, and is mainly based on the work of Jänich [2] for common topological terms, Berti [18] for the cell complexes, and Zomorodian [19] for the computational topology part. The most basic property of topology is that it separates global space properties from local geometric attributes. Additionally, and more importantly for this work, it provides a precise notation and language for discussing and handling various properties.

Topological spaces offer several operations for sets and subsets. The formal definition of a topological space is given next, where only the concept of a set is implied:

Definition 3 (Topology) *A topological space (X, \mathcal{T}) consists of a set X and a set \mathcal{T} of subsets, called open sets, of X such that:*

- (T1) $\emptyset \in \mathcal{T}$ and $X \in \mathcal{T}$.
- (T2) a finite intersection of members of \mathcal{T} is in \mathcal{T} .
- (T3) an arbitrary union of members of \mathcal{T} is in \mathcal{T} .

The second property (T2) states that arbitrary intersections of subsets have to be in the topological space again. Also the union (T3) of subsets has to be contained in the space. These properties are later used to describe inter-dimensional elements for the complex, such as edges of a cell. An example is given in the following which presents a basic set $X = \{a, b, c\}$ and the corresponding topology. This pair of the original set and the set of subsets generates the topological space. Vertices are modeled by the singletons $\{a\}$, $\{b\}$, and $\{c\}$.

$$(X, \mathcal{T}) = \{\emptyset, \{a\}, \{b\}, \{c\}, \\ \{a, b\}, \{a, c\}, \{b, c\}, \\ \{a, b, c\}\}$$

To handle all subsets in a concise way, the concept of a subspace is introduced.

Definition 4 (Subspaces) *Let (X, \mathcal{T}) be a topological space. Any subset Y of X inherits a topology in a natural way. It is given by:*

$$\mathcal{T}_Y := \{V \subseteq Y \mid V = U \cap Y \text{ for some } U \in \mathcal{T}\} \quad (2.23)$$

To introduce the concept of a topological base, it is necessary to create a topology on a set X in which a set \mathcal{S} of subsets of X are open sets.

Definition 5 (Bases) *If \mathcal{S} is already closed under finite intersections, then \mathcal{T} can be defined to be those sets which are unions of sets in \mathcal{S} . Then \mathcal{T} satisfies (T1), (T2), and (T3) and \mathcal{S} is said to be a basis for \mathcal{T} .*

In general, to obtain a topology containing \mathcal{S} one has to first form \mathcal{B} , the set of sets which are finite intersections of members of \mathcal{S} , and then define \mathcal{T} to be all arbitrary unions of members of \mathcal{B} . In this case \mathcal{S} is called a sub-basis for \mathcal{T} .

Definition 6 (Homeomorphism) *A homeomorphism $f : X \rightarrow Y$ is a 1 – 1 onto function, such that both f, f^{-1} are continuous.*

¹Appendix A reviews necessary common terms.

Then, X is homeomorphic to Y , $X \approx Y$. This means that X and Y have the same topological type. Homeomorphisms are topological isomorphisms.

With these definitions, sets of subsets can be handled with additional properties in a common way. The common definition for a topological space is very general and allows several topological spaces which are not useful in the field of data structures, e.g., a topological space (X, \mathcal{T}) with a trivial topology $\mathcal{T} = \{\emptyset, X\}$. Therefore the basic mechanism of separation within a topological space is introduced. Of a hierarchy of possible separation conditions augmenting the topological space axioms, an important characteristic is the Hausdorff condition.

Definition 7 (Hausdorff spaces) *The topological space (X, \mathcal{T}) is said to be Hausdorff if, given $x, y \in X$ with $x \neq y$, there exist open sets U_1, U_2 such that $x \in U_1, y \in U_2$ and $U_1 \cap U_2 = \emptyset$.*

Common data structures in the field of scientific computing embody the separation characteristics of a Hausdorff space. The assumption to be made is that all scientific data can be modeled by the concept of a Hausdorff space. This can be seen as a generalization of Butler's model, which is based on the assumption that all scientific data can be modeled by trivial fiber bundles [21].

The concept of a cover is introduced to equip a topological space with a type of dimension.

Definition 8 (Cover) *A cover of a set X is a set of nonempty subsets of X whose union is X .*

A cover is an open cover if it is contained in the topology.

Definition 9 (Covering Dimension) *A topological space has a (Lebesgue) covering dimension p if any open cover C has a second open cover D , the refinement, where each element $d \in D$ is a subset of an element in the first cover such that no point is included in more than $p + 1$ elements.*

If a topological space is homeomorphic to \mathbb{R}^p with $p \in \mathbb{N}$, then its dimension is p . If a space does not have a Lebesgue covering dimension p for any $p \in \mathbb{N}$, it is called infinite-dimensional. The dimension of the empty set \emptyset is defined as -1 . It is important to note that the reverse is not always true, which means that a topological space with dimension p is not necessarily homeomorphic to \mathbb{R}^p .

Another fundamental topological concept is compactness, which may be regarded as a substitute for finiteness. It frequently compensates for the restriction to finite intersections in axiom (T2) by allowing arbitrary sets of open sets to be reduced to finite sets of sets.

Definition 10 (Compactness) *Let (X, \mathcal{T}) be a topological space and let $\mathcal{U} := U_{i \in I} \subseteq \mathcal{T}$. The set \mathcal{U} is called an open cover of $Y \subseteq X$ if $Y \subseteq \bigcup_{i \in I} U_i$. A finite subset of \mathcal{U} whose union still contains Y is a finite sub-cover. It can then be said that Y is compact if every open cover of Y has a finite sub-cover.*

An important part of the characterization of data structures is the possibility to identify the object with the highest dimension, modeled by the concept of an open cell.

Definition 11 (Open Cell) *A subset $c \subset X$ of a Hausdorff space X is an open cell if it is homeomorphic to the interior of an open p -dimensional ball $\mathbb{D}^p = \{x \in \mathbb{R}^p : |x| < 1\}$.*

Collections of cells form larger structures, so-called complexes which are identified by the cell with the highest dimension, e.g., a p -dimensional space contains p -cells.

Definition 12 (Decomposition) *A decomposition \mathcal{E} of a topological space X is a decomposition of X into subspaces.*

With the concepts already introduced, the concept of a cell complex can be formulated.

Definition: CW-Complex \mathfrak{K} [2]

A pair $(\mathcal{T}, \mathcal{E})$, with \mathcal{T} a Hausdorff space and a decomposition \mathcal{E} into cells is called a CW-Complex if and only if the following axioms are satisfied:

- (C1) mapping function: for each p -cell $c \in \mathcal{E}$ a continuous function $\Phi_c : \mathbb{D}^p \rightarrow \mathcal{T}$ exists, which transforms \mathbb{D}^p homeomorphically onto the cell c and \mathbb{S}^{p-1} in the union of maximal $(p-1)$ dimensional cells. \mathbb{D}^p represents an p -dimensional closed unit ball and \mathbb{S}^{p-1} represents the sides of c .
- (C2) finite hull: the closed hull(c) of each cell $c \in \mathcal{E}$ connects only with a finite number of other cells.
- (C3) weak topology: $A \subset \mathcal{T}$ is open if and only if each $A \cap \text{hull}(c)$ is open.

A CW-cell complex with the underlying space X guarantees that all inter-dimensional objects are connected in an appropriate manner, such that $X^{(p)}$ is obtained from $X^{(p-1)}$ by attaching p -cells to each $(p-1)$ -cell and $X^{(-1)} = \emptyset$. The respective subspaces $X^{(p)}$ are called the p -skeletons of the cell complex.

A p -cell describes the cell with the highest dimension, e.g., 1-cell edges and 2-cell triangles. For this work the most important property of a CW-complex can be explained by using different p -cells and consistently attaching sub-dimensional cells to the p -cells. This fact is taken care of by the mapping function. For the common case a p -cell can be described by the oriented collection of the 0-cells, e.g., for a simplex cell $\tau_p = \{v_0, v_1, \dots, v_p\}, v_i \in \mathfrak{K}$, which is introduced in more detail in Section 2.3.1.

So far all mechanisms to handle the underlying topological space of data structures have been introduced. Each subspace can be uniquely characterized by its dimension. All subspaces are connected appropriately by the concept of a finite CW-cell complex.

2.3.1 Order Theory and Sets

The following section introduces concepts of order theory to formalize the combinatorial structure of cells and the global structure of cell complexes [54]. First, the most basic notion of a binary relation is introduced which is a specialization of the relation concept, where two arguments are used.

Definition 13 (Binary relation) *A binary relation R is a subset of the Cartesian product of two sets A and B . That is, any $R \subseteq A \times B$ is a binary relation.*

Based on this concept a partial order is defined. A partial order is required to create a hierarchy for our subsets, which can then be used to create our traversal mechanisms.

Definition 14 (Partial order) *A partial order is a binary relation that satisfies the following three properties:*

- (PO1) Reflexivity: $a \leq a, \forall a \in A$.
- (PO2) Antisymmetry: if $a \leq b$ and $b \leq a \forall a, b \in A$, then $a = b$.
- (PO3) Transitivity: if $a \leq b$ and $b \leq c \forall a, b, c \in A$, then $a \leq c$.

Definition 15 (Partial order on a topological space) *Let (X, \mathcal{T}) be a Hausdorff space. For any $x, y \in X$, a binary relation \leq on X : $x \leq y$, if and only if $x \in \text{cl}(y)$ is defined, where $\text{cl}(y)$ represents the closure for a set within a topological space, where a closure is the intersection of all closed sets containing y . Then this binary relation is a partial order.*

Definition 16 (Ordered Sets) *Let P be a set and \leq be a (partial) order on P . Then P and \leq form a (partially) ordered set.*

To give examples for ordered sets, Figure 2.2 illustrates the possible terms:

- If the order is total, so that no two elements of P are incomparable, then the ordered set is a totally ordered set, called *chain*. A formal definition is then given in Section 2.4.
- If no two elements are comparable unless they are equal, then the ordered set is an *anti-chain*.
- If P has two or more incomparable elements, then the ordered set is a *partially ordered set* or *poset*. One example of an incomparable expression is $\{a\}$ with $\{b, c\}$.

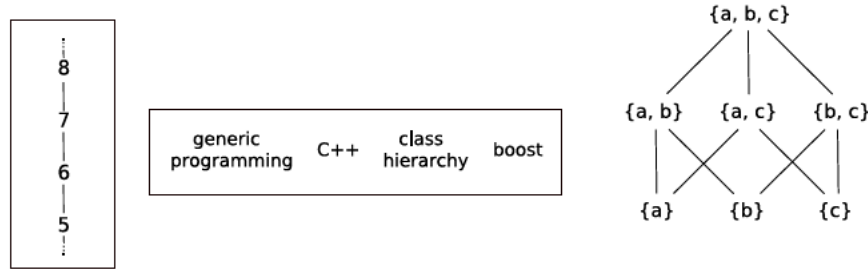


Figure 2.2: Left: Integers form a chain, totally ordered by \leq . Middle: Incomparable items forming an anti-chain. Right: The power-set of $\{a, b, c\}$ ordered by \leq as a partially ordered set.

Definition 17 (Extrema of Partial Ordered Sets [54]) Let S be an ordered set.

- $u \in S$ is said to be maximal in S if and only if there is no $v \in S$ such that $u \leq v$. A set may have any number of maximal elements, including none.
- If u is S 's only maximal element, then u is the maximum of S .
- The maximum element u (if it exists) is also called the top of S and is denoted by \top .
- Dually, $t \in S$ is said to be minimal in S if and only if there is no $s \in S$ such that $s \leq t$. A set may have any number of minimal elements, including none.
- If t is S 's only minimal element, then t is the minimum of S .
- The minimum element t (if it exists) is also called the bottom of S and is denoted by \perp .

Finally, the concept of *covering* is needed. For $x, y \in P$ ordered by \leq , it is said x is covered by y (written $x \prec y$), if $x < y$ and for any $z \in P$, $x \leq z < y$ implies $x = z$. This means that there is no element of P "between" x and y . Equivalently it can be said that y covers x .

Definition 18 (Hasse Diagram) A Hasse diagram of an ordered set is a graph in which:

- Each node corresponds to an element of the set,
- Each edge corresponds to a covering relation between the nodes it connects, and
- If x is covered by y ($x \prec y$), then the node for x is drawn in a lower position than the node for y .

It can be seen that in interpreting diagrams, it does not matter whether one node is above or below another unless there is a monotonic path between them; and that if there is a monotonic path from y through one or more nodes down to x , there is no separate edge directly from y to x , as can be seen in Figure 2.2.

Definition 19 (Lattice [18]) A lattice is a bounded poset in which any two elements a, b possess a unique maximal lower bound \perp and a unique minimal upper bound \top .

A lattice, e.g., for $p + 1$ -simplices, is atomic, if each element is the \top of elements of dimension 1 (the atoms), it is coatomic, if each element is the \perp of elements of maximal dimension p (the coatoms).

The \perp of two elements c_1, c_2 corresponds to their geometric intersection $\bar{c}_1 \cap \bar{c}_2$, the \top to the element c of least dimension containing both of them $\bar{c} \subset c_1 \cup c_2$. The \top of two elements may be the whole mesh.

The poset of a convex polytope is an atomic and coatomic lattice. For an arbitrary complex, the poset is in general not a lattice. If a grid's lattice is atomic, it means that p -cells are uniquely determined by their vertex sets if the lattice is coatomic, the vertex sets of p -cells for $p \leq n$ can be determined from the knowledge of the vertex sets of the p -cells alone.

2.3.2 Cell Topology

Order concepts for sets just introduced can be used to formalize the internal structure of an arbitrary p -cell, e.g., with the Hasse diagram. A simplex cell is thereby distinguishable from a cuboid cell type in several dimensions as depicted in Figure 2.3 and Figure 2.4, respectively.

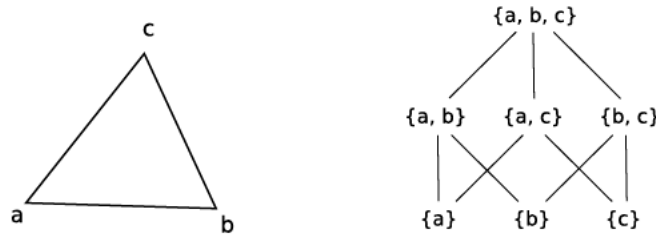


Figure 2.3: Cell topology of a simplex cell in two dimensions.

Also, sub-cells such as the corresponding edges and facets and their direct relations to the cell can be identified. An example of extracting all edges of the simplex cell which corresponds with the middle layer of the Hasse diagram can also be seen in Figure 2.3. Another important fact to mention is that the topological structure and the corresponding order hierarchy is invariant of the type of cell used. The $p - 1$ -layer of the cuboid cell represents the edges of this type of cell. As a higher dimensional example,

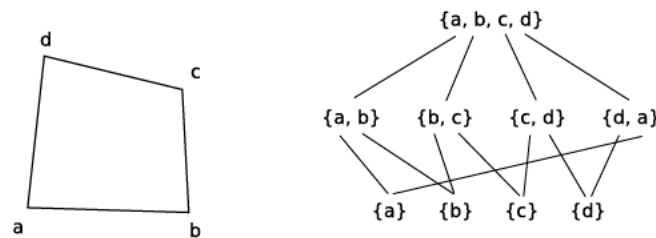


Figure 2.4: Cell topology of a cuboid cell in two dimensions.

a three-dimensional simplex example is given in Figure 2.5. Here, the $p - 1$ -cells are the facets, and, in this particular example, triangles. By using the actual dimension of the p -cell and deriving the $p - n$ cells only by means of the order structure, a flexible framework can be built, where an important property for characterizing cell complexes is required, the regularity of cells and the complex. If the mapping function Φ_e extends to a homeomorphism on the whole of \mathbb{D}^p , then the p -cell is regular; its sides are then homeomorphic to a decomposition of S^{p-1} , the boundary. The cell complex \mathfrak{K} is regular if all cells are regular. The lattice property of a cell complex then guarantees that intersections of the vertex sets of elements always uniquely define a lower dimensional element.

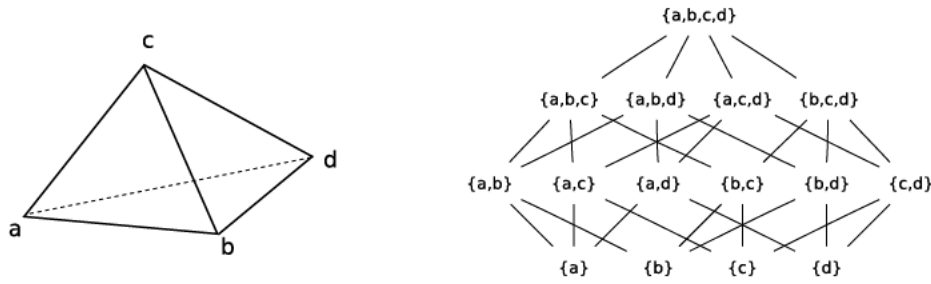


Figure 2.5: Cell topology of a 3-simplex cell.

2.3.3 Complex Topology

In contrast to the internal cell topology, a cell complex requires adjacency information of cells. There are different possibilities of storing this information efficiently [35, 53]. This work develops an abstract means of storing all different types of cells orthogonally, based on the cell topology of the complex topology. For dimensions greater than one, Figure 2.6 illustrates the complex topology of a 2-simplex cell complex where the bottom sets are now the cells. The rectangle in the figure marks the relevant cell number. The

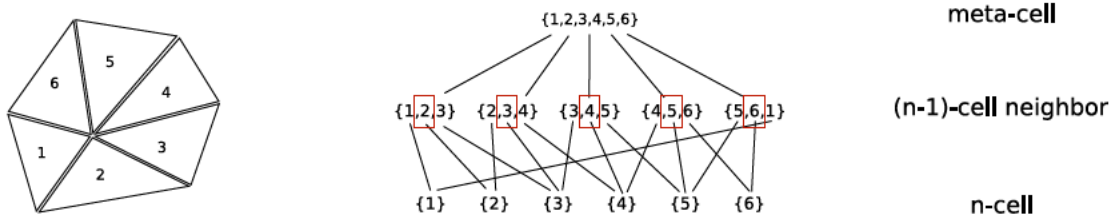


Figure 2.6: Complex topology of a simplex cell complex.

topology of the cell complex is only available locally because of the fact that the top set can have an arbitrary number of elements. The term *meta-cell* is used to describe various subsets with a common name. In other words, there can be an arbitrary number of triangles in this cell complex attached to the innermost vertex.

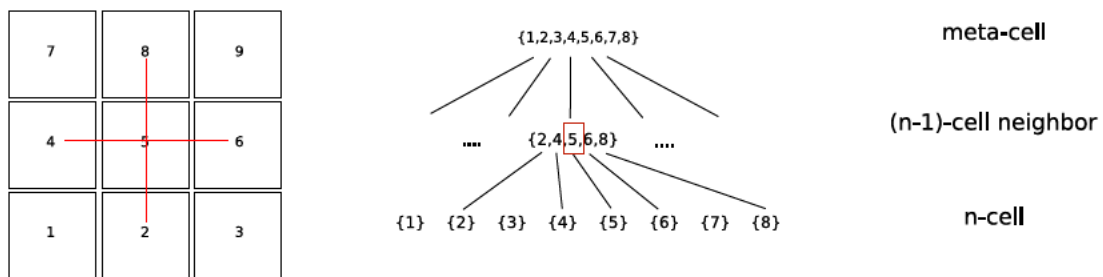


Figure 2.7: Complex topology of a cuboid cell complex.

Figure 2.7 presents the complex topology of a cuboid cell complex. The rectangle in the figure marks the main cell (5) under consideration. As can be seen, the number of attached cells is constant inside the space. The topology of the cell complex can thereby be used as a globally known property.

2.4 Computational Topology

The previous sections introduced combinatorial concepts for CW-complex representations and abstract classification mechanisms to manage scientific data. Compared to the theory of CW-complexes, this section focuses more on the details of algebraic properties of linear mappings by a procedure to associate a sequence of Abelian groups or modules with, e.g., a topological space. Only a few concepts are introduced in this section [22, 55], because of the fact that computational topology [19, 56] is a complex and emerging part of scientific computing in its own right.

The motivation for this section is to retain the structure of geometrical objects and even physical field approximations for computational mechanisms, because the recovery of lost structural information of objects has proven to be a very complex and difficult tasks.

2.4.1 Chains

To use the elements of a dimension of a cell complex, e.g., all edges, in a computational manner, a mapping of the p -cells onto an algebraic structure is needed. An algebraic representation of the assembly of cells with a given orientation is thereby made available. Whereas the cell topology is concerned about the internal structure of a given cell, the chain concept acts on certain p -cells. A formal definition for a p -chain (p^{th} chain group) is given by:

Definition 20 (P-Chain) A p -chain c_p defined over a cell complex \mathfrak{K} and a vector space \mathcal{V} is a formal sum

$$c_p = \sum_{i=1}^{n_p} w_i \tau_p^i \quad \tau_p^i \in \mathfrak{K}, w_i \in \mathcal{V} \quad (2.24)$$

respecting that the operation is closed under orientation reversal:

$$\forall \tau_p^i \in c_p \text{ there is } -\tau_p^i \in c_p \quad (2.25)$$

All different topological parts are called cells, and the dimensionality is expressed by adding the dimension such as a 3-cell for a volume, a 2-cell for surface elements, a 1-cell for lines, and a 0-cell for vertices.

Thus, two p -chains can be added, or a p -chain can be multiplied by a scalar. In addition, p -chains support algebraic-topological operations, including the boundary and coboundary operations. Based on these concepts, a cell complex can be seen as a formal structure where cells can be added, subtracted, and multiplied. The cell complexes used in this work have a chain group in every dimension. Homological concepts are applied here for the first time, due to the fact that homology examines the connectivity between two immediate dimensions. A structure-relating map between sets of chains C_p is therefore introduced².

Definition 21 (Boundary Homomorphism) Let \mathfrak{K} be a cell complex and $\tau_p^i \in \mathfrak{K}, \tau_p^i = \{k_0, k_1, \dots, k_p\}$. The boundary homomorphism $\partial_p : C_p(\mathfrak{K}) \rightarrow C_{p-1}(\mathfrak{K})$ is:

$$\partial_p \tau_p^i = \sum_i (-1)^i [k_0, k_1, \dots, \tilde{k}_i, \dots, k_n] \quad (2.26)$$

where \tilde{k}_i indicates that k_i is deleted from the sequence.

²This map is restricted to simplex cells. A more general mechanism related to the boundary operation is given in Section 2.3.2 and the corresponding practical concept in Section 6.2

This can be seen as a boundary operator, that maps p -chains onto the $p - 1$ -chains in their boundary. It should not be confused with the geometric boundary of a point set. This algebraic-topological operation defines a $(p - 1)$ -chain in terms of a p -chain. It is compatible with the additive and the external multiplicative structure of chains and builds a linear transformation:

$$C_p \xrightarrow{\partial_p} C_{p-1} \quad (2.27)$$

Therefore, the boundary operator can be used linearly

$$\partial \left(\sum_i w_i \tau_p^i \right) = \sum_i w_i (\partial \tau_p^i) \quad (2.28)$$

which means that the boundary operator can be used separately for each cell. The cell complex properties can be easily calculated by means of chains. 3-cells intersect on 2-cells or have an empty intersection. This operation can be described by the boundary operator ∂c_p of the cell complex and the corresponding orientation induced on them:

Definition 22 (Cell complex) A cell complex \mathfrak{K} is a set of cells that satisfy the following properties:

- The boundary of each p -cell τ_p^i is a finite union of $(p - 1)$ -cells in \mathfrak{K} : $\partial_p \tau_p^i = \bigcup_m \tau_{p-1}^m$
- The intersection of any two cells τ_p^i, τ_p^j in \mathfrak{K} is either empty, or is a unique cell in \mathfrak{K}

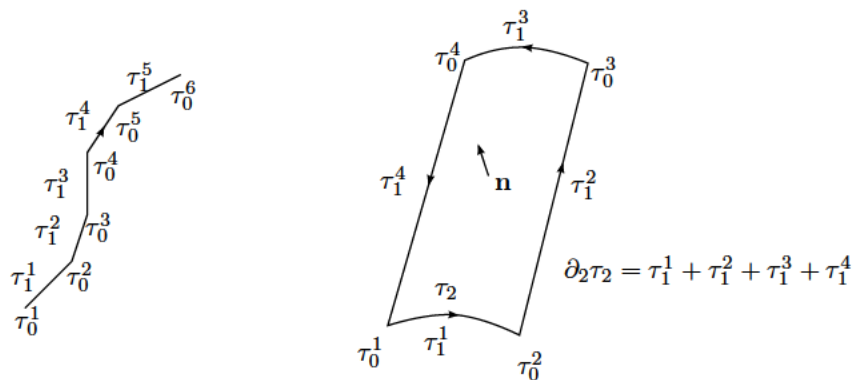


Figure 2.8: Representation of a 1-chain with boundary (left) and a 2-chain with boundary (right).

Figure 2.8 depicts two examples of 1-chains, 2-chains, and an example of the boundary operator. Applying the appropriate boundary operator to the 2-chain example read:

$$\partial_2 \tau_2 = \tau_1^1 + \tau_1^2 + \tau_1^3 + \tau_1^4 \quad (2.29)$$

$$\partial_1 (\tau_1^1 + \tau_1^2 + \tau_1^3 + \tau_1^4) = \tau_0^1 + \tau_0^2 - \tau_0^2 + \tau_0^3 - \tau_0^3 + \tau_0^4 - \tau_0^4 - \tau_0^1 = 0 \quad (2.30)$$

Using the boundary operator on a sequence of chains of different dimensions, a chain complex is obtained:

Definition 23 (Chain Complex) A chain complex $C_* = \{C_p, \partial_p\}$ is a sequence of modules C_p over a ring R and a sequence of homomorphisms

$$\partial_p : C_p \rightarrow C_{p-1} \quad (2.31)$$

such that

$$\partial_{p-1} \partial_p = 0 \quad (2.32)$$

2.4.2 Cochains

Before the introduction of the concept of chains only the simple structure of a cell complex was available. The cell complex only contains the set of cells and their connectivity. The introduction of the chain concept provides the concept of an assembly of cells and the corresponding algebraic structure. Chains can be seen as mappings from oriented cells as part of a cell complex to another space. This definition establishes the algebraic access of computational methods to handle the concept of a cell complex.

In addition to cell complexes, scientific computing requires the notation and access mechanisms to global quantities related to macroscopic p -dimensional space-time domains, introduced in Section 2.2. This collection of possible quantities, which can be measured, can then be called a field, which permits the modeling of these measurements as a field function that can be integrated on arbitrary p -dimensional (sub)domains. An important fact which has to be stated here is that all quantities which can be measured are always attached to a finite region of space. A field function can then be seen as the abstracted process of measurement of this quantity [31, 35]. The concept of cochains allows the association of numbers not only to single cells, as chains do, but also to assemblies of cells. Briefly, the necessary requirements are that this mapping is not only orientation-dependent, but also linear with respect to the assembly of cells, modeled by chains. A cochain representation is now the global quantity association with subdomains of a cell complex, which can be arbitrarily built to discretize a domain. Physical fields therefore manifest on a linear assembly of cells. Based on cochains, topological laws can be given a discrete representation.

Definition 24 (Cochains [25]) *A linear transformation σ of the p -chains into the field \mathbb{R} of real numbers forms a vector space $c_p \xrightarrow{\sigma} \mathbb{R}$ and is called a vector valued p -dimensional cochain or short p -cochain.*

The space of all linear mappings on c_p is denoted by C^p , where the elements of C^p are called cochains. Cochains express a representation for fields over a discretized domain \mathfrak{K} . Addition and multiplication by a scalar are defined for the field functions and so for cochains. To extend the expression possibilities, coboundaries of cochains are introduced.

Definition 25 (Coboundary) *The coboundary δ of a p -cochain is a $(p+1)$ -cochain defined as:*

$$\delta c^p = \sum_i v_i \tau_i, \quad \text{where} \quad v_i = \sum_{b \in \text{faces}(\tau_i)} \sigma(b, \tau_i) c_p(b) \quad (2.33)$$

Thus, the coboundary operator assigns non-zero coefficients only to those $(p+1)$ cells that have c_p as a face. As can be seen, δc_p depends not only on c_p but on how c_p lies in the complex \mathfrak{K} . This is a fundamental difference between the two operators ∂ and δ . An example is given in Figure 2.9 where the coboundary operator is used on a 1-cell. The coboundary of a p -cochain is a $p+1$ cochain which assigns to each $(p+1)$ cell the sum of the values that the p -cochains assigns to the p -cells which form the boundary of the $(p+1)$ cell. Each quantity appears in the sum multiplied by the corresponding incidence number. Cochain complexes [22, 53] are defined similarly to chain complexes except that the arrows are reversed.

Definition 26 (Cochain Complex) *A cochain complex $C^* = \{C^p, \delta^p\}$ is a sequence of modules C^p and homomorphisms:*

$$\delta^p : C^p \rightarrow C^{p+1} \quad (2.34)$$

such that

$$\delta^{p+1} \delta^p = 0 \quad (2.35)$$

Then, the following sequence with $\delta\delta = 0$ is generated:

$$0 \xrightarrow{\delta} C^0 \xrightarrow{\delta} C^1 \xrightarrow{\delta} C^2 \xrightarrow{\delta} C^3 \xrightarrow{\delta} 0 \quad (2.36)$$

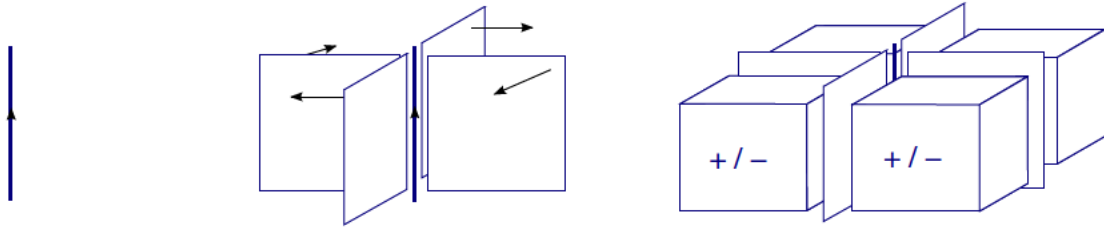


Figure 2.9: Cochain complex with the corresponding coboundary operator: $\mathfrak{K}^1 \xrightarrow{\delta} \delta\mathfrak{K}^1 \xrightarrow{\delta} \delta\delta\mathfrak{K}^1 = 0$

2.4.3 Homology and Cohomology

The concepts of chains can also be used to characterize properties of spaces, the homology and cohomology where it is only necessary to use $C_p(\mathfrak{K}, \mathbb{Z})$. The algebraic structure of chains is an important concept, e.g., to detect a p -dimensional hole that is not the boundary of a $p + 1$ -chain, which is called a p -cycle.

Definition 27 (Cycle) The p^{th} cycle (group) is $Z_p = \ker \partial_p$. A chain which is an element of Z_p is a k -cycle.

In other words, a cycle is a chain whose boundary is $\partial_p C_p = 0$, a closed chain. The already introduced boundary concept can also be related to homological terms:

Definition 28 (Boundary) The p^{th} boundary (group) is $B_p = \text{im } \partial_{p+1}$. A chain which is an element of B_p is a p -boundary.

Or, a boundary is a chain D_p for which there is a chain C_p such that $\partial_p C_p = D_p$. As introduced, the set of all n -cycles Z_n and the set of all n -boundaries B_n are vector spaces over \mathbb{Z}_2 . Since $\partial^2 = 0$, $B_n \subset Z_n$ is obtained. The homology is then defined by $H_n = Z_n/B_n$. The homology of a space is a sequence of vector spaces. The topological classification of homology is introduced [19] by $B_p = \text{im } \partial_{p+1}$ and $Z_p = \ker \partial_p$ so that $B_p \subset Z_p$ and $H_p = Z_p/B_p$ where $\beta_p = \text{Rank } H_p$. For cohomology $B^p = \text{im } d^{p+1}$ and $Z^p = \ker d^p$ so that $B^p \subset Z^p$ and $H^p = Z^p/B^p$ where $\beta^p = \text{Rank } H^p$.

Figure 2.10 depicts the homology of the three-dimensional chain complex with the respective images and kernels, where the chain complex of \mathfrak{K} is defined by $\text{im } \partial_{p+1} \subseteq \ker \partial_p$. As can be seen, the boundary operator expression yields $\partial_p \partial_{p+1} = 0$

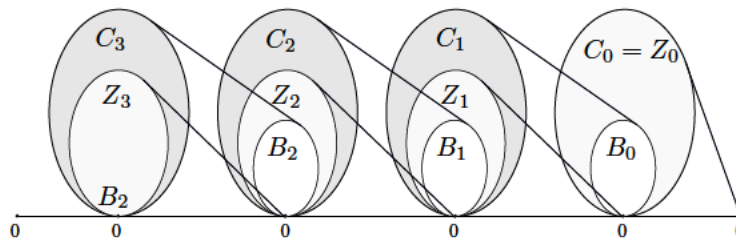


Figure 2.10: A graphical representation of (co)homology for a three-dimensional cell complex.

The first homology group [53] is the set of closed 1-chains (curves) in a space, modulo the closed 1-chains which are also boundaries. In the remainder of this work the ring R will be \mathbb{R} or \mathbb{Z} , in which case the modules C_p are vector spaces or Abelian groups, respectively, e.g., for \mathbb{R} the chain complex is given by $C_*(\mathfrak{K}; \mathbb{R} = \{C_p(\mathfrak{K}; \mathbb{R}), \partial_p\}$ or for the coefficient group \mathbb{Z} the following cochain complex $C_*(\mathfrak{K}; \mathbb{Z} = \{C_p(\mathfrak{K}; \mathbb{Z}), \partial_p\}$.

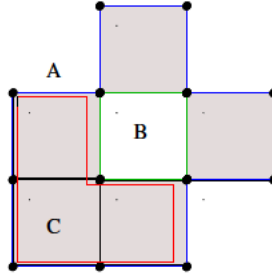


Figure 2.11: Illustration of cycles A, B, C and a boundary C . A, B are not boundaries.

To give an example, the first homology group is the set of closed 1-chains (curves) modulo the closed 1-chains which are also boundaries. This group is denoted $H_1 = Z_1/B_1$, where Z are cycles or closed 1-chains and B are 1-boundaries. Another example is given in Figure 2.11.

2.4.4 Duality between Chains and Cochains

The concepts of chains and cochains coincide on finite complexes [55]. Geometrically, however, C_p and C^p are distinct [25] despite an isomorphism. An element of C_p is a formal sum of p -cells, where an element of C^p is a linear function that maps elements of C_p into a field. Chains are dimensionless multiplicities, whereas those associated with cochains are physical quantities [35]. The extension of cochains from single cell weights to quantities associated with assemblies of cells is not trivial and makes cochains very different from chains, even on finite cell complexes. Nevertheless, there is an important duality between p -chains and p -cochains.

For a chain $c_p \in C_p(\mathfrak{K}, \mathbb{R})$ and a cochain $c^p \in C^p(\mathfrak{K}, \mathbb{R})$, the integral of c^p over c_p is denoted by $\int_{c_p} c^p$, and integration can be regarded as a mapping, where n represents the corresponding dimension:

$$\int : C_p(\mathfrak{K}) \times C^p(\mathfrak{K}) \rightarrow \mathbb{R}, \quad \text{for } 0 \leq p \leq n \quad (2.37)$$

Integration in the context of cochains is a linear operation: given $a_1, a_2 \in \mathbb{R}$, $c^{p,1}, c^{p,2} \in C^p(\mathfrak{K})$ and $c_p \in C_p(\mathfrak{K})$, reads

$$\int_{c_p} a_1 c^{p,1} + a_2 c^{p,2} = a_1 \int_{c_p} c^{p,1} + a_2 \int_{c_p} c^{p,2} \quad (2.38)$$

Reversing the orientation of a chain means that integrals over that chain acquire the opposite sign

$$\int_{-c_p} c^p = - \int_{c_p} c^p, \quad (2.39)$$

using the set of p -chains with vector space properties $C_p(\mathfrak{K}, \mathbb{R})$, e.g., linear combinations of p -chains with coefficients in the field \mathbb{R} . For coefficients in \mathbb{R} , the operation of integration can be regarded as a bilinear pairing between p -chains and p -cochains. Furthermore, for reasonable p -chains and p -cochains this bilinear pairing for integration is non-degenerate,

$$\text{if } \int_{c_p} c^p = 0 \quad \forall c_p \in C_p(\mathfrak{K}), \text{ then } c^p = 0 \quad (2.40)$$

and

$$\text{if } \int_{c_p} c^p = 0 \quad \forall c^p \in C^p(\mathfrak{K}), \text{ then } c_p = 0 \quad (2.41)$$

2.5 Fiber Bundles

To handle complex sets of scientific data, various concepts are already available, e.g., the concept of fiber bundles [21]. This concept enables the separation of data structural components from the data storage mechanisms. A further step towards a data structure model was investigated [1] where a possible identification of the discretized base space by a CW-cell complex of the fiber bundle was introduced. This section formally introduces the concept of fiber bundles and gradually refines this concept to cover the concepts of a CW-complex and chains within a common theoretical framework. A possible separation of combinatorial data structure and traversal properties modeled by a CW-complex and data access attributes is also provided here. The first required concept is a projection map to deal with correlated spaces.

Definition 29 (Projection Map) A projection map pr_1 is a function that maps each element of a product space to the element of the first space:

$$\begin{aligned} pr_1 : X \times Y &\longrightarrow X \\ (x, y) &\longmapsto x \end{aligned}$$

Definition 30 (Fiber Bundle) A fiber bundle structure on a topological space E , with fiber F^3 , consists of a map $\pi : E \rightarrow B$ such that each point of the topological space B has a neighborhood U_b for which there is a homeomorphism $f : \pi^{-1}(U_b) \rightarrow U_b \times F$ making the following diagram commute.

$$\begin{array}{ccc} \pi^{-1}(U_b) & \xrightarrow{f} & U_b \times F \\ \pi \downarrow & \searrow pr_1 & \\ U_b & & \end{array}$$

E is called the *total space*, B is called the *base space*, and F is called the *fiber space*. Commutativity of the diagram means that f carries each fiber $F_b = \pi^{-1}(U_b)$ homeomorphically onto $U_b \times F$. Thus the fibers F_b are arranged locally as in the product $U_b \times F$, though not necessarily globally. An example related to the discussed homeomorphism is given in Figure 2.12.

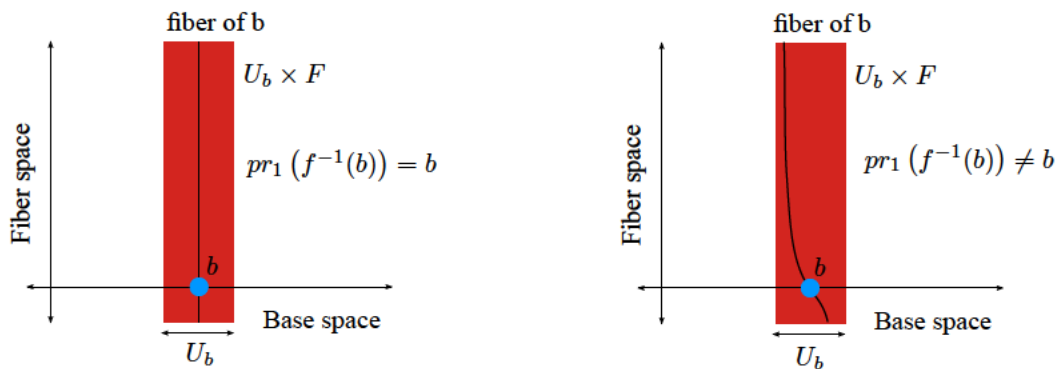


Figure 2.12: Left: a fiber bundle with the homeomorphism f . Right: A homeomorphism into $U_b \times F$, which does not preserve the projection, thus not revealing a fiber bundle [1].

The concept of an n -dimensional CW-cell complex introduced in Section 2.3 can be seen as a hierarchical system of spaces $X^{(-1)} \subseteq X^{(0)} \subseteq X^{(1)} \subseteq \dots \subseteq X^{(n)}$. This system of spaces can then be used to describe discrete base spaces by the identification of each $X^{(p)}$, with $0 \leq p \leq n$, as a separate base space,

³The concept of a fiber, or preimage, is introduced in Appendix A

with fibers that describe the relationship to the remaining $X^{(q)}$ spaces with $p! = q$. The total space of a complete CW-complex is then modeled by intrinsic fiber bundles on the set of p -skeletons.

An important fiber bundle related to the transition from a local fiber bundle to a global:

Definition 31 (Trivial Bundle) *If a fiber bundle over a space B with fiber F can be written as $B \times F$ globally, then it is called a trivial bundle $(B \times F, B, pr_1)$.*

An important representative of a fiber bundle is a vector bundle, where operations from linear algebra can be utilized and the fiber space is fully described by its dimensionality, also called the multiplicity, as it defines the number of quantities per point. The original fiber bundle data model approach [21] was based on the concept of a vector bundle concept.

Definition 32 (Vector Bundle) *If the fiber of a bundle is a vector space and has the general linear group of that vector space as structure group, then it is called a vector bundle.*

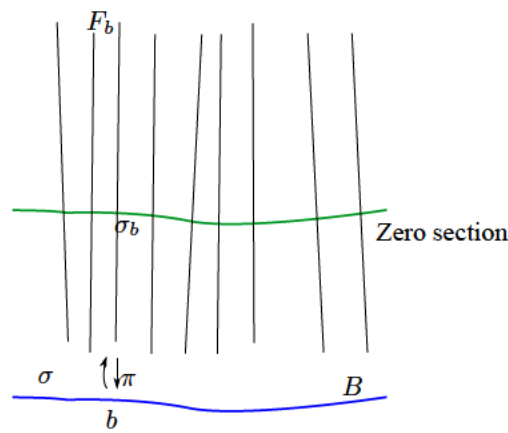


Figure 2.13: Zero section of a vector bundle [2].

Definition 33 (Section) *A map $\sigma : B \rightarrow E$ is called a section of a fiber bundle $(E, B, p : E \rightarrow B)$ if $p \circ \sigma : B \rightarrow B = id$.*

The section concept for fiber bundles provides the necessary mechanisms to obtain an element of the fiber over every point in the base space B , e.g., the *zero section* consists of the zero vectors, depicted in Figure 2.13, where the fibers corresponding to the base space are also given. Generally, a vector field is a section of this type of bundle.

To render the final transition to the approximation of physical semantics by a discrete setting for a space X , as given in Section 2.3, the concept of a manifold is introduced. This enables a local as well as global abstraction from the Euclidean perspective, but maintains the identification of Euclidean structure locally. The already declared concepts are sufficient to establish these abstractions.

Definition 34 (Topological Manifold) *A Hausdorff space \mathfrak{S} is a topological manifold if it is locally (that means in a neighborhood for every element $e \in \mathfrak{S}$) homeomorphic to an open subset \mathcal{M} of \mathbb{R}^n , where n is the dimension of \mathfrak{S} . This implies that for a neighborhood \mathcal{U} around an element $e \in \mathfrak{S}$ a mapping of the form*

$$\kappa : \mathcal{U} \rightarrow \mathcal{M} \subseteq \mathbb{R}^n \tag{2.42}$$

exists.

The neighborhood \mathcal{U} is called Euclidean. κ is called a chart and assigns a set of values from \mathbb{R}^n , commonly called coordinates, to the points in the neighborhood \mathcal{U} . This models the local Euclidean structure, where the elements e are usually called points.

In non-empty intersections of two neighborhoods $\mathcal{U}_1, \mathcal{U}_2 \subseteq \mathfrak{M}$ it is possible to define a transition from one chart to another in the following manner:

$$\kappa_1 \circ \kappa_2^{-1} : \kappa_2(\mathcal{U}_1 \cap \mathcal{U}_2) \rightarrow \kappa_1(\mathcal{U}_1 \cap \mathcal{U}_2) \quad (2.43)$$

This expresses the agreement of different charts in overlapping regions. A union of Euclidean neighborhoods that yields the topological space in combination with their respective charts is called an atlas. The specification of a Hausdorff space and an atlas characterizes a topological manifold.

To be able to construct more complex algebraic structures, which allow an appropriate modeling of physical fields, additional requirements are imposed on the purely topological manifold to arrive at a differentiable manifold.

Definition 35 (Differentiable Manifold) *A differentiable manifold \mathfrak{M} is obtained by demanding that the transition functions defined in Equation 2.43 be of differentiability class C^k .*

To use the well known concepts of integration and properties of functions, such as continuity and differentiability, the concept of a pullback is introduced.

Definition 36 (Pullback) *Let $f : \mathfrak{M} \rightarrow A$ be a map between a differentiable manifold \mathfrak{M} and a set A . By using a chart κ related to an element $e \in \mathfrak{M}$ with \mathcal{U} , a pullback is given by:*

$$f \circ \kappa^{-1} : \kappa(\mathcal{U}) \rightarrow A \quad (2.44)$$

Thereby the properties of $f \circ \kappa^{-1}$ in the open set \mathcal{U} can be translated to f related to the chart $\kappa(\mathcal{U})$. Before this concept becomes useful, the concept of a space attached to an element of the manifold must be introduced. A step towards such an attached space is the introduction of an abstract mechanism guiding the coordinate axes.

A mapping γ of an interval $[a, b] \rightarrow \mathfrak{M}$ describes a curve in the manifold \mathfrak{M} . A curve is called smooth if the composition $\gamma_\kappa = \kappa^{-1} \circ \gamma : [a, b] \rightarrow \mathbb{R}^n$ with an arbitrary chart κ is continuously differentiable with respect to at least one component of \mathbb{R}^n does not vanish.

Definition 37 (Tangent Vector) *A tangent vector is the differential of a smooth curve γ_κ .*

Different charts lead to different representatives of the same tangent vector. This concept finally allows tangent spaces to be attached at each point.

Definition 38 (Tangent Space) *A tangent space $\mathcal{T}_p(\mathfrak{M})$ at a point p is defined as the union of tangent vectors in this point p . The thus defined tangent space $\mathcal{T}_p(\mathfrak{M})$ is a vector space of the same dimension as \mathfrak{M} :*

$$\dim \mathcal{T}_p(\mathfrak{M}) = \dim \mathfrak{M} \quad (2.45)$$

Finally a connection between a differentiable manifold and the attached tangent spaces, and fiber bundles can be established by the concept of a tangent bundle $\mathcal{T}(\mathfrak{M})$, formally expressed in the following definition:

⁴The class C^k is composed by functions f which have continuous k -derivatives.

Definition 39 (Tangent Bundle) *The union of all tangent spaces $\mathcal{T}_p(\mathfrak{M})$ on a manifold \mathfrak{M} together with the manifold is called the tangent bundle $\mathcal{T}(\mathfrak{M})$:*

$$\mathcal{T}(\mathfrak{M}) := \{(p, v) : p \in \mathfrak{M}, v \in \mathcal{T}_p(\mathfrak{M})\} \quad (2.46)$$

A fiber of a point $p \in \mathfrak{M}$ is the tangent space $\mathcal{T}_p(\mathfrak{M})$ with a special dimensional restriction of the tangent space. The dimension of the bundle $\mathcal{T}(\mathfrak{M})$ is twice the dimension of the underlying manifold \mathfrak{M} ; its elements are points in addition to tangent vectors.

Where this given dimensional restriction of a bundle is a very specific specialization of the fiber bundle concept, several other identification can be obtained in a more general way by the amount of information that is available on the data in the fiber space (Figure 2.14). The fiber space of a fiber bundle has a certain dimension and thus an element has the same dimensionality at each point. If the dimensionality is unknown or may vary at each point, then the generalization of fiber bundles to a sheaf with stalks is modeled [57, 58]. If more information for an object with fixed dimension is available, e.g., some linearity relationship, then a vector bundle is specified. In the special case of the same dimensionality of the fiber space and the base space, a tangent bundle is obtained, and so it is a special case of the vector bundle that is built directly from the derivatives.

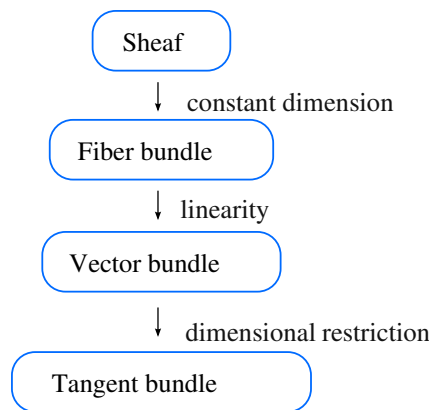


Figure 2.14: A hierarchy of concepts with partial specialization. The most general form is represented by a sheaf concept. The concept of fiber bundles is obtained by using fibers with a certain dimension. If the fiber space satisfies linear vector space properties, the concept of a vector bundle is derived. Finally, by confining the dimension of the base and fiber space, a tangent bundle is obtained.

A hierarchy for various abstractions to deal with scientific data is thereby available, but all are confined to one type of attached space. The following concepts now introduce spaces where additional attributes can be specified in a separate space. The basic properties of combinatorial elements and attached physical quantities are thereby possible. First, a non-degenerate mapping between two spaces is required.

Definition 40 (Scalar Product) *A scalar product is a non-degenerate bilinear mapping of two vector spaces into a field*

$$\varsigma : \mathcal{V} \times \mathcal{V}^* \rightarrow \mathbb{R} \quad (2.47)$$

Non-degenerate means that $\varsigma \neq 0$ for any fixed vector \mathbf{v} from one of the vector spaces, for all elements of the other vector space, except for the 0 element.

The scalar product is noted by $\langle \cdot, \cdot \rangle$ and enables the concept of a dual vector space.

Definition 41 (Dual Vector Space) *A vector space \mathcal{V}^* is called dual if it is related to a vector space \mathcal{V} by a scalar product.*

The elements of a dual vector space \mathcal{V}^* act as linear mappings⁵, or linear forms, on elements of \mathcal{V} . In the same fashion that the vector space \mathcal{V} has a dual vector space \mathcal{V}^* , the dual vector space is also associated with a dual vector space \mathcal{V}^{**} that again contains the linear forms on the elements of \mathcal{V}^* . These linear forms $\mathbf{v}^{**} \in \mathcal{V}^{**}$ can be obtained by applying the scalar product ζ to the linear forms $\alpha \in \mathcal{V}^*$ for fixed $\mathbf{v}_0 \in \mathcal{V}$

$$\langle \mathbf{v}_0, \alpha \rangle = \mathbf{v}_0^{**}(\alpha) \quad (2.48)$$

As the elements from \mathcal{V} now uniquely identify linear forms on \mathcal{V}^* by using only the structure of the scalar product that makes \mathcal{V}^* the dual of \mathcal{V} , the vector space \mathcal{V}^{**} can be identified with the original vector space \mathcal{V} . Accordingly a scalar product

$$\zeta^* : \mathcal{V} \times \mathcal{V}^* \rightarrow \mathbb{R} \quad (2.49)$$

is defined that is connected to the initial scalar product by $\langle \alpha, \mathbf{v} \rangle_* = \langle \mathbf{v}, \alpha \rangle$. This structure facilitates the introduction of an additional space, the cotangent space, as an extension to the concepts of a tangent space and tangent bundle.

Definition 42 (Cotangent Space) *The vector space structure of a tangent space \mathcal{T}_p along with a scalar product induces a dual vector space, the cotangent space \mathcal{T}_p^* .*

Figure 2.15 graphically illustrates the correlation of the concepts, whereas the linear forms on \mathcal{T}_p are called covectors or 1-forms and are an important step to describe arbitrary abstract quantities. The linear forms on \mathcal{T}_p^* are called (contravariant) vectors or multivectors \mathbf{v}_p [36, 59] and can be used to enable a discrete combinatorial setting of a manifold. An important step of this translation is the identification of the given subdomains \mathfrak{M}_p of the manifold by the concept of so-called multivectors [31, 34].

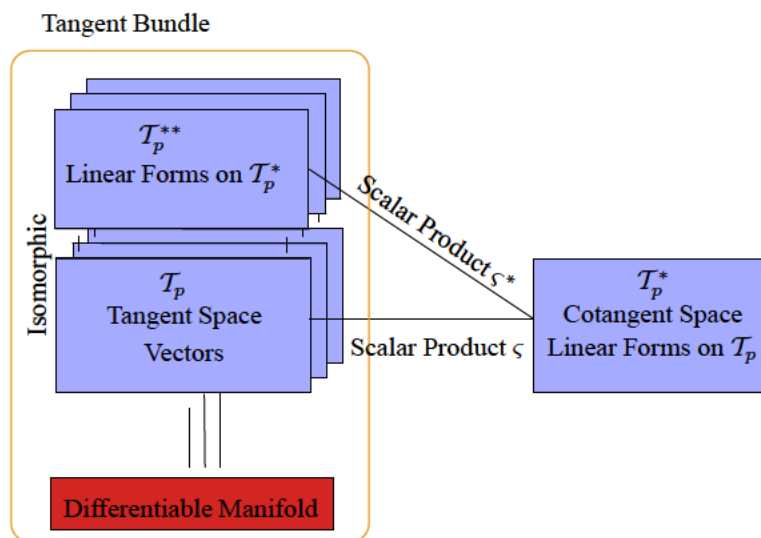


Figure 2.15: Illustration of the correlation of tangent space and cotangent space.

As already mentioned, the basic approach to describe scientific data by the concept of fiber bundles started by using the vector bundle concepts, Figure 2.14. To model cospaces, e.g., a cotangent space, the dimensionality alone is not sufficient to describe a fiber space, but further requirements have to be met to identify covariant and contravariant elements, thus requiring more structure. By using vector bundles, e.g., a tangent vector cannot be distinguished from a normal vector. To address this lack of structure an extension of the original vector bundle approach was given [1] to provide the fiber bundle with additional semantic meta-information.

⁵See Appendix A for a definition.

2.6 Fiber Bundles and Chains/Cochains

This section combines the fiber bundle concepts and the computational topological concepts. The fiber bundle approach and the separation properties related to a base space offers a distinct modularization for application development for scientific computing. A modular system of software components is directly induced which always communicates over a formal interface, the preimage property. Likewise, the concept of chains transforms the properties of a cell complex directly to a computationally manageable algebraic structure. The final objective is to introduce operators that can be formulated generically, independently from all dimensional attributes. First, the tangent bundle and cotangent bundle concept is illustrated in Figure 2.16.

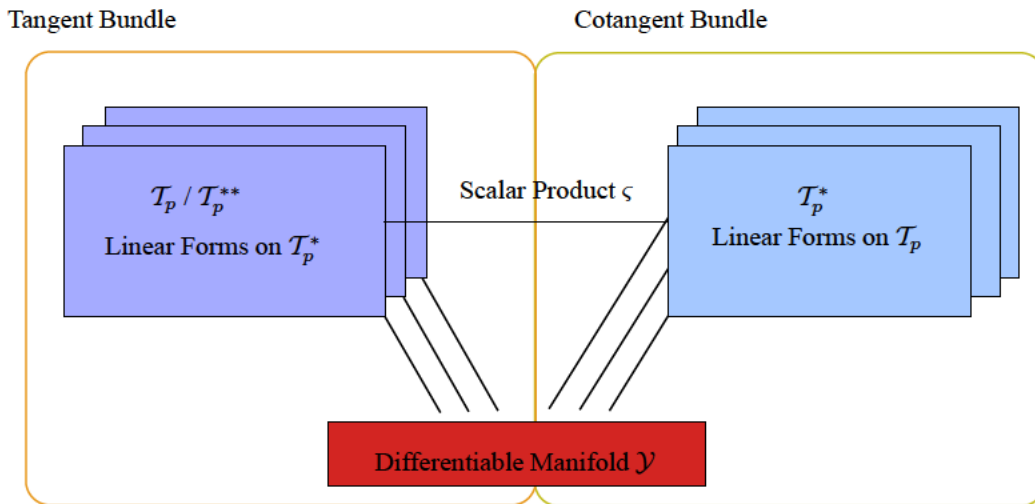


Figure 2.16: The identification of the concept of fiber bundles and the chain and cochain concept as dual spaces.

Then, the approach of a fiber bundle data model, where a discrete representation of a manifold is modeled by a CW-complex and the corresponding $X^{(p)}$ spaces, is given. These subspaces are modeled by separate base spaces, where a fiber describes the relationship to the remaining $X^{(m)}$ spaces, with $p! = m$. The base space of the original problem is thereby constructed from these intrinsic relationship fiber bundles on the skeletons. An example is sketched in Figure 2.17, where a three-dimensional cell complex is depicted. Here, the vertex-on-cell and cell-on-vertex information is stored within the respective fiber spaces.

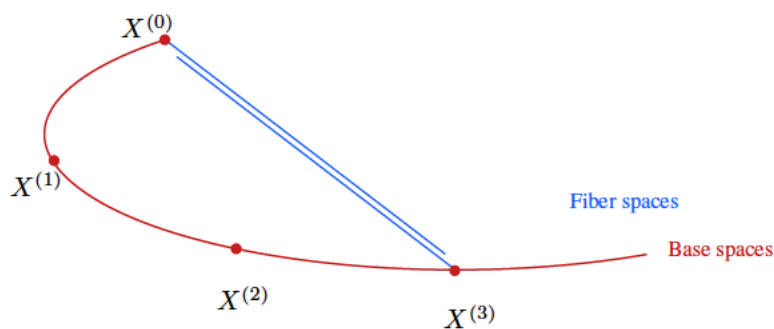


Figure 2.17: A representation of the intrinsic fiber bundles of the respective skeleton base spaces.

Related to this formulation of the fiber bundle concept, the chain concept uses the following concepts, where \mathfrak{K} represents an arbitrary cell complex, as given in Section 2.4:

$$c_p = \sum_{i=1}^{n_p} w_i \tau_p^i \quad \tau_p^i \in \mathfrak{K}, w_i \in \mathcal{V} \quad (2.50)$$

So a chain can be described by the intrinsic relationship fiber bundle of a p -skeleton, whereas the τ_p^i part is mapped to a linear form within $\mathcal{T}_p / \mathcal{T}_p^{**}$. The additional weight property is normally degenerate, which means that only the \mathbb{Z}_2 information is stored, e.g., only storing the cells of interest. These concepts can then be used in combination by the following identifications:

- Cell identification: this is the linear form on \mathcal{T}_p^* , here represented as τ_p^i , an element of the n -skeleton.
- Incidence information: inherent fiber space property of the corresponding skeleton (not directly visible in the definition).
- Weight concepts (optional) with w_i : an additional inherent fiber space property.

The only difference for the cochain concept is the space under consideration for the linear form, in this case the linear form on \mathcal{T}_p . Then the linear forms on \mathcal{T}_p and the linear forms on \mathcal{T}_p^* , which are identified by multivectors and p -forms, can be transferred to the algebraic bodies of chain and cochains complexes. The formal duality between vectors and covectors is also transferred to the duality between chains and cochains $\langle c_p, c^p \rangle$.

2.7 Discrete Electromagnetics

Several issues related to the differential formulation of the Maxwell equations were presented in Section 2.2. Finally, this section introduces a concise way of formulating physical problems regarding the fiber bundle and algebraic topology concepts. Starting with the integral formulation and partially reinserting the geometrical objects expressed in vector calculus notation, but omitting the orientation reads:

$$\int_{\partial V} \mathbf{B} \cdot d\mathbf{S} \quad (2.51)$$

$$\int_V \rho dV \quad (2.52)$$

A better-suited representation, which directly references the oriented geometric object a quantity is assigned to, is given with the formalism of ordinary and twisted differential forms which can be seen as the continuous counterpart of cochains, as introduced in Section 2.4. For a brief introduction of this topic, a p -dimensional differential form, or short p -form, can be seen as the subject to integration on p -dimensional domains [24, 60]. If the domain is internally oriented, then the p -form is called ordinary p -form which is denoted by ω^p and the corresponding externally oriented p -form is called twisted, denoted by $\tilde{\omega}^p$. By the concept of a multivector (see Section 2.4), a p -form is given as a linear function on the space of multivectors with values in an algebraic field. Then it follows that the pairing of a multivector, or p -vector \mathbf{v}_p , and a p -form ω^p gives a value like the pairing of a chain and cochain [35, 61], as given in Section 2.4.4. This analogy suggests the following representation of the pairing of a p -vector and a p -form:

$$\langle \mathbf{v}_p, \omega^p \rangle \quad (2.53)$$

The duality property, stated by $\langle c_p, c^p \rangle$, between chains and cochains transfers directly to the continuous multivectors and p -forms, introduced in Section 2.5. This is an important step towards a formal, consistent, and computationally manageable concept. A p -form ω^p on a continuous domain Ω can then be correlated to its discrete counterpart on a cell complex, a cochain c^p , since it associates a value c^i with each cell $\tau_p^i \in \mathfrak{K}$

$$c^i = \int_{\tau_p^i} \omega^p \quad (2.54)$$

Another correspondence between cochains and p -forms is given by the concept of the coboundary operator. As introduced in Section 2.4.2, the coboundary operator is defined to allow the transition from a topological equation of the form

$$\langle \partial c_{p+1}, a^p \rangle = \langle c_{p+1}, b^{p+1} \rangle \quad (2.55)$$

to the following relation between cochains:

$$\delta a^p = b^{p+1} \quad (2.56)$$

The continuous differential forms can then be related

$$\int_{D_{p+1}} d\omega^p := \int_{\partial D_{p+1}} \omega^p \quad \forall D_{p+1} \in D \subseteq \Omega \quad (2.57)$$

where d is an operator transforming p -forms into $p + 1$ -forms. This operator is called the exterior differential and mimics the property of the coboundary operator by transforming a topological equation given in integral form

$$\int_{\partial D_{p+1}} \alpha^p = \int_{D_{p+1}} \beta^{p+1} \quad \forall D_{p+1} \in D \quad (2.58)$$

into

$$d\alpha^p = \beta^{p+1} \quad (2.59)$$

Given the properties of the coboundary operator δ , the exterior differential d can be seen as the continuous counterpart [53] of δ . The following table depicts the correspondence between discrete and continuous concepts [35].

discrete setting		continuous setting	
p -cell	τ_p	Ω_p	p -dimensional domain
boundary of a p -cell	$\partial\tau_p$	$\partial\Omega_p$	boundary of a p -dimensional domain
p -chain	c_p	\mathbf{v}_p	weighted p -domain
p -cochain	c^p	ω_p	p -differential form
pairing of p -chain and p -cochain	$\langle c_p, c^p \rangle$	$\int_{\mathbf{v}_p} \omega^p$	weighted p -integral of a p -form
coboundary operator	δ	d	exterior differential operator

Based on these concepts, the local vector field representation \mathbf{B} becomes an ordinary 2-form b^2 and the scalar field ϱ a twisted 3-form $\tilde{\varrho}^3$:

$$\mathbf{B} \rightarrow b^2 \quad (2.60)$$

$$\varrho \rightarrow \tilde{\varrho}^3 \quad (2.61)$$

The numbers and orientation give the dimension on which these quantities are to be integrated. The adjoint of the exterior differential as the boundary of a weighted domain is the generalized Stoke's theorem:

$$\int_{\mathbf{v}_{p+1}} d\omega^p = \int_{\partial\mathbf{v}_{p+1}} \omega^p \quad (2.62)$$

It has to be noted, that the given differential form expression is more general than the vector calculus notation due to the fact that the expression is valid for $\text{div}()$, $\text{grad}()$, $\text{curl}()$ and the discrete chain and cochain representations automatically express the type of the dimension with the general notion of:

$$\langle c_{(p+1)}, \delta c^{(p)} \rangle = \langle \partial c_{(p+1)}, c^{(p)} \rangle \quad (2.63)$$

Examples of p -form complexes for differential operators encountered in different works [24, 62] for vector analysis in three dimensions are denoted by:

$$0 \rightarrow \{\text{scalar functions}\} \xrightarrow{\text{grad}()} \{\text{field vector}\} \xrightarrow{\text{curl}()} \{\text{flux vectors}\} \xrightarrow{\text{div}()} \{\text{volume densities}\} \rightarrow 0 \quad (2.64)$$

The concept of constitutive links closes the gap between ordinary and twisted cochains with discrete links between them. Two different types can be obtained:

$$L_1 : C^p(\mathcal{R}) \rightarrow C^q(\tilde{\mathcal{R}}) \quad (2.65)$$

$$L_2 : C^r(\tilde{\mathcal{R}}) \rightarrow C^s(\mathcal{R}) \quad (2.66)$$

The inherently discrete computer implementation can now be equipped with all the necessary information and structure regarding the physical entities.

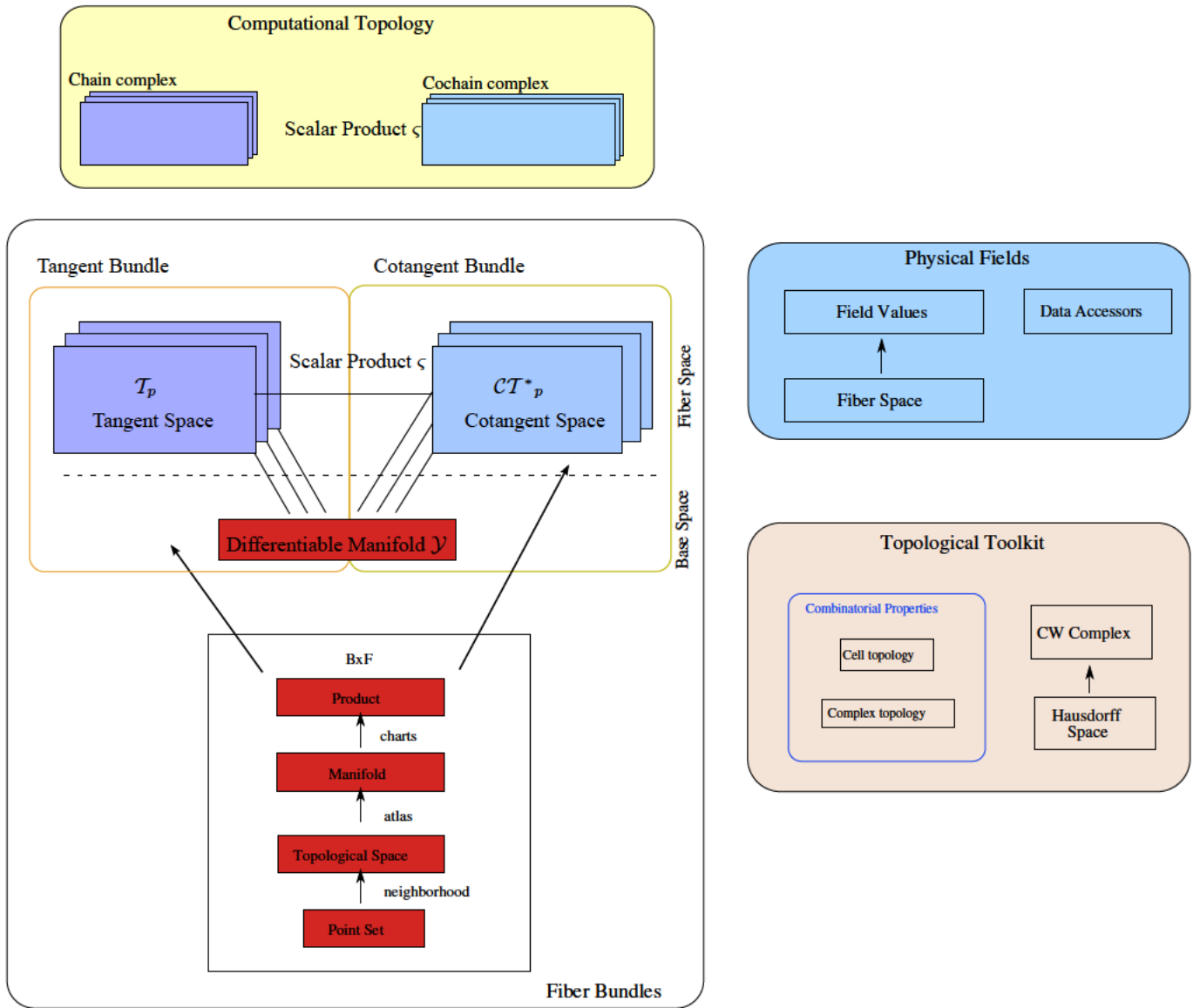


Figure 2.18: Overview of the mathematical concepts which have been introduced for the separation of a physical field domain into a base space and fiber space related to the concept of fiber bundles. The topological toolkit provides concepts to describe the internal structure of the cells of the cell complex to enable a generic data structure specification as well as generic traversal mechanisms. Concepts from computational topology such as chain and cochain complexes to obtain vector space structures within the tangent and cotangent space, are presented. Generic data access mechanisms are based on discrete field representations by cochains with their corresponding geometrical orientation and dimension.

Chapter 3

Numerical Discretization Schemes

An important step in handling partial differential equations is to use and develop stable, consistent, and accurate algebraic replacements where most of the global/continuous information of the original problem and more importantly, the inherent structure, are retained. Several methods are currently in use, such as the finite volume (FV), finite element (FE), and finite difference (FD) methods, each with specific approaches to discretization. Topological equations have an intrinsically discrete nature, compared to the constitutive parts of the field equations, which are the central issues in the construction of effective discretization schemes and the only place where recourse to local representations is fully justified. Numerical discretization schemes can be briefly represented as a model reduction, e.g.:

$$\mathcal{L}(u) = f \rightarrow \mathbf{Ax} = \mathbf{b} \quad (3.1)$$

which transforms an infinite-dimensional operator equation into a finite-dimensional algebraic equation. Here it can already be seen that this is always accompanied by an inevitable loss of information due to the reduction of dimension. Briefly, the given discretization schemes address differently the task of replacing the partial differential equation system with algebraic ones. Therefore, generic discretization concepts, based on what has been called the reference discretization scheme [33, 35], are introduced first. These concepts are then presented in the context of each of the other methods.

- The finite volume method is, with respect to the global and discrete formulation, based on topological laws, the most natural. The method is based on the approximation of conservation laws directly in its formulation and is therefore flux conserving by construction. The topological laws and time stepping procedures can be integrated easily.
- The finite element method can be seen as a remarkably flexible and general method for solving partial differential equations. Compared to the finite volume method, the spatial discretization can be much more arbitrary with fewer quality constraints. The continuous problem with an infinite amount of degrees of freedom is reformulated as an equivalent variational problem with a finite dimensional space. As described in the corresponding finite element section, this method is able to incorporate the constitutive relations appropriately. The incorporation of topological laws and time-dependent problems is more complex.
- The last scheme used in this work is the finite difference scheme, which addresses the problem of numerical analysis from a quite different approach when compared to the preceding two. Instead of using the conservation of the original problem (FV) or projecting the continuous problem into a finite dimensional space (FE), the finite difference method uses a finite difference approximation for the differential operators. Despite the fact that this method is simple and effective as well as easy to derive and implement, this approach gives an optimal solution to a different problem than the originally intended discretized field equation. This method is limited to structured grids or global cell complexes.

3.1 Generic Discretization Concepts

This section gives an overview of the basic concepts for discretization schemes in scientific computing. The most important mechanisms are introduced and are then later converted into applicable and well known discretization schemes, such as the above mentioned finite volumes, finite elements, and finite differences. The purpose of this section is to unify different discretization schemes to a common kernel which is then directly converted to software design concepts. The concepts for discretization are mostly derived from an abstract reference discretization scheme [35], which is not a full discretization scheme by itself. Here only the necessary concepts for time-dependent electromagnetic problems which are discussed in subsequent chapters, are introduced, with some additional concepts from literature [24, 32, 33, 36]

3.1.1 Domain Discretization

The simulation domain, either a simple space or a more complex space-time, is discretized by means of two dual oriented cell complexes for the primary and secondary mesh. The two cell complexes and therefore the primary and secondary mesh are not required to be physically or geometrically distinct. The logical separation is required to distinguish the different cochains/differential forms only. With dual cell complexes, the attribution of boundary conditions and the treatment of material discontinuities can be greatly simplified [36]. The use of non-dual cell complexes, e.g., operator adjointness, is more complex to obtain. The primary p -cells are identified by τ_p whereas the corresponding secondary $n - p$ -cells are identified by τ_{n-p} and the default orientation, as given in Figure 3.1 for three dimensions.

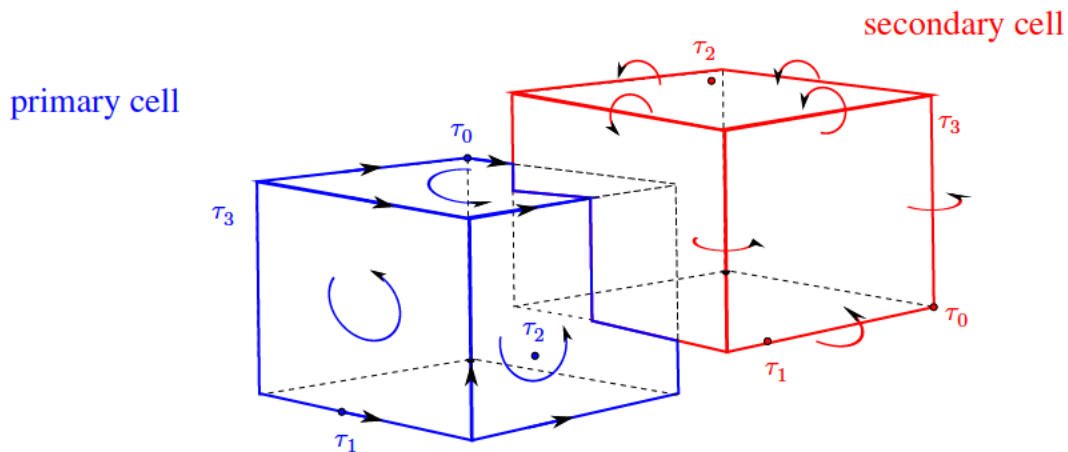


Figure 3.1: Discretization concept for the primary and secondary cell complex with consistent orientation.

Time is also subdivided into two dual cell complexes. Here the 0-cells are time instants indexed with increasing time. The time interval between t^n to t^{n+1} is indexed as $t^{n+1/2}$, similar to the formulation used by Yee (see Section 3.4.2). The secondary cells are given by \tilde{t}_{1-p}^n and depicted in Figure 3.2.

The time-dependent equation

$$\text{curl}(\mathbf{E}) = -\partial_t \mathbf{B} \quad (3.2)$$

can then be interpreted as a higher dimensional object of the spatial domain \times the time domain:

$$E(\partial A \times I) + B(A \times \partial I) = 0(A \times I) \quad (3.3)$$

A graphical representation is given in Figure 3.3 where two 0-cells are used for the corresponding time

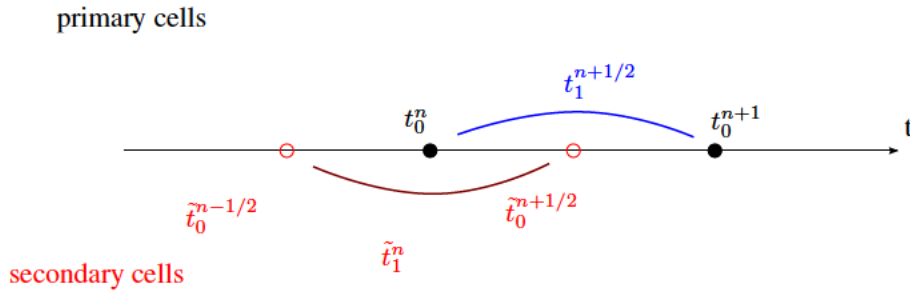


Figure 3.2: Time stepping for the discretization concepts.

steps t_0^n and t_0^{n+1} and one 1-cell for the time interval $t_1^{n+1/2}$. The corresponding quantity B is attached to the time interval $t_1^{n+1/2}$, whereas the quantity E corresponds to the t_0 cells.

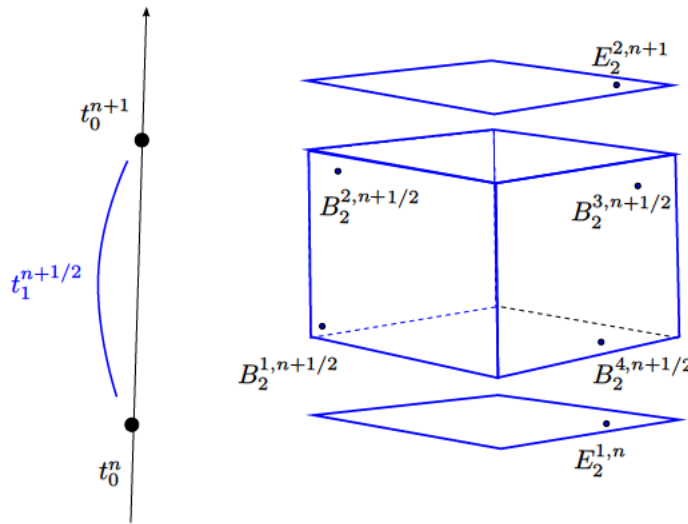


Figure 3.3: Space-time stepping for the discretization concepts.

This topological time stepping was proven [35] to be consistent with the conservation of charge as well as to preserve the absence of sources for the Faraday and Amperé laws. Based on this unique topological time stepping and on various discrete constitutive links, a variety of numerical time-stepping procedures, including explicit and implicit ones, can be derived.

3.1.2 Topological Equations

Balance or conservation equations can be written in local or global forms [24, 33], e.g.:

$$\operatorname{div}(\mathbf{D}) = \varrho \quad Q_{src}(V) = Q_{flow}(\partial V) \quad (3.4)$$

where the left equation represents a local or differential formulation and the right equation represents a global or discrete form. The discretization of the given relation means that the equation must be written for domains of finite extension. For topological equations this transition takes place without any error of approximation [33]. The balance equation by itself does not require uniform fields, homogeneous materials or any other conditions. Various laws which can be described by topological equations are intrinsically discrete. They state a balance of global quantities whose validity does not depend on metrical or material

properties and is therefore invariant for very general transformations. This gives them topological significance and the name topological equation was used [33] to express the idea of invariance under arbitrary homeomorphic transformations. For a topological equation, the discrete or global version appears as the fundamental one, with the differential statement resulting from it if additional requirements are fulfilled.

3.1.3 Constitutive Relation Discretization

Discretizing constitutive relations determines the link between various cochains which represents the field quantities approximated by the local constitutive equations. This step of discretization offers a great number of different choices [35].

Field Function Reconstruction and Projection

For most of the finite volume and finite element schemes, a field projection is used by:

$$\mathbf{B} = f_\mu \mathbf{H} \quad (3.5)$$

In the discrete setting the field functions \mathbf{B} and \mathbf{H} do not belong to the problem's variable. Instead the magnetic flux cochain Φ^2 and the magnetic field cochain $\tilde{\Psi}^2$ are linked by the relation $\Phi^2 = F_\mu(\tilde{\Psi}^2)$.

From the cochain $\tilde{\Psi}^2$ a field function is derived by a reconstruction operator:

$$\mathbf{H} = R_2(\tilde{\Psi}^2) \quad (3.6)$$

Next, the local constitutive link, Equation 3.5, is used to derive \mathbf{B} . Then, the cochain Φ^2 has to be obtained by means of a projection operator P^2 , which produces a cochain for each field function \mathbf{B} :

$$\Phi^2 = P^2(\mathbf{B}) \quad (3.7)$$

The discrete constitutive link F_μ is then finally given by:

$$\Phi^2 = F_\mu(\tilde{\Psi}^2) = P^2(f_\mu(R_2(\tilde{\Psi}^2))) \quad (3.8)$$

A natural requirement for the reconstruction and projection operators is that for each cochain c^p the following relation holds true [35]:

$$P^*(R_*(c^p)) = c^p \quad (3.9)$$

To obtain a sparse matrix representation the reconstruction process is usually performed locally, so the value of the reconstructed field function in a particular point depends only on the values of the original cochain on the cells in a sufficiently small neighborhood of the point.

During the last decade a comprehensive framework for the reconstruction and projection operators, called mimetic discretization [25, 37, 63–66], was derived. This framework analyzes in detail how these two operators can be used to mimic the analytical and continuous nature of partial differential equation.

3.2 Finite Volumes

The finite volume method is the most natural discretization scheme, because it makes use of the conservation laws in integral form. It subdivides the domain into cells and evaluates the field equations in integral form on these cells. In the area of TCAD this method is often called the finite box method [26] to express the geometrical origin of the discretized domain. Most applications of the finite volume method do not include the time variable, which has to be enforced by a separate discretization. The concepts and implementation are simple for different dimensions and different types of cell complexes.

3.2.1 Basic Concepts

For the finite volume method the domain Ω is first subdivided into non-overlapping control volumes V_i of a cell complex \mathfrak{K} . In each control volume an integral conservation law is imposed [27]:

Definition 43 (Integral Conservation Law) *An integral conservation law asserts that the rate of change of the total amount of a quantity with density u in a fixed control volume V is equal to the total flux of the quantity through the boundary ∂V .*

$$\partial_t \int_V u \, dV + \int_{\partial V} f(u) \cdot d\mathbf{A} = 0 \quad (3.10)$$

By the step towards the discrete space, the integral conservation law is transferred to small control volumes:

$$V = \bigcup_{i=1}^N V_i, \quad V_i \cap V_j = 0, \quad \forall i \neq j \quad (3.11)$$

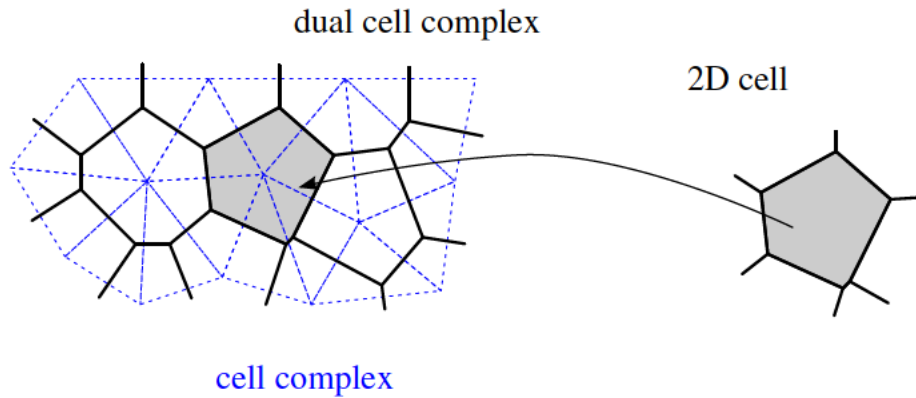


Figure 3.4: Finite volume requirements for a primary and secondary cell complex.

The integral conservation law is readily obtained upon spatial integration of, e.g., a divergence equation in a region Ω_i . The choice of control volume tessellation is flexible in the finite volume method. The primary requirement is a secondary mesh (see Figure 3.4) with special properties related to the primary mesh.

In a vertex-centered method the control volumes are formed as a geometric dual to the cell complex and unknown solutions are stored on a per-vertex basis. In the cell-centered method the cells serve directly as control volumes containing the unknown solutions (degrees of freedom) stored on a per-cell basis. See Figure 3.5 for a graphical depiction of these two methods. The integral conservation law is enforced for each control volume and for the entire domain. To obtain a linear system of algebraic equations, integrals must be expressed in terms of mean values.

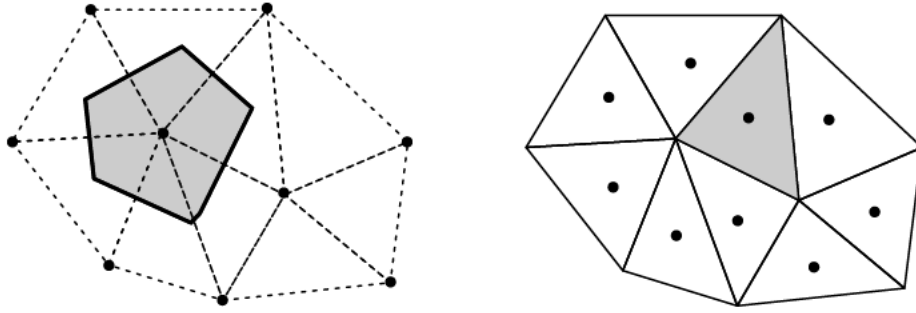


Figure 3.5: Primary control volumes used in the finite volume method. Left: cell-centered. Right: vertex-centered.

Two assumptions are fundamental to the finite volume method. First, a piecewise constant cell average is introduced for each control volume:

$$u_i = \frac{1}{|V_i|} \int_{V_i} u \, dV \quad (3.12)$$

This cell average results in a discontinuity at the cell interfaces. The corresponding single solution flux is thereby ambiguous at the interface. A geometrical interpretation is given in Figure 3.6 for a one-dimensional cell complex.

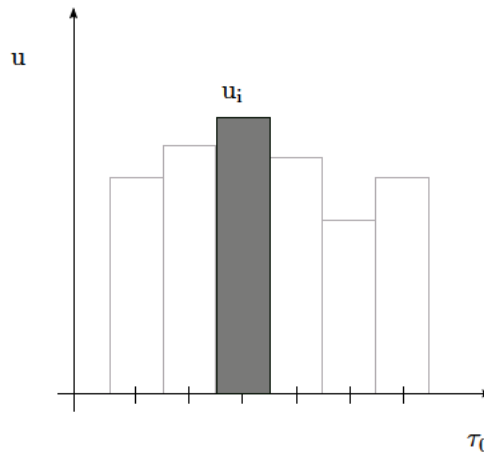


Figure 3.6: Finite volume and the rendered cell average value within each cell for a one-dimensional cell complex.

Second, the true flux at the interfaces is replaced by a numerical flux function $g(u, v) : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$. For arbitrary spatial dimensions, the flux integral is approximated by a numerical integration scheme, e.g., by a quadrature rule:

$$\int_{\partial V} f(u) \cdot d\mathbf{A} \approx \sum_{f_{jk} \in \partial V} g_{jk}(u_j, u_k) \quad (3.13)$$

The numerical flux has to satisfy the following properties:

- Flux conservation: the flux resulting from adjacent control volumes sharing an interface has to cancel exactly under summation:

$$g_{j,k}(u, v) = -g_{k,j}(v, u) \quad (3.14)$$

- Consistency: the numerical flux, evaluated with identical arguments, has to reduce to the true total flux passing through the boundary elements f_{jk} :

$$g_{jk}(u, u) = \int_{f_{jk}} f(u) \cdot d\mathbf{A} \quad (3.15)$$

Using Equation 3.12 and the previous interpretation of finite volumes for stationary meshes produces the following evolution equation for cell averages:

$$\frac{d}{dt} \int_{V_i} u \, dV = |V_i| \frac{d}{dt} u_i \quad (3.16)$$

One of the simplest finite volume schemes in a semi-discrete formulation can be obtained by utilizing representations which are continuous in time, $t \in [0, \infty]$, and piecewise constant in space, $u_h(t) \in V_h^0$ such that:

$$u_j(t) = \frac{1}{|V_i|} \int_{V_i} u_h(x_i, t) \, dV \quad (3.17)$$

with initial data

$$u_j(0) = \frac{1}{|V_i|} \int_{V_i} u_0(x_i) \, dV \quad (3.18)$$

and the numerical flux functions $g_{jk}(u_j, u_k)$ given by the following system of ordinary differential equations:

$$\frac{d}{dt} u_j + \frac{1}{|V_i|} \sum_{f_{jk} \in \partial V} g_{jk}(u_j, u_k) = 0 \quad \forall V_i \in V \quad (3.19)$$

The final remaining issue is that the solution is available only at the computational nodes, the control volume centers. Interpolation is needed to obtain the function values at the vertices.

Using an explicit or implicit time integration formula [67], e.g., a forward Euler scheme:

$$\frac{d}{dt} u_j \approx \frac{u_j^{n+1} - u_j^n}{\Delta t} \quad (3.20)$$

produces a fully-discrete finite volume form:

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{|V_i|} \sum_{f_{jk} \in \partial V} g_{jk}(u_j^n, u_k^n) = 0 \quad \forall V_i \in V \quad (3.21)$$

3.3 Finite Elements

The finite element method is a systematic approach to approximate the unknown exact solution of a partial differential equation based on basis functions and the projection of a given domain onto a consistent finite cell complex. The finite elements correspond to the p -cells of the complex.

The origin of the finite element method is in the solid mechanics of rigid bodies [29]. Some constraints should be placed on the selection of the shape functions to guarantee the fact that with an arbitrary number of shape functions the exact solution is approximated best, and in the limit, the exact solution should be obtained.

A good approximation is obtained by a residual formulation where the residuum is formulated as the difference of the unknown exact solution and the calculated approximate solution. This residuum is weighted over the simulation domain and integrated with the requirement that the integral vanishes with a set of linearly independent weighting functions. The other possible mechanism is the variational formulation of the partial differential equation [28, 68].

One of the main advantages of the finite element method is the possibility to adapt the basis functions to the eigenfunctions of the differential operators [28]. Thereby a high precision of this method can be obtained with a moderate number of mesh elements. Different areas of materials as well as anisotropic, inhomogeneous, and non-linear quantities can be treated. If no boundary conditions are declared, homogeneous/natural Neumann boundary conditions are implicitly given.

3.3.1 Basic Concepts for a Galerkin Method

In the following, a theoretical part of the finite element method is summarized, which is a special Galerkin method [28, 29] based on the following construction of finite dimensional subspaces \mathcal{V}_h :

- Ω is triangulated into a consistent cell complex
- \mathcal{V}_h consists of piecewise polynomials
- \mathcal{V}_h has a finite basis ϕ with local support

The main part of this section and the corresponding notation is based on [29, 69]. Galerkin's method can be briefly explained as a general technique for the construction of solution approximations not represented by infinite space basis functions $\phi_i \in \mathcal{V}$ but instead by a finite dimensional space $\mathcal{V}_h \subset \mathcal{V}$ of variational problems. The examples presented in this work are modeled by the following general form:

$$\mathcal{L}(u) = f(\mathbf{x}) \quad (3.22)$$

which is defined on an arbitrary dimensional, single connected domain Ω with boundary $\partial\Omega$. Equation 3.22 is fulfilled by functions of class C^2

$$u = u(\mathbf{x}) \in \mathcal{V} \quad (3.23)$$

The notation $\mathcal{L}()$ being a linear spatial differential operator. Additionally, it is assumed that the domain Ω has a piecewise smooth boundary $\Gamma_i \in \Gamma (\forall i \neq j \rightarrow \Gamma_i \cap \Gamma_j = \emptyset, \Gamma = \bigcup_i \Gamma_i)$. A general form of boundary conditions can thereby be specified by:

$$\mathbf{A}_i u + \mathbf{B}_i \partial_n u = \mathbf{g}_i(u) \quad \text{on } \Gamma_i \quad (3.24)$$

where \mathbf{A}_i and \mathbf{B}_i are matrices consisting of functions sufficiently smooth on Γ_i and \mathbf{g}_i is a vector of continuous linear functionals. $\partial_n u$ denotes the outward normal derivative. A weak formulation of Equation 3.22 is obtained by using a test function v :

$$\text{Find } u \in \mathcal{V} : \langle v, \mathcal{L}(u) \rangle = \langle v, f \rangle \quad \forall v \in C^1(\Omega) \quad (3.25)$$

The infinite-dimensional space \mathcal{V} is then replaced by a sequence of finite-dimensional spaces \mathcal{V}_h with n linearly independent functions $\phi_i \in \mathcal{V}$ which span the space \mathcal{V}_h , e.g.,

$$\mathcal{V}_h = \left\{ v : v(\mathbf{x}) = \sum_{i=1}^n s_i \phi_i(\mathbf{x}) \right\}, \quad s_i \in \mathbb{R} \quad (3.26)$$

The basis should build a complete function system to guarantee the approximation for $n \rightarrow \infty$. Any function $v_h \in \mathcal{V}_h$ is uniquely determined by a finite number of degrees of freedom, e.g., function values. This results in the following discrete variational problem for conforming methods ($\mathcal{V}_h \subset \mathcal{V}$):

$$\text{Find } u_h \in \mathcal{V}_h : \langle v_h, \mathcal{L}(u_h) \rangle = \langle v_h, f \rangle \quad \forall v_h \in \mathcal{V}_h \quad (3.27)$$

By selecting $\phi_i \in \mathcal{V}_h$ the following system is obtained:

$$\langle \phi_i, \mathcal{L}(u_h) \rangle = \langle \phi_i, f \rangle \quad \text{for } i = 1, \dots, n \quad (3.28)$$

The solution variable u_h is then also expanded in terms of the basis $(\phi_i)_{1 \leq i \leq n}$ of \mathcal{V}_h .

$$u_h = \sum_{i=1}^n u_i \phi_i \quad (3.29)$$

A graphical representation of u_h is given in Figure 3.7.

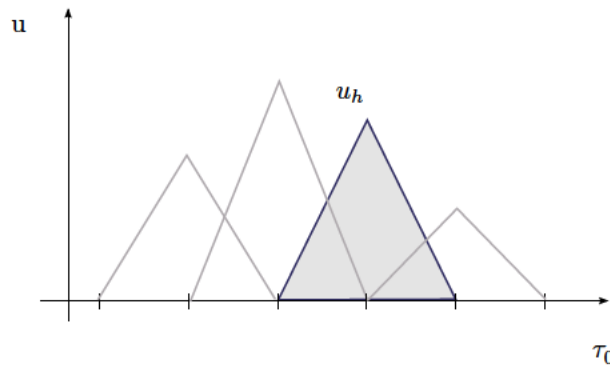


Figure 3.7: Geometrical interpretation of the basis of the expanded solution variable u_h for finite elements for a one-dimensional cell complex.

By the given selection of the test function and the expansion of the solution variable, it can be observed [69] that the scalar product in Equation 3.28 transforms the differential operator $\mathcal{L}()$ into the discrete operator

$$L : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad (3.30)$$

The residuum is obtained by using the operator of Equation 3.30

$$R_i = L(u_1, \dots, u_n) - \langle \phi_i, f \rangle \quad (3.31)$$

The next step for the discrete approximation of a given problem is the subdivision of the domain Ω according to cell complex properties (see Section 2.3 for details). The elements of the cell complex, the collection of cells, is then used as finite elements for the finite space \mathcal{V}_h . The basis functions $\phi_i \in \mathcal{V}_h$ are

defined on the global vertices τ_0^i , the 0-cells, only and are therefore called nodal basis functions. They can be expressed as:

$$\phi_i(\tau_0^i) = \delta_{ik}, \quad i, k = 1, \dots, n \quad (3.32)$$

Based on this subdivision of the finite element space, the function u_e can be expressed in local cell terms by

$$u_c = \sum_{i=1}^{n_p} u_i \phi_i \quad (3.33)$$

where i represents the local index of the element and n_p the number of element vertices. Now the operator from Equation 3.30 can be defined locally for, e.g., a 3-simplex (tetrahedron):

$$L^c : \mathbb{R}^4 \rightarrow \mathbb{R}^4 \quad (3.34)$$

and the local residuum is then defined as:

$$R_i = L^c(u_1, \dots, u_n) - \langle \phi_i, f \rangle \quad (3.35)$$

To determine the operator given in Equation 3.35 for a particular cell type, the basic nodal functions have to be calculated, e.g., for a 3-simplex cell (tetrahedron):

$$\phi_k(\mathbf{x}) = \phi_k(x_1, x_2, x_3) = \frac{1}{3V}(a_k x_1 + b_k x_2 + c_k x_3 + d_k) \quad (3.36)$$

where the coefficients a_k, b_k, c_k, d_k are functions of the vertex coordinates and V is the volume of the element. Next, the following integral has to be calculated:

$$M_{pq} = \int_{\tau_3^i} \phi_p(\mathbf{x}) \phi_q(\mathbf{x}) d\Omega, \quad p, q = 1, 2, 3, 4 \quad (3.37)$$

The key for an efficient practical realization is that global finite elements are defined as transformations of a reference element in a normalized coordinate system. Each point of the cell $(x_1, x_2, x_3) \in \tau_3^i$ can be expressed as a bijective function onto a reference point $(\xi, \eta, \zeta) \in \tau_{r,3}^i$:

$$x_1 = x_1^1 + (x_1^2 - x_1^1)\xi + (x_1^3 - x_1^1)\eta + (x_1^4 - x_1^1)\zeta, \quad (3.38)$$

$$x_2 = x_2^1 + (x_2^2 - x_2^1)\xi + (x_2^3 - x_2^1)\eta + (x_2^4 - x_2^1)\zeta, \quad (3.39)$$

$$x_3 = x_3^1 + (x_3^2 - x_3^1)\xi + (x_3^3 - x_3^1)\eta + (x_3^4 - x_3^1)\zeta \quad (3.40)$$

The nodal basis functions on the reference element $\tau_{r,3}^i$ are:

$$\phi_1^0(\xi, \eta, \zeta) = 1 - \xi - \eta - \zeta, \quad (3.41)$$

$$\phi_2^0(\xi, \eta, \zeta) = \xi, \quad (3.42)$$

$$\phi_3^0(\xi, \eta, \zeta) = \eta, \quad (3.43)$$

$$\phi_4^0(\xi, \eta, \zeta) = \zeta. \quad (3.44)$$

$$(3.45)$$

Equation 3.37 is then calculated by:

$$M_{pq}(\tau_{r,3}^0) = \det(\mathbf{J}) \int_{\tau_{r,3}^0} \phi_p^0(\xi, \eta, \zeta) \phi_q^0(\xi, \eta, \zeta) d\xi d\eta d\zeta, \quad p, q = 1, 2, 3, 4 \quad (3.46)$$

where \mathbf{J} is the Jacobian of the projection Equation 3.38 - Equation 3.40. The final assembly process consists of calculating stencil matrices $\Pi(\tau_n^i)$ for each element of the cell complex, mapping the local vertex indices to a global index and then building the global matrix \mathbf{A} .

For an electrostatic problem formulated by the Laplace equation $\text{div}(\text{grad}(\varphi)) = 0$ the stencil matrix is represented by:

$$\Pi(\tau_3^i) = M_{pq}(\tau_3^i) = \int_{\tau_3^i} \text{grad}(\phi_p(\mathbf{x})) \text{grad}(\phi_q(\mathbf{x})) d\Omega, \quad p, q = 1, 2, 3, 4 \quad (3.47)$$

This problem is linear and the global matrix \mathbf{A} is simply obtained by assembling Π with the corresponding index transformation.

3.4 Finite Differences

The finite difference discretization scheme is one of the simplest forms of discretization and does not easily include the topological nature of equations. A classical finite difference approach approximates the differential operators constituting the field equation locally. Therefore a structured grid is required to store local field quantities. For each of the points of the structured grid the differential operators appearing in the main problem specification are rendered in a discrete expression. The order of the differential operator of the original problem formulation directly dictates the number of nodes to be involved.

Here, the main drawback of finite differences can already be seen. The association of physical field values only to points cannot handle higher dimensional geometrical objects. Furthermore, the n -point discretization given by the order of the original formulation only includes the logically direct orthogonal neighbors while the other neighbors are neglected, depicted in Figure 3.8.

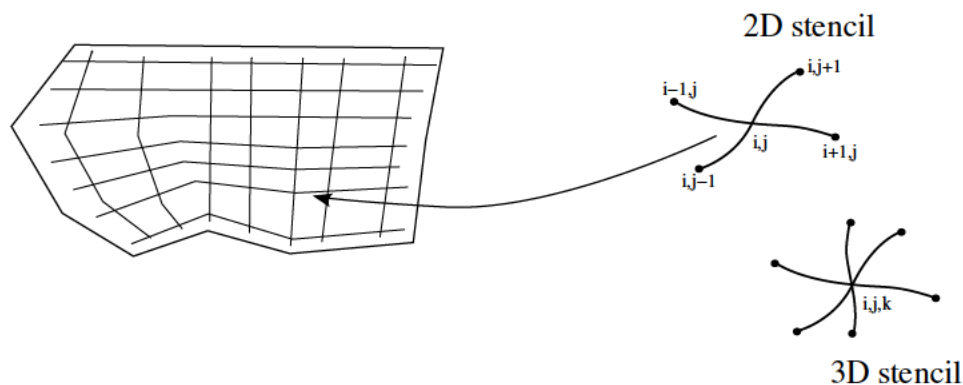


Figure 3.8: Finite difference requirements for the mesh.

The advantages of this method are that it is easy to understand and to implement, at least for simple material relations. The finite difference method optimizes the approximation for the differential operator in the central node of the considered patch. Enhancements related to the use of non-orthogonal grids and the low order of accuracy were developed but have not proven successful.

3.4.1 Basic Concepts

The derivatives of the partial differential equation are approximated by linear combinations of function values at the structured grid points. Arbitrary order approximations can be derived from a Taylor series

expansion:

$$u(x) = \sum_{n=0}^{\infty} \frac{(x - x_i)^n}{n!} \left(\frac{\partial^n u}{\partial x^n} \right)_i, \quad u \in C^\infty \quad (3.48)$$

A geometric interpretation of the different equations is shown in Figure 3.9.

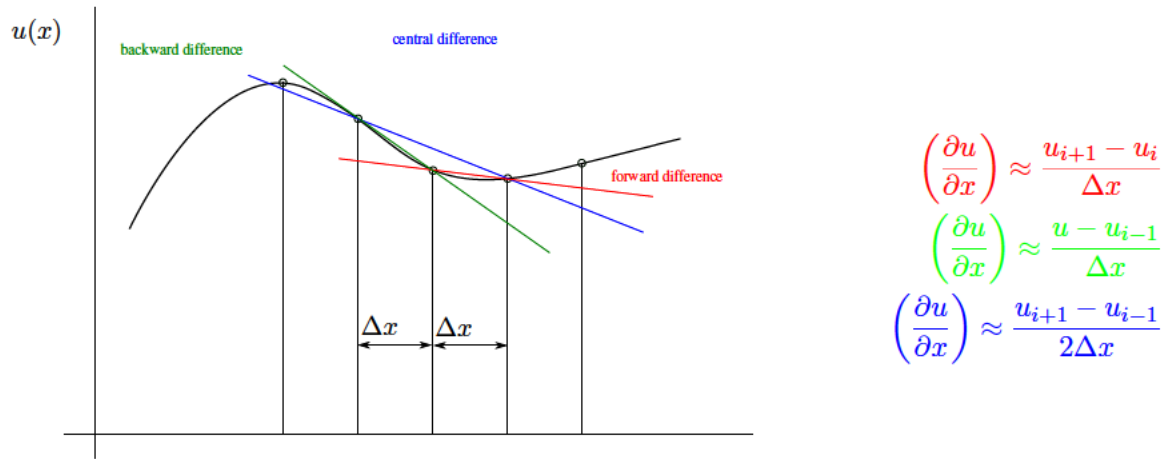


Figure 3.9: Different geometric interpretations of the first-order finite difference approximation related to forward, backward, and central difference approximation.

For second-order derivatives the central difference scheme can be used:

$$T1: u_{i+1} = u_i + \Delta x \left(\frac{\partial u}{\partial x} \right)_i + \frac{(\Delta x)^2}{2} \left(\frac{\partial^2 u}{\partial x^2} \right)_i + \frac{(\Delta x)^3}{6} \left(\frac{\partial^3 u}{\partial x^3} \right)_i \quad (3.49)$$

$$T2: u_{i-1} = u_i - \Delta x \left(\frac{\partial u}{\partial x} \right)_i + \frac{(\Delta x)^2}{2} \left(\frac{\partial^2 u}{\partial x^2} \right)_i - \frac{(\Delta x)^3}{6} \left(\frac{\partial^3 u}{\partial x^3} \right)_i \quad (3.50)$$

$$T1 + T2 \Rightarrow \left(\frac{\partial^2 u}{\partial x^2} \right)_i = \frac{u_{i+1} - 2u_i + u_{i-1}}{(\Delta x)^2} + \mathcal{O}(\Delta x)^2 \quad (3.51)$$

The accuracy of the finite difference approximations is given by:

- forward difference: truncation error: $\mathcal{O}(\Delta x)$
- backwards difference: truncation error: $\mathcal{O}(\Delta x)$
- central difference: truncation error: $\mathcal{O}(\Delta x^2)$

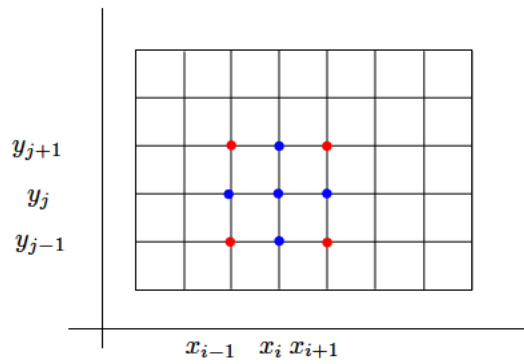
Mixed derivatives, illustrated in Figure 3.10 can be approximated, e.g., for two dimensions by:

$$\frac{\partial^2 u}{\partial x \partial y} = \frac{\partial}{\partial x} \left(\frac{\partial u}{\partial y} \right) = \frac{\partial}{\partial y} \left(\frac{\partial u}{\partial x} \right) \quad (3.52)$$

A second-order finite difference approximation for two dimensions results in:

$$\left(\frac{\partial^2 u}{\partial x \partial y} \right)_{i,j} = \frac{u_{i+1,j+1} - u_{i+1,j-1} - u_{i-1,j+1} + u_{i-1,j-1}}{4\Delta x \Delta y} + \mathcal{O}((\Delta x)^2, (\Delta y)^2) \quad (3.53)$$

Boundaries also have to be handled by the finite difference method, see Figure 3.11,



$$\begin{aligned} \left(\frac{\partial^2 u}{\partial x \partial y}\right)_{i,j} &= \frac{\left(\frac{\partial u}{\partial y}\right)_{i+1,j} - \left(\frac{\partial u}{\partial y}\right)_{i-1,j}}{2\Delta x} + \mathcal{O}(\Delta x)^2 \\ \left(\frac{\partial u}{\partial y}\right)_{i+1,j} &= \frac{u_{i+1,j+1} - u_{i+1,j-1}}{2\Delta y} + \mathcal{O}(\Delta y)^2 \\ \left(\frac{\partial u}{\partial y}\right)_{i-1,j} &= \frac{u_{i-1,j+1} - u_{i-1,j-1}}{2\Delta y} + \mathcal{O}(\Delta y)^2 \end{aligned}$$

Figure 3.10: Approximation of two-dimensional mixed derivatives.

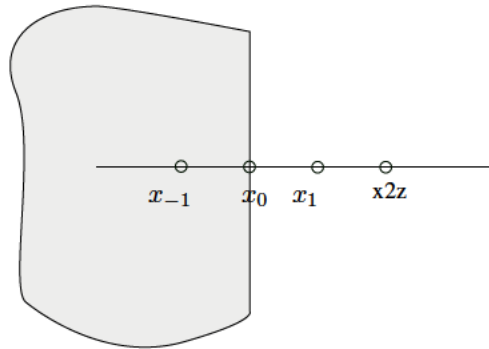


Figure 3.11: Boundary treatment for the finite difference scheme. The necessary grid points are not available for all different types of approximation schemes.

$$\left(\frac{\partial u}{\partial x}\right)_{x_0} = \frac{u_1 - u_0}{\Delta x} + \mathcal{O}(\Delta x) \quad (3.54)$$

but at the boundary on the left side, the backward and central difference approximations would need x_{-1} which is not available. Therefore treatment of boundaries is more complex for the finite difference method compared to the other discretization schemes. This particular problem grows in complexity if higher order discretization schemes are used, because more grid points are required to evaluate the corresponding approximation. Higher order approximations with finite differences are given by, e.g.:

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{2u_{i+1} + 3u_i - 6u_{i-1} + u_{i-2}}{6\Delta x} + \mathcal{O}(\Delta x)^3 \text{ backward difference} \quad (3.55)$$

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{-u_{i+2} + 6u_{i+1} - 3u_i + 2u_{i-1}}{6\Delta x} + \mathcal{O}(\Delta x)^3 \text{ forward difference} \quad (3.56)$$

$$\left(\frac{\partial u}{\partial x}\right)_i = \frac{-u_{i+2} + 8u_{i+1} - 8u_{i-1} + u_{i-2}}{12\Delta x} + \mathcal{O}(\Delta x)^4 \text{ central difference} \quad (3.57)$$

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_i = \frac{-u_{i+2} + 16u_{i+1} + 30u_i - 16u_{i-1} + u_{i-2}}{(12\Delta x)^2} + \mathcal{O}(\Delta x)^4 \text{ central difference} \quad (3.58)$$

3.4.2 Analysis of the Finite Difference Method

One method of directly transferring the discretization concepts (Section 3.1) is the finite difference time domain method. It is analyzed here related to time-dependent Maxwell equations, as was first introduced by Yee [30]. It is one of the exceptional examples of engineering illustrating great insights into discretization processes.

With this method, the partial spatial and time derivatives are replaced by a finite difference approximation. This system is solved using an explicit time evaluation. One of the main advantages of this method is that no matrix operations or algebraic solution methods have to be used.

The spatial domain is discretized by two dual orthogonal regular Cartesian grids based on cubes with spatial subdivisions of $\Delta x, \Delta y, \Delta z$, whereas the time domain is subdivided into intervals of Δt . The original formulation was based on half-step staggered grids in space and time. The quantities from the second complex are denoted with $\Delta\tilde{x}, \Delta\tilde{y}, \Delta\tilde{z}, \Delta\tilde{t}$. It is important to highlight that two different grids are necessary, due to the fact that different quantities with different orientation reside on these two distinct grids, even if in this method the secondary quantities coincide numerically with the primary ones.

Maxwell's equations, given in Section 2.2, if projected onto a regular Cartesian structured grid, yield the following six coupled scalar equations:

$$\partial_t H_x = \frac{1}{\mu} (\partial_z E_y - \partial_y E_z) \quad (3.59)$$

$$\partial_t H_y = \frac{1}{\mu} (\partial_x E_z - \partial_z E_x) \quad (3.60)$$

$$\partial_t H_z = \frac{1}{\mu} (\partial_y E_x - \partial_x E_y) \quad (3.61)$$

$$\partial_t E_x = \frac{1}{\varepsilon} (\partial_y H_z - \partial_z H_y - \gamma E_x) \quad (3.62)$$

$$\partial_t E_y = \frac{1}{\varepsilon} (\partial_z H_x - \partial_x H_z - \gamma E_y) \quad (3.63)$$

$$\partial_t E_z = \frac{1}{\varepsilon} (\partial_x H_y - \partial_y H_x - \gamma E_z) \quad (3.64)$$

These equations are the basic expressions for the finite difference time domain method (FDTD). The divergence relations are fulfilled by this method implicitly.

The components of the electric and magnetic field \mathbf{E} and \mathbf{H} with their corresponding projections to the coordinate axes are the variables used. These variables and the local values of material properties are attached to the midpoints of the grid edges. The variable indexing scheme is also used consistently with [30]

$$E_x(i\Delta x, j \pm 1/2\Delta y, k\Delta z, n\Delta t) = E_{x|i,j\pm 1/2,k}^n \quad (3.65)$$

With this notation the following expressions are obtained with central difference approximation:

$$\partial_x E_z(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{E_{z|i+1/2,j,k}^n - E_{z|i-1/2,j,k}^n}{\Delta x} + \mathcal{O}(\Delta x)^2 \quad (3.66)$$

$$\partial_t E_z(i\Delta x, j\Delta y, k\Delta z, n\Delta t) = \frac{E_{z|i,j,k}^{n+1/2} - E_{z|i,j,k}^{n-1/2}}{\Delta t} + \mathcal{O}(\Delta t)^2 \quad (3.67)$$

The time-stepping formulas for H_x are:

$$H_{x|i+1/2,j,k}^{n+1} = H_{x|i+1/2,j,k}^n + \frac{\Delta t}{\Delta z \mu_{|i+1/2,j,k}} \left(E_{y|(i+1/2,j,k+1/2)}^{n+1/2} - E_{y|(i+1/2,j,k-1/2)}^{n+1/2} \right) - \frac{\Delta t}{\Delta y \mu_{|i+1/2,j,k}} \left(E_{z|(i+1/2,j+1/2,k)}^{n+1/2} - E_{z|(i+1/2,j-1/2,k)}^{n+1/2} \right) \quad (3.68)$$

and the time-stepping for E_x :

$$E_{x|(i,j+1/2,k+1/2)}^{n+1/2} = E_{x|(i,j+1/2,k+1/2)}^{n-1/2} + \frac{\Delta \tilde{t}}{\Delta \tilde{z} \varepsilon_{|(i,j+1/2,k+1/2)}} \left(H_{y|(i,j+1/2,k+1)}^n - H_{y|(i,j+1/2,k)}^n \right) - \frac{\Delta \tilde{t}}{\Delta \tilde{y} \varepsilon_{|(i,j+1/2,k+1/2)}} \left(H_{z|(i,j+1,k+1/2)}^n - H_{z|(i,j,k+1/2)}^n \right) \quad (3.69)$$

3.4.3 Further Analysis

As already introduced, the FDTD method does not use global quantities. Instead only local nodal values of the corresponding vector values are used. Based on the initial problem formulation, it can be seen that these local values are the projections of averaged field components onto 2-cells, and therefore are local representatives of the global quantities. With local constitutive relations (only the x part is given):

$$B_{x|(i+1/2,j,k)}^n = \mu_{|(i+1/2,j,k)} H_{x|(i+1/2,j,k)}^n \quad (3.70)$$

and

$$D_{x|(i,j+1/2,k+1/2)}^{n+1/2} = \varepsilon_{|(i,j+1/2,k+1/2)} E_{x|(i,j+1/2,k+1/2)}^{n+1/2} \quad (3.71)$$

the following expressions are obtained using global quantities:

$$\Delta y \Delta z \mu_{|(i+1/2,j,k)} H_{x|(i+1/2,j,k)}^{n+1} = \Phi_{B,x|(i+1/2,j,k)}^{n+1} \quad (3.72)$$

$$\Delta y \Delta t E_{y|(i+1/2,j,k\pm 1/2)}^{n+1/2} = \Phi_{E,y|(i+1/2,j,k\pm 1/2)}^{n+1/2} \quad (3.73)$$

$$\Delta z \Delta t E_{z|(i+1/2,j\pm 1/2,k)}^{n+1/2} = \Phi_{E,z|(i+1/2,j\pm 1/2,k)}^{n+1/2} \quad (3.74)$$

and

$$\Delta \tilde{y} \Delta \tilde{z} \varepsilon_{|(i,j+1/2,k+1/2)} E_{x|(i,j+1/2,k+1/2)}^{n+1/2} = \Psi_{D,x|(i,j+1/2,k+1/2)}^{n+1/2} \quad (3.75)$$

$$\Delta \tilde{z} \Delta \tilde{t} H_{z|(i,j+1,k+1)}^n = \Psi_{H,z|(i,j+1,k+1)}^n \quad (3.76)$$

Rewriting Equation 3.72 and Equation 3.76 yields:

$$H_{x|(i+1/2,j,k)}^n = \frac{1}{\Delta y \Delta z \mu_{|(i+1/2,j,k)}} \Phi_{B,x|(i+1/2,j,k)}^n \quad (3.77)$$

$$H_{x|(i+1/2,j,k)}^n = \frac{1}{\Delta \tilde{y} \Delta \tilde{t}} \Psi_{H,x|(i+1/2,j,k)}^n \quad (3.78)$$

which can be expressed for Φ and Ψ . For Φ it reads:

$$\frac{\Phi_{B,x|(i+1/2,j,k)}^n}{\Delta y \Delta z} = \mu_{|(i+1/2,j,k)} \frac{\Psi_{H,x|(i+1/2,j,k)}^n}{\Delta \tilde{x} \Delta \tilde{t}} \quad (3.79)$$

while the result for Ψ is:

$$\frac{\Psi_{D,x|}^{n+1/2}(i,j+1/2,k+1/2)}{\Delta\tilde{y}\Delta\tilde{z}} = \varepsilon_{|(i,j+1/2,k+1/2)} \frac{\Phi_{E,x|}^{n+1/2}(i,j+1/2,k+1/2)}{\Delta x \Delta t} \quad (3.80)$$

Therefore this equation represents a discrete constitutive equation of the simplest type, obtained by extending the local constitutive equations $\mathbf{B} = \mu\mathbf{H}$, and $\mathbf{D} = \varepsilon\mathbf{E}$ with the assumption of planarity, regularity, and orthogonality of the cells.

Examining the time-stepping formulae for H_x , the time-stepping for $\Phi_{B,x}$ becomes:

$$\begin{aligned} \Phi_{B,x|(i+1/2,j,k)}^{n+1} &= \Phi_{B,x|(i+1/2,j,k)}^n + \\ &\quad \Phi_{E,y|(i+1/2,j,k+1/2)}^{n+1/2} - \Phi_{E,y|(i+1/2,j,k-1/2)}^{n+1/2} - \\ &\quad \Phi_{E,z|(i+1/2,j+1/2,k)}^{n+1/2} + \Phi_{E,z|(i+1/2,j-1/2,k)}^{n+1/2} \end{aligned} \quad (3.81)$$

and from the E_x and the corresponding $\Psi_{D,x}$ is obtained:

$$\begin{aligned} \Psi_{D,x|(i,j+1/2,k+1/2)}^{n+1/2} &= \Psi_{D,x|(i,j+1/2,k+1/2)}^{n-1/2} - \\ &\quad \Psi_{H,y|(i,j+1/2,k+1)}^n + \Psi_{H,y|(i,j+1/2,k)}^n + \\ &\quad \Psi_{H,z|(i,j+1,k+1/2)}^n + \Psi_{H,z|(i,j,k+1/2)}^n \end{aligned} \quad (3.82)$$

The expressions can then be transformed into equations depending only on two global values:

$$\begin{aligned} \Phi_{B,x|(i+1/2,j,k)}^{n+1} &= \Phi_{B,x|(i+1/2,j,k)}^n + \frac{\Delta x \Delta t}{\Delta\tilde{y}\Delta\tilde{z}} \\ &\quad \left(\frac{1}{\varepsilon_{|(i+1/2,j,k+1/2)}} \Psi_{D,x|(i+1/2,j,k+1/2)}^{n+1/2} - \frac{1}{\varepsilon_{|(i+1/2,j,k-1/2)}} \Psi_{D,x|(i+1/2,j,k-1/2)}^{n+1/2} - \right. \\ &\quad \left. \frac{1}{\varepsilon_{|(i+1/2,j+1/2,k)}} \Psi_{D,x|(i+1/2,j+1/2,k)}^{n+1/2} + \frac{1}{\varepsilon_{|(i+1/2,j-1/2,k)}} \Psi_{D,x|(i+1/2,j-1/2,k)}^{n+1/2} \right) \end{aligned} \quad (3.83)$$

For the special case of a dominant magnetic system, TM-mode, the following expressions can be derived:

$$\partial_t H_x = \frac{1}{\mu} (\partial_z E_y - \partial_y E_z) \quad (3.84)$$

$$\partial_t H_y = \frac{1}{\mu} (\partial_x E_z - \partial_z E_x) \quad (3.85)$$

$$\partial_t E_z = \frac{1}{\varepsilon} (\partial_x H_y - \partial_y H_x - \gamma E_z) \quad (3.86)$$

With these expressions the TM-mode at $t = n\Delta t$ is discretized by:

$$\partial_t H_x = \frac{1}{\mu} (\partial_z E_y - \partial_y E_z) \quad (3.87)$$

$$\partial_t H_y = \frac{1}{\mu} (\partial_x E_z - \partial_z E_x) \quad (3.88)$$

$$H_{x|i,j,k}^{n+1/2} - H_{x|i,j,k}^{n-1/2} = \frac{\Delta t}{\mu\Delta y} \left(E_{z|i,j+1/2,k}^n - E_{z|i,j-1/2,k}^n \right) \quad (3.89)$$

$$H_{y|i,j,k}^{n+1/2} - H_{y|i,j,k}^{n-1/2} = \frac{\Delta t}{\mu\Delta x} \left(E_{z|i+1/2,j,k}^n - E_{z|i-1/2,j,k}^n \right) \quad (3.90)$$

For a dominant electric system, the TE-mode is given by:

$$\partial_t E_x = \frac{1}{\varepsilon} (\partial_y H_z - \partial_z H_y - \gamma E_x) \quad (3.91)$$

$$\partial_t E_y = \frac{1}{\varepsilon} (\partial_z H_x - \partial_x H_z - \gamma E_y) \quad (3.92)$$

$$\partial_t H_z = \frac{1}{\mu} (\partial_y E_x - \partial_x E_y) \quad (3.93)$$

With these expressions the TE-mode at $t = (n + 1/2)\Delta t$ is discretized by:

$$\partial_t E_z = \frac{1}{\varepsilon} (\partial_x H_y - \partial_y H_x - \gamma E_z) \quad (3.94)$$

$$E_{z|i,j,k}^{n+1} - E_{z|i,j,k}^n = \frac{\Delta t}{\varepsilon} \left(\frac{H_{y|i+1/2,j,k}^{n+1/2} - H_{y|i-1/2,j,k}^{n+1/2}}{\Delta x} - \frac{H_{x|i,j+1/2,k}^{n+1/2} - H_{x|i,j-1/2,k}^{n+1/2}}{\Delta y} - \gamma E_{z|i,j,k}^{n+1/2} \right) \quad (3.95)$$

As can be seen, the E_z values for the full time-steps and the H_x , H_y for half time-steps are already available, and the the update for H_x , H_y can be done without any further calculation. On the contrary $E_{z|i,j,k}^{n+1/2}$ is not available and has to be approximated by:

$$E_{z|i,j,k}^{n+1/2} = \frac{1}{2} (E_{z|i,j,k}^{n+1} + E_{z|i,j,k}^n) \quad (3.96)$$

From this the following expression is finally obtained:

$$E_{z|i,j,k}^{n+1} = \frac{1 - \frac{\gamma\Delta t}{2\varepsilon}}{1 + \frac{\gamma\Delta t}{2\varepsilon}} E_{z|i,j,k}^n + \frac{\frac{\Delta t}{\varepsilon}}{1 + \frac{\gamma\Delta t}{2\varepsilon}} \left(\frac{H_{y|i+1/2,j,k}^{n+1/2} - H_{y|i-1/2,j,k}^{n+1/2}}{\Delta x} - \frac{H_{x|i,j+1/2,k}^{n+1/2} - H_{x|i,j-1/2,k}^{n+1/2}}{\Delta y} \right) \quad (3.97)$$

As can be seen only the neighboring values H_x , H_y are used to evaluate the spatial derivative of E_z . Also, only the neighboring elements of H_x , H_y and E_z are used to calculate the spatial derivatives of H_x , H_y . Therefore this method calculates both fields, \mathbf{E} and \mathbf{H} , based on the $\text{curl}()$ expressions with special requirements on the given field. Note this discretization can also be derived by the global integral discretization [33], which eases the evaluation of boundary conditions.

Part II

Practical Concepts

Chapter 4

Overview

This chapter serves as a catalyst to transform the theoretical Part I into applicable software engineering concepts which are required to develop generic and reusable software components. Although the underlying ideas were presented by McIlroy at the NATO conference in 1968 [70], several programming languages were analyzed to implement reusable components, but no satisfactory language has been found [16] since then. A possible realization was given by the advent of C++ and the template mechanisms and the related generic programming paradigm. This trend met a milestone with the development of the C++ STL, the first generic template library where data structures have been separated from the algorithms. The STL systematically developed a catalog of concepts [38], started with the work of Stepanov [16] by implementing generic algorithms. The systematic study of algorithms and data structures and the possibilities of combining programming paradigms, necessary for further development of generic programming, require not only a firm theoretical base but also sufficient support within a programming language. In 1995 several requirements [71] to enable the transition of imperative application design to a generic component programming were noted:

- A software component industry is required. By standardizing the interfaces to data structures and algorithms, different component vendors can provide software implementations for special areas. Application vendors can then integrate these systems and build applications.
- One of the most fundamental questions in software design is whether design is a top-down or bottom-up activity. Generic components are a partial solution, because they already constitute the bottom layer. But the question remains when a component has to be developed.
- Another important issue is language design. A programming language suitable for a broad spectrum of applications has to offer various, sometimes contradictory, options, e.g., high performance and abstract formulation.
- Software engineering education: many of the principles of good and modern programming are, in fact, difficult to demonstrate to a beginning student. Teaching component-based programming can ease this introduction greatly.

Beside this catalog of requirements, which facilitates the identification and subsequent development of orthogonal and reusable components, a design guide specifying how to efficiently assemble generic components is needed for the development of complex applications. The benefits of orthogonal and therefore decoupled and reusable components are self evident, the design of generic libraries has been difficult [71] and has remained challenging although the STL has already demonstrated that all issues are resolvable. The evolution of languages and software technology has enabled to take the goals one step further. The quest for mathematical notations in programming languages, for example, was greatly advanced by the efforts leading to the grid algorithm library (GrAL [18]), where algorithms were specified independently of the underlying mesh. The theoretical contribution of this work was very powerful, but a complete and efficient implementation of usable software components is still missing.

This chapter identifies and decouples generic components which are then assembled in Part III into the generic scientific simulation environment. An important fact is that the identification process has not been started from scratch, but is based on the work and concepts identified and implemented during many years at our institute. The approach which has been developed and is presented here renders them in a different context for multi-paradigm theoretically driven, generic component-based programming. The building blocks are not based on low-level components, instead a small number of application-oriented building-block components is sketched. The low-level components, such as the STL and various Boost libraries [44, 72–75], are heavily used to build these blocks. Nevertheless, the deeper goal of transcending the imperative line-by-line programming paradigm towards a component programming paradigm remains. Only by relying on a component-based approach is it possible to educate the next generation of engineers in the design and use of these components. A consistent minimal base for application design can thereby be identified without restricting global applicability related to the following issues:

- The concept of mapping cell complex properties into algebraic structures using the concept of chains and projecting the physical fields by means of cochains into a dual algebraic space translates to practical concepts of generic data structures and generic quantity accessors. Generic traversal mechanisms and the identification and unification of previously developed iteration mechanisms are also derived.
- The notion of fiber bundles is used to introduce the separation of the structural components of data structures from the stored data. This is done in a neighborhood-preserving fashion. The main benefit of applying this concept in the area of application design is a clean and consistent interface and therefore a set of interoperable libraries.
- Each task requires a different programming paradigm for transformation, if an overall high performance is required. By not relying on a single programming paradigm, each concept can be transferred to the best suitable programming paradigm, and even domain-specific languages can be derived for these concepts. The necessary amount of design and implementation can thereby be drastically reduced.
- Additionally, it can be shown that only a small set of generic algorithms is required to accomplish different tasks such as discretization.

Before introducing the transition of cell complex and fiber bundle concepts, an important part of programming has to be introduced: the concept of a data model which identifies and states the possible mechanism for storing data within a computer as well as the necessary operations on them. Several programming paradigms can thereby also be identified. After introducing these concepts for programming, the transition to applied concepts is finalized by stating several application concepts as well as reviewing related work.

The language of choice is C++ for various reasons. Up to now only C++ can transform all of these concepts into code which results in an overall high performance, while also exhibiting the smallest distances between the specification of concepts and their realization [76, 77]. The STL has shown that genericity and efficiency do not have to be contradictory requirements by the concept of meta-programming [45]. The application built with generic components has to provide the same performance guarantees as with manually tuned source code. This is only realizable if each component guarantees its performance. The STL has introduced run-time constraints for each of its components.

It is important to mention that in software engineering the discussion and comparison of concepts by referral to the resulting code is necessary. Therefore code parts are not given in an abstract form but with an actual implementation in C++.

Chapter 5

Programming Concepts

5.1 Data Model for Scientific Computing

For the practical realization of the transition of the given theoretical concepts into applicable concepts, an effective data model is essential to the design and development of scientific software [78]. Data exchange and data access, whether it may be internally within an application via independent software components or externally among applications, requires defined and consistent interfaces which explicitly communicate all the properties of a given dataset. Especially by the current worldwide development of shared and open-source resources, the concept of data and even application interoperability is a growing demand. The concept of a data model, denoting a data structure with a set of operations on it, is exactly this type of concept and was first considered by F. Codd [79] by defining a relational scheme for structuring data and providing access and query operations for relational databases. So a data model is a primary consideration in the design of any software system dealing with different data occurring in scientific computing.

Four different concepts for application design are therefore identified which are required to describe scientific data. The different requirements for the interfaces between these levels are analyzed according to their relationships, as depicted in Figure 5.1.

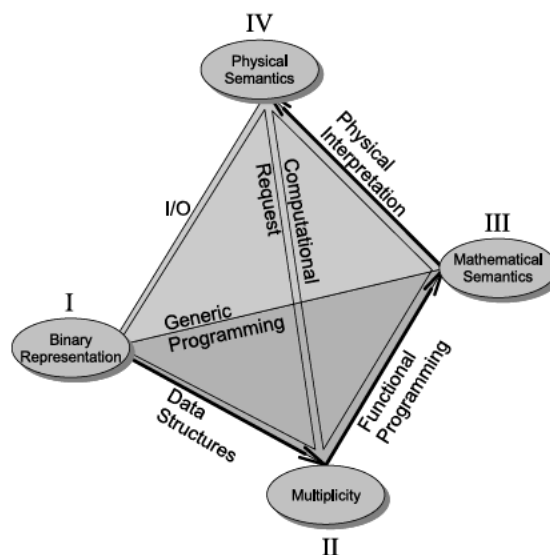


Figure 5.1: A simplex representation of a data model and the corresponding interfaces.

The affinity to various programming paradigms can be identified by the respective couplings among each of these abstraction layers, where each level intrinsically requires a special transformation into software components. Therefore each link between these levels is classified by computational issues of scientific computing. The details of each level of abstraction are reviewed next:

- I.) The level of *binary representation* is the bottom most level of each computational design. An important fact is that high-level applications must not depend on assumptions made within this level, e.g., floating point numbers and their byte representation. For a modern application design approach and the correlated rapid development in computer hardware and computer systems, it is not even allowed for high performance calculations to depend on this layer.
- II.) The semantic topological information of *multiplicity* must state how many atomic types are required to describe the domain of interest, e.g., how many floating point values are required to numerically describe the coordinates of a given domain.
- III.) *Mathematical semantics* are related to the identification of the mathematical purpose of a certain object, e.g., the separation of a tangent vector, a normal vector, or a coordinate location, which are all represented numerically by the same number of atomic types. The mathematical properties for the given objects are defined by their chart transition rules.
- IV.) The final level of *physical semantics* represents the physical interpretation of a certain data set, which has to be associated with data in order to determine its purpose, e.g., given a vector field and its corresponding role in a differential equation.

While mathematical semantics (Level III) specify which operations are possible on a certain field, the specification of which operations are meaningful (Level IV) are mostly user-driven and on the application level. When the given levels of abstraction are used, the requirements for a data model are deployed. One such possible data model was therefore suggested by Butler and Pendley [20, 21], who identified an abstraction for an object-oriented scientific data model of considerable generality based on the mathematical concept of fiber bundles (see Section 2.5). Its adoption was, until now, restricted to scientific visualization. Scientific visualization deals with results from large-scale simulations that generate a large amount of numerical data. The analysis of data has become an increasingly important research task, and various overviews and related implementations are available [1, 78, 80–82].

Butler’s approach was originally limited to vector bundles and implemented in Eiffel [21]. It was the first step towards a data model for scientific computing, but not without practical issues [1]. In this approach there was no concept of cell and complex topology (cells and skeletons) on a discretized manifold. The use of different types of p -cells is a fundamental requirement for arbitrary p -cochains, e.g., the use of cells instead of vertices is important when interpolating data at arbitrary locations.

5.2 Evolution of Programming Paradigms

The data model introduced in the last section shows that several tasks in the development of scientific application inherently require different programming paradigms by definition to model the underlying issues best.

In scientific computing there is a multitude of software tools available which provide methods and libraries for the solution of very specific problems. It is clear that such tools impose restrictions in various ways. The quest for highly reusable software components is still ongoing and demands different programming paradigms for an efficient realization.

A software component is reusable if it can be used beyond its initial use within a single application or group of applications without modification. The programming paradigms which are most widely used and implemented by various programming languages are:

- Imperative programming
- Object-oriented programming (OO)
- Functional programming (FP)
- Generic programming (GP)
- Meta-programming (MP)

These various paradigms were developed to cope with the important issue of enabling an efficient transformation of algorithms and concepts into code. One of the main issues is related to the given code reusability thereby identifying the concepts of monomorphic and polymorphic programming, one of the basic concepts for code reuse and orthogonal program development. Classical languages, such as different types of assembler or C, use a straight-forward monomorphic programming style based on the imperative programming paradigm. In the field of scientific computing, only one- and two-dimensional data structures were initially used to develop applications, due to the limitations of computer resources. The imperative programming paradigm was sufficient for this type of task. Code which is developed this way supports only single data types and cannot be reused at all.

With the improvement of computer hardware and the rise of the object-oriented programming paradigm, the shift to more complex data models was possible. Modern high-level languages such as C++ or Java [6] implement means for polymorphic programming which make code reuse possible, e.g., by inheritance. Whereas several new programming languages such as Ruby [83] and other derivatives do not bother the user with the data types and therefore offer various automatically casted types, statically typed programming languages such as C, C++, or Java offer different strategies. Implications to application development can be observed clearly by studying the evolution of the object-oriented paradigm from imperative programming. The object-oriented programming paradigm has significantly eased the software development of complex tasks, due to the decomposition of problems into modular entities. It allows the specification of class hierarchies with its virtual class polymorphism (subtyping polymorphism), which has been a major enhancement for many different types of applications. But another important goal in the field of scientific computing, *orthogonal libraries*, cannot be achieved easily by this paradigm [39]. A simple example for an orthogonal library is a software component which is completely exchangeable, e.g., a sorting algorithm for different data structures. An inherent property of this paradigm is the divergence of generality and specialization [84–86].

Thus the object-oriented programming paradigm is pushed to its limits by the various conceptual requirements in the field of scientific computing, due to problems with interface specifications, performance issues, and lack of orthogonality. Even though the trend of combining algorithms and data structures is able to provide generalized access to the data structures through objects, it is observable that the interfaces of these objects become more complex as more functionality is added. The intended generality often results in inefficiency of the programs, due to virtual function calls which have to be evaluated at run-time. Compiler optimizations such as inlining or loop-unrolling cannot be used efficiently, if at all. A lot of research has been carried out to circumvent these issues [87], but major problems arise in the details [88].

Modern paradigms, such as the generic programming paradigm [16, 89], have the same major goals as object-oriented programming, such as reusability and orthogonality. However, the problem is tackled from a different point of view [90]. Together with meta-programming [45], generic programming accomplishes both a general solution for most application scenarios and highly specialized code parts for minor scenarios without sacrificing performance [91–93] due to partial specialization. The C++ language supports this paradigm with a type of polymorphism which is realized through template programming [94], static parametric polymorphism. Combining this type of polymorphism with meta-programming, the compiler can generate highly specialized code without adversely affecting orthogonality. This allows

the programmer to focus on libraries which provide concise interfaces with an emphasis on orthogonality, as can already be found, e.g., in the the BGL [43]. Although Java has gained more functionality with respect to a multi-paradigm approach [95, 96], its performance still cannot be compared to the run-time performance of C++ [5]. Another comparison of high-level languages as well as scripting languages is also available [97]. To give a brief overview, the simple problem of an inner product is used to demonstrate the discussed issues of employing several programming paradigms:

$$v = \sum_i x_i \cdot y_i \quad (5.1)$$

A detailed performance comparison of the paradigms given here in C++ has been published [46].

5.2.1 Object-Orientation Programming

The object-oriented programming paradigm [98] introduced mechanisms required to obtain modular software design and reusability compared to universal accessibility of implementations by imperative programming. The main part of the object-oriented paradigm is related to the introduction of classes which cover basic properties of concepts to be implemented. Based on this description of properties, an *object* is created, which can be briefly explained by a self-governing unit of information which actively communicates with other objects. This is the main difference compared to a passive access as used in imperative programming languages. The concept of inheritance was introduced due to the fact that a particular concept, e.g., a mathematical n -dimensional vector, cannot be described in its generality by a single class. By inheritance, semantically related concepts can be implemented in a class from which other classes can be derived, which should enable means of code reusability.

The first source snippet presents an implementation of the given problem in a fully object-oriented language, Java.

```

public class Vector
{
    public int dimension = 0;
    public double[] value;
    public Vector(int i) { // .... }
    public double getValue(int i) { return this.value[i]; }
    public double inner_product(Vector vec)
    {
        double val = 0;
        for(int i=0; i < this.dimension; i++)
            tempval += this.getValue(i) * vec.getValue(i);
        return val;
    }
}
public class VectorTest extends Vector
{
    public VectorTest(int i) { super(i); }
    public static void main(String[] agrs)
    {
        VectorTest wvt1 = new VectorTest(3);
        VectorTest wvt2 = new VectorTest(3);
        System.out.println("output:" + wvt1.inner_product(wvt2));
    }
}

```

Object-oriented implementation in Java

As can be observed, the connection between the implementation of the algorithm and the corresponding class `Vector` is very tight. All classes and methods which want to have access to this algorithm have to interface this class structure. Otherwise, the whole class has to be transferred to an *interface*, which has to be incorporated by other classes which want to use this algorithm. Either way this approach is intrusive, which means that the application of this algorithm requires a modification of the existing code. The object-oriented modeling, as presented, is strictly narrowed down for reusability within the class hierarchy. Despite the different aids for crossing this border, the inherent restriction in this example demonstrates that pure object-orientation does not lead to fully reusable components, due to the subtype bounds [77].

Another point of view can be presented if the amount of source code is reviewed which has to be implemented to develop an application in the field of scientific computing. For the case of a multi-discretization environment, different data structures have to be implemented, see Figure 5.2, where the top line gives a brief but incomplete overview of data structures required for the finite volume and the finite element methods. The vertical line presents different types of discretization schemes with the corresponding blocks, such as higher order finite elements, or different types of interpolation schemes for the finite volume method. For the object-oriented paradigm, all blue rectangles have to be implemented, e.g., finite volume with nonlinear interpolation functions.

		Structured				Unstructured		
		1D	2D	3D	ND	1D	2D	3D
Finite Volume	LSF	■	■	■		■	■	■
	SG	■	■	■		■	■	■
		■	■	■		■	■	■
Finite Element	P=1	■	■	■		■	■	■
	P=2	■	■	■		■	■	■
	P=3	■	■	■		■	■	■
	P=n	■	■	■		■	■	■

Figure 5.2: Application design based on the object-oriented programming paradigm. Each of the inner blocks has to be implemented, resulting in an implementation effort of $\mathcal{O}(D \cdot A)$, where D stands for the number of data structures and A for the number of algorithms.

For a general view of this paradigm, the implementation effort is estimated by $\mathcal{O}(D \cdot A)$, where D stands for the number of data structures and A for the number of algorithms. Therefore, libraries for scientific computing, implemented by an object-oriented paradigm with a currently available run-time dispatch mechanism are not suitable for an efficient implementation, despite the performance penalty of object-oriented languages [46].

5.2.2 Functional Programming

A different type of programming paradigm is functional programming, where computation is treated as the evaluation of functions based on the following properties:

- Objects are provided only as constants or as expressions on objects.
- Functional programming avoids states of objects and mutable data. Loops therefore have to be replaced by recursion.

An important part of functional programming related to high performance is the delayed or lazy evaluation. This means that the moment in which a function is bound with some of its arguments can be separated from the one in which it is evaluated, thereby supporting the following techniques:

- Partial binding and partial evaluation, e.g., a binary function where one of the arguments is already bound to the function and only one parameter has to be specified to finally evaluate the function.
- Lambda expressions, based on the lambda calculus [99], use partial binding to define the expression $\lambda xy.fxyz$, with the reserved prefix λ , followed by the formal parameters, a dot, and the function definition.
- Currying, which creates unnamed lambda expressions automatically.

An implementation of the given problem is presented in the following source snippet using Haskell [7] as the host language. This programming paradigm, with the corresponding purely functional programming languages, such as Haskell, is currently widely used in the area of mathematical modeling. The static type mechanism and the generic mechanism, e.g., the availability of return type deduction, which goes beyond the capabilities of, e.g., C++, is an important feature and advantage of this language.

```
inner_product :: [[Integer]] -> Integer
inner_product = foldr (+) 0 . map (foldr (*) 1) . transpose

transpose :: [[a]] -> [[a]]
transpose [] = []
transpose ([]:xss) = transpose xss
transpose (x:xs) : xss = (x : [h | (h:t) <- xss]) :
                        transpose (xs : [t | (h:t) <- xss])

val = inner_product x
```

Functional programming in Haskell

As can be seen, this representation of the algorithm is completely generic for all different types of input sequences, which can be added, folded, and transposed. The drawback of this mechanism is that it is already detached from the conventional programming style of C, Java, C++, or C# [3]. The functional modeling is fully reusable, because no assumptions about the used data types are made. Pure functional programming languages, such as Haskell, are completely polymorphic, concept-based programming languages. However, the issues with these languages are briefly explained by two drawbacks. First, the compiler has to do a lot of optimization work to reach the excellent performance of other programming languages such as Fortran or C++ [46, 77, 100]. Second, some tasks, such as input/output operations are inherently not functional and therefore difficult to model in such a programming languages. These parts of an application are always given by different states due to the fact that data sources, e.g., hard-disk or memory, are state-based. Functional programming does not support states from the beginning.

C++'s compile-time programming supports only the functional programming style. However, C++ is not a functional programming language, e.g., operators are not functions and cannot be used as first-class objects, but C++ allows the simulation of functional behavior. Functional programming therefore eases the specification of equations and offers extendable expressions, while retaining the functional dependence of formulae by lambda expressions (higher order functions). The C++ language uses additional libraries to support this paradigm [44, 101, 102].

5.2.3 Generic Programming

Generic programming denotes (object-oriented) programming methods based on the mechanism of parametric polymorphism of data structures and algorithms to obtain orthogonality, while avoiding the performance drawback of virtual function call costs [103] in C++. The generic programming paradigm thereby establishes homogeneous interfaces between algorithms and data structures without sub-typing polymorphism by an abstract access mechanism, the STL iterator concept. This concept specifies an abstract mechanism for accessing data structures in a concise way.

```

template<typename InputIterator1, typename InputIterator2, typename NumericT>
NumericT inner_product(InputIterator1 start1,
                      InputIterator1 end1,
                      InputIterator2 start2,
                      NumericT startval) const {
    for (; start1 != end1; ++start1, ++start2)
        startval += ( *start1 * (*start2) );
    return startval;
}
// ..
vector_type vec1, vec2;
vector_type::numeric_t val;

val = inner_product(vec1.begin(), vec1.end(), vec2.begin(),
                  vector_type::numeric_t(0));

```

Generic programming in C++

This algorithm can be used with any data structure which offers iterator access, whereby the most simple implementation of an iterator is a pointer to the data storage of a data structure. Based on the indirection of the iterator concept the overall implementation effort is greatly reduced. Problems arise in the details of the requirements of the iterators themselves, which are discussed in Appendix B. Nevertheless, an overall high performance can be obtained.

		Structured				Unstructured		
		1D	2D	3D	ND	1D	2D	3D
Finite Volume	LSF	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	SG	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Finite Element	P=1	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	P=2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	P=3	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
	P=n...	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 5.3: Application design based on the generic programming paradigm. Only the topmost and leftmost parts have to be implemented, which reduces the implementation effort to $\mathcal{O}(D + A)$.

In contrast to the object-oriented programming paradigm, the generic programming paradigm supports an orthogonal means for application design with the separation of algorithms and data structures, connected by the concept of iterators, or more generally, by traversal and abstract data accessors.

The implementation effort-combined with functional programming, which is greatly supported by the generic programming paradigm-is thereby reduced to $\mathcal{O}(D + A)$ and illustrated in Figure 5.3.

Generic modeling, together with the corresponding programming paradigm, supports both the derivation and state model mechanisms of object-oriented modeling, with the fully polymorphic support of functional programming. In other words, the generic paradigm supports the state-driven input/output with iterator or traversal concepts and the functional specification mechanism of functional programming languages. It can thereby be seen as the connector between the data sources and the specification. Not only can the implementation effort be reduced by several orders of magnitudes, but also be used to implement run-time code with the highest performance [46].

5.2.4 Meta-Programming

The paradigm of meta-programming can be briefly explained as writing a computer program that writes or manipulates other programs or themselves. For programming tasks which have to be repeated a great number of times, this paradigm can be used to obtain an efficient implementation [50, 100].

Template meta-programming is a type of meta-programming in which the template (parametric polymorphism) system of the host language is used. The compiler uses these templates to generate new source code at compile-time. Template meta-programming is currently implemented by various programming languages, such as C++, Haskell, or D [4]. Here, only the implementation of this paradigm in C++ is used. C++ implements a Turing-complete template system, which means that any computation is expressible. In contrast to run-time programming, template meta-programming has non-mutable variables and therefore requires the functional programming paradigm, most often used by recursion patterns.

The main advantages and disadvantages of this paradigm in C++ are:

- The generic programming paradigm is supported through template meta-programming by automatic code generation for various generic data structures or algorithms, e.g., the parametrization of data types for a simple container structure. Thereby truly generic code, facilitating code minimization and maintainability, is available.
- Due to the processing of the template system at compile-time, compilation can take a large amount of time. A careful selection of compile-time libraries and run-time libraries is therefore of utmost importance to implement large software projects. The most important benefit of this tradeoff is that the extended compile-times are used to obtain a highly efficient run-time code, sometimes superior even to Fortran implementations. See Section 8.4 for more details.
- One of the most severe drawbacks is the poor readability of template meta-programms. The operator overloading mechanism of C++ reduces this problem, but some operators cannot be overloaded, thus different language elements which cannot be easily identified as C++ source code are generated.
- Another drawback of template meta-programming is the reduced portability. However, especially the GNU compiler collection (GCC [104]) implementation of the C++ standard has significantly increased the availability of C++ on various platforms thereby reducing the portability problem in the future.

Together with generic programming, C++ template meta-programming enables another important concept for highly expressive code: domain-specific embedded languages.

5.3 Domain-Specific Embedded Language

Closely related to the programming paradigms which have been discussed above is an emerging concept for high-level languages, such as C++, domain-specific embedded languages.

First, a domain-specific language (DSL) is defined by a set of symbols as well as by a well-defined set of rules specifying how the symbols are used to build well-formed compositions [45]. The domain-specific part of a DSL can then be employed by a simplified grammar and an increased expressiveness. The possibility of writing code in terms close to the level of abstraction of the initial problem domain is the characteristic property of DSLs.

The next relevant issue regarding a DSL is interoperability. A possible way of dealing with interoperability is to integrate the domain-specific language into a host language, finally obtaining a domain-specific embedded language (DSEL), e.g., YACC [105]. All software engineering tasks, such as designing, implementing, and maintaining the DSL, are reduced to implementing and maintaining a library of the host language. And more importantly, the interoperability can be enhanced to the highest level, due to the fact that both concepts are now available in one host language. All the libraries already developed and their functionality are available at the same time. To summarize, the advantages of domain-specific embedded languages are:

- Abstracting the underlying language/system/compiler in the direction of the user/domain expert.
- The overhead of learning or adopting a new language is greatly reduced.
- Reduction of documentation due to expressive names and self-documentation.
- Validation of semantics, e.g., by a compiler.

The advent of having multiple paradigms available in a single programming language creates new possibilities for domain-specific languages. The advantages and disadvantages of several programming paradigms and the reduction of specification and implementation effort clearly suggest the use of domain-specific embedded languages, such as reducing the stated implementation effort of the object-oriented and generic programming paradigm to $\mathcal{O}(1 + 1)$. This means that the effort of building applications can be reduced to a constant effort by utilizing a DSEL without additional transformation steps. Maintenance and further development for an extra transformation tool is thereby greatly reduced, compared to, e.g., the transformation tool for Sophus [106]. But up to now, the implementation of a DSEL for non-trivial areas is far from simple or user-oriented. Also, domain-specific languages require various mechanisms from the host languages, where the following features are almost mandatory:

- *operator overloading* means that different operators, such as $+$, $-$ can be overloaded.
- *parametric polymorphism* means that a kind of template system is available.
- *functional mechanism* means that higher order and lambda objects are available.
- *time of evaluation* means that program code can be specified for compile and for run-time.

Language	operator overloading	parametric polymorphism	functional mechanisms	time of evolution
C	no	partial (macros)	no	compile/run-time
D	partial	yes	no	compile/run-time
Java	no	partial (version)	no	run-time
C#	no	partial (version)	no	run-time
Haskell	yes	partial (version)	yes	run-time
C++	yes	yes	yes	compile/run-time

Table 5.1: Language comparison.

As can be seen from the brief comparison of languages provided in Table 5.1, only a few languages are available which allow an efficient modeling of operators. Languages such as Haskell were not developed

with an overall high performance approach, but they still offer the definition of new operators which extend the built-in language syntax. Based on this overview, the only language offering the best support for DSELs with syntactic expressiveness as well as run-time efficiency is C++. In addition to the great possibilities of using C++ as a host language for a DSEL, the multi-paradigm approach offers a great degree of freedom in the implementation of high performance scientific code and even applications [107, 108]. The main feature is the paradigm of template meta-programming of C++, introduced in Section 5.2, which means that a DSEL can function as an extension to the general-purpose host language. Meta-programming and DSEL are also actively developed areas [48, 49]. An example is given next which illustrates the embedded nature of a grammar specification in C++. The first part yields the normal grammar specification by YACC:

```

group      ::= '(' expression ')'
factor    ::= integer | group
term      ::= factor (('*' factor) | ('/' factor))*
expression ::= term (('+' term) | ('-' term))*

```

Next, an example of Spirit [75] is given, where Spirit is an object-oriented, recursive-descent parser and output generation framework enabling target grammars written entirely in C++ as a DSEL.

```

group      = '(' >> expression >> ')';
factor    = integer | group;
term      = factor >> * (('*' >> factor) | ('/' >> factor));
expression = term >> * (('+' >> term) | ('-' >> term));

```

Another advantage of the DSEL concept is the further development of compiler technology, which drastically increases the performance of high-level code. Recent performance analysis has shown that each compiler generation increases the overall run-time performance [109].

5.4 Concept Development and Related Work

In the last decade many approaches towards implementing a general purpose simulation environment for the solution of partial differential equations have been taken. Most of the tools resulting from these attempts use topological structures which are specialized to work with a particular discretization scheme. This reduces resource use but comes at the cost of greatly diminishing the flexibility of topological traversal. As an example, the finite volume method does not require vertex-face traversal. However, for some reasons it might be advantageous to implement discretization schemes based on a mixed finite element/finite volume scheme which requires such traversal operations. The following brief overview shows some significant steps towards a more flexible and generic application design:

- 1992: a generic data model based on the concept of fiber bundles [21]
- 1993: generic application for scientific visualization based on fiber bundles [80]
- 1995: multi-paradigm language's run-time performance (C++) equal to Fortran [100]
- 1997: emergence of the generic programming paradigm by C++'s STL [17]
- 2002: generic data structure interface and generic traversal operations [51]
- 2002: generic graph library [43] with the introduction of the concept of a property map
- 2003: concepts for separation of traversal and data access [110]

Modern scientific computing requires a huge number of different physical models and numerical methods. Increasingly complex physical formulations are required by today's simulation problems. Flexibility of the definition of models as well as of numerical solution methods is required. But the transformation of a physical model into a computer model can usually be seen as a very time-consuming procedure. Therefore a great number of different applications have been developed during the last decades.

First, TCAD-related applications and environments that have been developed at the Institute for Microelectronics [11–13, 111, 112] are reviewed and the choices made with respect to the field of scientific computing are investigated¹. Then generic libraries which are directly linked to this work are also reviewed.

5.4.1 STAP, SCAP

The SAP tools have been developed for high performance interconnect analysis for TCAD [113, 114], especially for thermal (STAP) and capacitive (SCAP) problems. The finite element discretization scheme is utilized to implement highly accurate capacitance extraction, resistance calculation, transient electric and coupled electro-thermal simulations. Simple mesh generation packages for data layout as well as visualization packages were additionally developed.

All algorithms and solving procedures were hand-optimized by experienced C specialists and domain experts. This yields excellent performance and portability but requires a considerable amount of maintenance by the development group. The next generation of programmers or scientists faced an ever-increasing period of adjustment and learning time, thereby strongly reducing the enhancement capabilities. The imperative programming methods used do not support the concept of reuse by themselves; therefore reusability considerations were almost completely left to the application designer, which results in missing expressiveness of the code and highly optimized code parts can barely be extended.

5.4.2 Minimos-NT

Minimos-NT [13, 115–117] is the successor to the well-known specialized Minimos [111] and implements concepts for a general-purpose, multi-dimensional semiconductor device simulator. Minimos-NT provides steady-state, transient, and small-signal analysis of arbitrary device structures as well as mixed-mode device and circuit simulations based on compact models.

Minimos-NT is a composition of the imperative and object-oriented paradigms. They were used, on the one hand, to ease the rapid prototyping of new models, and, on the other hand, to facilitate code reuse. Unfortunately the employment of these programming paradigms complicates a clean interface specification and enforces a monolithic application design.

5.4.3 AMIGOS, FEDOS

The analytical model interface and general object-oriented solver (AMIGOS [118]) was developed to meet the requirements of a general partial differential equation solver. A common kernel for physical modeling, parameter and model hierarchies, grid-adaptation, a numerical solver, a simple front-end controller, and geometry and boundary definitions were defined. AMIGOS can handle different discretization schemes, such as finite elements and finite volumes.

FEDOS [69], the successor to AMIGOS, is a finite-element-based simulator, which uses a general nucleus matrix assembly procedure. It is intended to deal with oxidation and diffusion phenomena and also features integrated mesh adaptation capabilities.

AMIGOS and FEDOS are highly flexible and great achievements toward generic application design, especially AMIGOS with its integrated specification language can reduce development times significantly. Investigations indicate [118] that the development time of a new application is reduced from days when using SAP [113, 114] to hours when using AMIGOS. Therefore the approach of an embedded language

¹A detailed related work presentation concerning currently available and state-of-the art tools in the area of TCAD's process and device simulation is given in Section 3 of [10]

is still an important feature for generic application design. Nevertheless, the object-oriented programming paradigm which was used, as given in Section 5.2, greatly reduces reusability of the developed code and drastically increases the effort necessary to develop and maintain source code.

5.4.4 Wafer-State Server

The Wafer-State Server aimed to define and implement concepts for a generic data model suitable to TCAD's device and process simulation [119], because there was no standard available for data transfer between tools. All the solutions are based on a file format instead of a data format with a corresponding data model, which has the major drawback of being easily semantically incompatible. Based on the analysis of various approaches and the issues arising from coupling different kinds of process simulators and to allow an efficient data exchange, the Wafer-State data model has been developed. The data model also specifies various geometrical and topological operations, such as interpolation mechanisms and consistency checks for simplex objects, especially in three dimensions. Additionally, optimization tasks as they occur in the simulation of semiconductor devices were investigated and algorithms for the Wafer-State-Server were introduced to aid complex high-level simulation tasks.

The focus on the object-oriented programming paradigm is again a major drawback, as it circumvents library-centric application design. The single components of the Wafer-State server cannot be reused directly. Modules always have to be transformed and partially rewritten, because interfaces are only consistent within a class hierarchy, thereby reducing orthogonal application design. The overhead for code development is also apparent.

5.4.5 Boost Graph Library

A major step towards a more flexible use of data structures was developed by the Boost Graph Library (BGL [43]). This library implements a generic interface to enable access to arbitrary graph structures but hides the details of the actual implementation. The interfaces make it possible for any graph library that implements these interfaces to be interoperable with the BGL. The approach is similar to the one taken by the C++ STL to ensure the interoperability of the various algorithms and containers. The property map concept [43] was also introduced. Unfortunately the BGL was designed for graphs only and neither lower nor higher dimensional data structures can be handled.

5.4.6 Computational Geometry Algorithm Library

The Computational Geometry Algorithm Library (CGAL [120]) is another important collection of reusable components for a great number of geometrical algorithms and data structures in a generic library-centered approach, such as two- and three-dimensional modules for mesh generation, Voronoi diagrams, and surface mesh simplification. The main contribution of CGAL is the concept of an algebraically parametrized kernel [121] related to the actual implementation robustness of mathematical operations.

5.4.7 Grid Algorithms Library

The Grid Algorithms Library (GrAL [18]) was one of the first contributions to the unification of data structures of arbitrary dimensions for the field of scientific computing. A common interface for grids with dimensionally and topologically independent access and traversal was designed. Mathematical concepts for topological spaces were introduced and applied to grids. Applications for the field of solving PDEs were presented, but no concrete implementation was given. Concepts of accessing data abstractly, similarly to property maps, were also introduced and demonstrated in applications.

5.4.8 Further Related Work

Several research groups have put a lot of effort into the development of libraries for sub-problems occurring in scientific computing.

FiberLib2 was inspired by Butler's vector bundle data model [21] and OpenDX and is a reimplementation of a data model which was originally conceived [1] for visualizing numerical data originating from general relativity. Since the mathematics of general relativity require explicit treatment of otherwise implicitly assumed properties of space and time, designing a data model to cover the issues of general relativity improves its genericity.

The **Sophus C++** library [122] aims at coordinate-free formulations. This library implements grid components for sequential and parallel high performance computing. A field layer for numerical discretization schemes, a tensor layer to handle various quantities related to a coordinate systems, and finally an application layer with solver interfaces were developed. However, this approach suffers from severe abstraction penalties and requires a code transformation tool [106].

Prophet [123] is an environment for the solution of PDEs. It introduces four different levels of abstraction. The first and base library layer consists of a database for models of coefficients and material parameters, macros for expressions, grid routines, and a linear solver. The PDE layer consists of an assembly control, discretizations, and modeling modules. The third layer consists of several modules, for example modules for solving. The fourth layer is the user layer, which consists of an input parser and the visualization mechanism. Prophet separates geometric and physical information and was originally developed for semiconductor process simulation.

deal.II [124] provides a framework for finite element methods and adaptive refinement for finite elements. It uses modern programming techniques of C++ and enables the treatment of a variety of finite element schemes in one, two, or three spatial dimensions and of time-dependent problems. Modern finite element algorithms, using, among other aspects, sophisticated error estimators, and adaptive meshes can be developed easily. This approach is specialized to finite element discretization schemes and cannot be used easily for other schemes, e.g., finite volumes or finite differences.

Femster [125] is a class library for finite element calculations based on differential forms. This means that users must provide their own code to assemble global FE matrices. In other words, Femster implements a general finite element API.

The **FEniCS** project [126], which is a unified framework for several tasks in the area of scientific computing, is a great step towards generic modules for scientific computing. Up to now most of the modules are in a prototype state.

Various toolkits have also been developed which can be seen as related work as well. The **Generative Matrix Computation Library** (GMCL [127]) is a framework based on expression templates, generative C++ programming idioms, and many template meta-programming facilities, e.g., control structures for static meta-programming. The **Template Numerical Toolkit** (TNT [128]) is a collection of interfaces and reference implementations of numerical objects (matrices) in C++. The toolkit defines interfaces for basic data structures, such as multidimensional arrays and sparse matrices, commonly used in numerical applications.

These libraries are an important step towards library-centric application design. But most of these libraries were not developed with interoperability as a necessary constraint. As a consequence, additional code has to be introduced which slows the development process down and impedes the execution speed of the final application. The environments reviewed here also completely veil the topological information through the use of formalisms such as element matrices [129] and control functions [123].

Chapter 6

Application Concepts

The data model which has been introduced (Section 5.1) as well as the corresponding programming paradigms (Section 5.2), are of utmost importance to develop application concepts where the theoretical concepts state what has to be implemented and the practical concepts given here guide the translation of how these concepts can be implemented efficiently revealing and conserving the theoretical character.

The related work presented in Section 5.4 reviewed the developed and implemented concepts, where two fundamental elements surfaced. The first part is related to a common classification of data structures suitable for scientific computing. In order to allow an arbitrary number of nested traversal operations, which is often required in applications, it is necessary to have appropriate data structures for traversal. This has the added benefit of keeping the program code as concise as possible. The traversal mechanism is derived from the given boundary (coboundary) operator. Therefore, the first part deals with a formal interface to several data structures based on cell complex properties and topological attributes.

The second part deals with the implementation of the concept of fiber bundles suggesting the separation of topological base space properties and quantity-related structure information. The base space properties are thereby identified by the generic data structure and traversal capabilities (chain concept), whereas the quantity properties are related to p -forms and their cochain counterpart. An identification with well-known C++ concepts is given. The concept of a DSEL as a missing part from the given related work section is used to enable highly expressive implementations. Finally, the library-centric software design approach is introduced and a catalog of essential requirements is assembled.

6.1 Generic Data Structures

The concepts given in Section 2.3, especially the cell topology and complex topology, are used here to derive a common specification of an interface to data structures. This specification is based on topological properties only and thereby separates the actual data storage structure from the stored data. In terms of the fiber bundle theory, generic data structures can be seen as modeling the base space. By the identification of the chain and cochain concepts (Section 2.4) within the fiber bundle concept (Section 2.6), the concepts presented here can also be seen as models for the chain concept with the intrinsic fibers of incidence information. The here given approach collects the incidence numbers [31, 35] as well as the connection matrix [53].

The first step towards a generic data structure is the integration of the various already existing data structures. For example, Figure 6.1 presents the structure of a singly linked list.

In terms of topological concepts, only locally neighboring cells can be traversed by this data structure, which can be seen by the representation of the complex topology by the poset on the right side. The

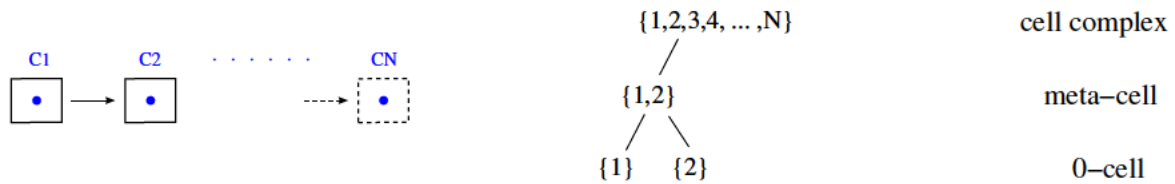


Figure 6.1: Complex topology of a singly linked list.

number of elements in the meta-cell subsets is bounded by a constant number, unique for each type of data structure in the 0-cell complex regime, whereas the concept of a meta-cell is introduced in Section 2.3.3. The singly linked list uses two elements in the meta-cell, later declared as `local (2)` and called `forward` concept in the STL. The mechanisms used to describe the doubly linked list are the same as those used for the singly linked list. Compared to the singly linked list, three elements in the meta-cell are used, `local (3)`, as can be seen in Figure 6.2. The concept of `bidirectional` traversal is used in the STL for this type of data structure. A binary tree data structure can also be modeled by this type of complex topology.

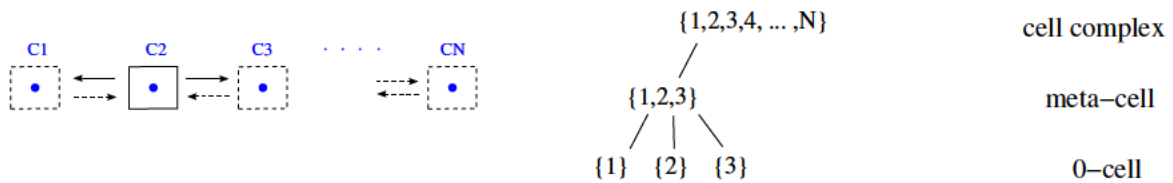


Figure 6.2: Complex topology of a doubly linked list.

Next, the complex topology of a tree structure is depicted in Figure 6.3, whereas the term `local (4)` is used for the characterization of the meta-cell. As can be seen, the singly linked list, the doubly linked list, and the tree contain a similar structure and differ only in the number of elements in their respective meta-cells.

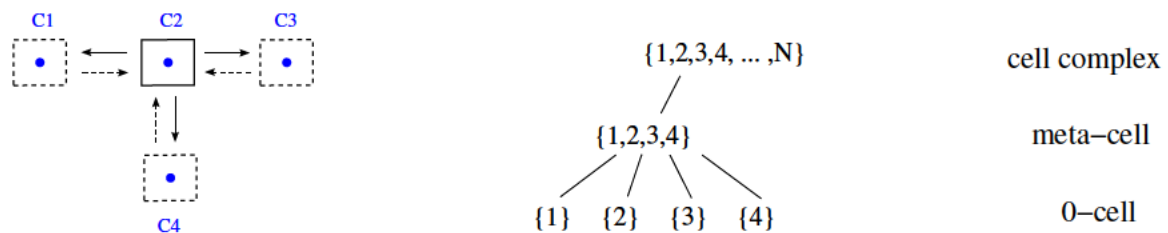


Figure 6.3: Complex topology of a tree.

The last data structure related to commonly used structures within the 0-cell complex is based on the concept of an array. Its complex topology is shown in Figure 6.4. A notable difference with respect to the data structures which have already been reviewed can be observed. The poset is built by only two layers: the 0-cell sets and the complex set. This type of data structure therefore uses a so-called `global` property. All cells are directly accessible from the complex. In terms of the STL, this is called `random access` property.

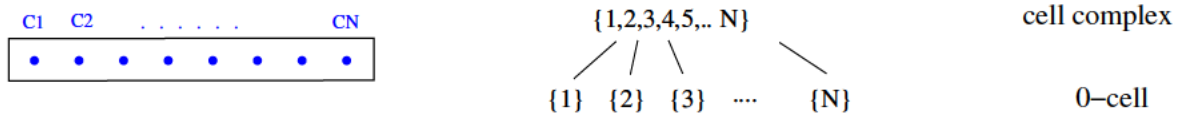


Figure 6.4: Complex topology of an array.

The next example uses a cell dimension of one and therefore a one-dimensional cell complex. Figure 6.5 introduces the complex topology of this type of cell complex, commonly known as graph. Vertices incident to an edge as well as adjacent edges can be traversed.

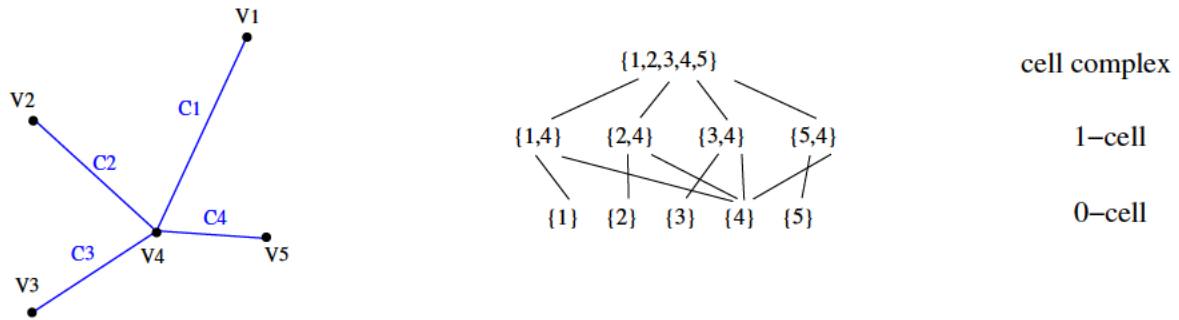


Figure 6.5: Complex topology of a graph.

These two types of complexes, the zero- and one-dimensional complexes, can now be abstracted into higher dimensional complexes with two different types of complex topologies: the local (Figure 6.6) and the global cell complex (Figure 6.7). Based on the cell complex types, the following cell types are available:

- 0-cell: vertex
- 1-cell: edge
- 2-cell: triangles, quadrilaterals
- 3-cell: tetrahedra, cubes
- 4-cell: hyper-tetrahedra, tesseract

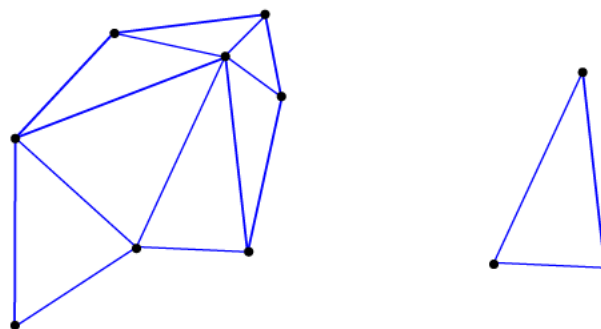


Figure 6.6: Local cell complex (left), normally called mesh, and a 2-cell representation (right). Vertices are marked with filled circles.

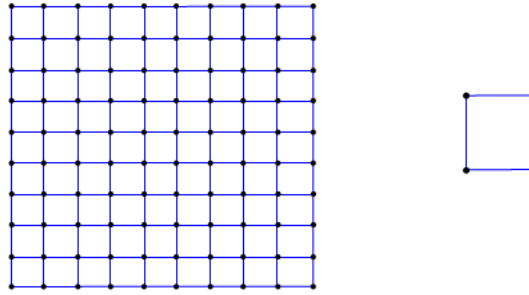


Figure 6.7: Global cell complex (left), normally called grid, and a 2-cell representation (right).

The most important property of a CW-complex can be explained by the use of different n -cells and by consistently attaching sub-dimensional cells to the n -cells. In the following an abbreviation to specify the CW-complex with its dimensionality, e.g., a 1-cell complex describes a one-dimensional CW-complex, is used. So far all mechanisms to handle the underlying topological space of data structures have been introduced. Each subspace, the p -skeleton, can be uniquely characterized by its dimension. All these subspaces are connected appropriately by the concept of a finite CW-cell complex. The first part of the data structure characterization is listed in Table 6.1.

data structure	cell dimension
array	0
singly linked list (SLL) / stream	0
doubly linked list (DLL) / binary tree	0
arbitrary tree	0
graph	1
grid, mesh	2,3,4,..

Table 6.1: Classification scheme based on the dimension of cells.

A common scheme of characterizing data structures by their dimensions is now available. A drawback of this characterization can easily be seen by the 0-cell complexes. The topological space and subspaces only guarantee the correct connection of its subsets, but traversal mechanisms cannot be derived based on these concepts. Therefore, concepts of order theory, as given in Section 2.3.1, are used to formalize the internal structure of cells and the structure of a cell complex. Incidence traversal can then be directly derived from this concept.

The characterization given in Table 6.1 is extended with the cell topology in Table 6.2. The zero-dimensional cell complex types do not differ at all using this classification. But for dimensions greater than one, a distinction of cell topologies becomes apparent.

data structure	cell dimension	cell topology
array	0	simplex
SLL/stream	0	simplex
DLL/binary tree	0	simplex
arbitrary tree	0	simplex
graph	1	simplex
grid	2,3,4..	cuboid
mesh	2,3,4..	simplex

Table 6.2: Classification scheme based on the dimension of cells and the cell topology.

Based on the formal concepts just defined, the classification scheme can be further refined using the number of elements in the meta-cells. An overview is provided in Table 6.3. The complex topology uses the number of elements of the corresponding meta-cells.

data structure	cell dimension	cell topology	complex topology
array/vector	0	simplex	global
SLL/stream	0	simplex	local(2)
DLL/binary tree	0	simplex	local(3)
arbitrary tree	0	simplex	local(4)
graph	1	simplex	local
grid	2,3,4,..	cuboid	global
mesh	2,3,4,..	simplex	local

Table 6.3: Classification scheme based on the dimension of cells, the cell topology, and the complex topology.

This final classification scheme introduces a unique classification scheme based on topological properties only, where well-known and used data structures, e.g., a singly linked list, are also integrated. Thus the internal structure of data and traversal is clearly distinct from all quantity storage mechanisms.

6.2 Boundary Operation

The concept of chains transforms the properties of a cell complex directly into a computationally manageable algebraic structure. The practical concepts in the subsequent sections use these mechanisms to derive different relations between the cells, such as incidence, adjacence, and boundary operations. The essential requirement for a generic data structure implementation of a cell complex is to satisfy the homological concepts of a chain in a computational environment and thereby the concept of the boundary operator thus obtaining, for a complex of dimension n , the sequence $0 \leftarrow C^0 \xleftarrow{\partial_1} C^1 \xleftarrow{\partial_2} C^2 \xleftarrow{\partial_3} C^3 \leftarrow 0$. The complete chain complex of \mathfrak{K} is defined by $\text{im } \partial_{p+1} \subseteq \ker \partial_p$, or less abstractly, $\partial_p \partial_{p+1} = 0$. This is of particular interest because, while $\text{im } \partial_{p+1} \subseteq \ker \partial_p$, in general $\text{im } \partial_{p+1} \neq \ker \partial_p$ and the part of $\ker \partial_p$ not in the inclusion contains useful information [53].

The given boundary operator, introduced in Section 2.4.1, lacks generality, because the cell topology (see Section 2.3.2 for further details) can be arbitrarily complex, e.g., the given boundary homomorphism already has to be extended if a cube cell is used instead of a simplex cell. Therefore, the given poset notation of the cell topology is used to introduce a more general boundary mechanism which can be easily converted into a computationally efficient operation. The boundary operator can then be used to traverse the levels of the poset, e.g., a three-dimensional simplex, illustrated in Figure 6.8.

As can be seen, the boundary operator simply decreases the level of evaluation of the cell topology poset. The boundary operator ∂ transforms p -chains into $p - 1$ -chains, which is compatible with the addition and external multiplication of chains. By the linearity property of the boundary operator, a unique and simple way of consistently deriving the boundary of a chain is given by:

$$\sum_i w_i (\partial \tau_p^i) = \partial \left(\sum_i w_i \tau_p^i \right) \quad (6.1)$$

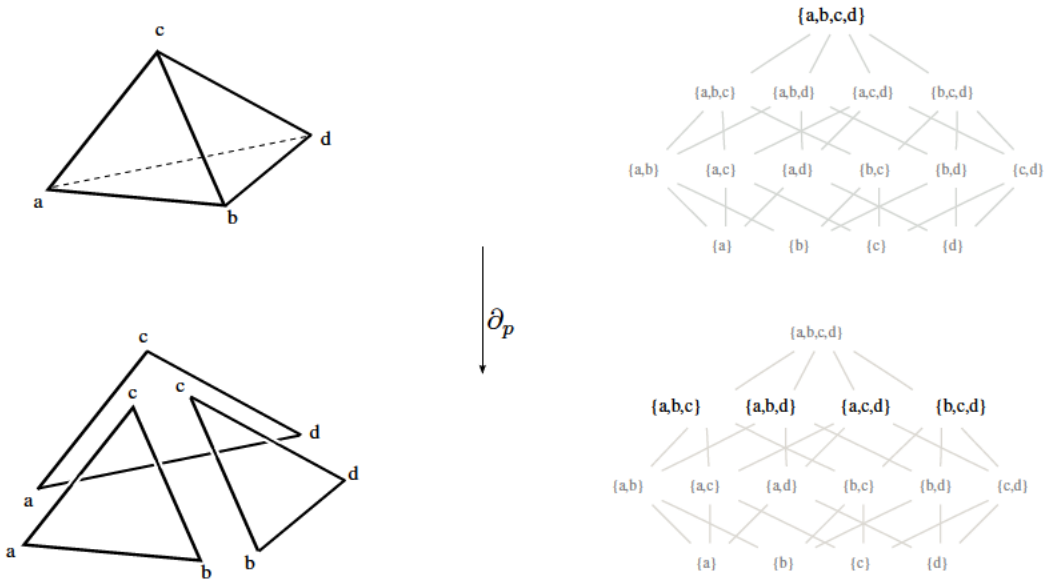


Figure 6.8: Boundary operator applied onto a 3-simplex poset.

6.3 Data Structure Storage Mechanism

As given in Section 2.6, the chain concept can be used in conjunction with the fiber bundle concept, by the so-called intrinsic incidence information of each p -skeleton. In this work, the information regarding the cell topology is stored within the poset, where the complex topology information, for the local case, has to be stored explicitly. To enable an efficient means for this type of storage, an additional concept of a connection matrix [53] is introduced not only to derive neighborhood information of cells, but also to enable different types of traversal mechanisms. To enable vertex-on-cell traversal, only the minimal information has to be stored. To enable a full traversal hierarchy, not only the vertex-on-cell but also the cell-on-vertex information has to be stored. Based on this twofold incidence information, the full hierarchy of traversal is available. It is also of utmost importance to store the local orientation of each cell to avoid additional implementation overhead for the subsequent applications.

The basic property for a consistent data structure is the total ordering of the m_n n -cells of an n -dimensional complex \mathfrak{K} , whereas the vertices of a cell are modeled by a partial ordering, as introduced in Section 2.3.1. Based on the lattice property of a cell complex, which every complex in this work fulfills, each cell can be uniquely identified by its set of vertices, e.g., a p -cell:

$$\tau_p = \{v_{k_0}, \dots, v_{k_p}\} \quad (6.2)$$

For the minimal case, the only information to be stored is the connectivity information of the cell-vertex relation with the corresponding orientation. The following connection matrix with a size of $m_n \times m_0$ is obtained. With additional data storage concepts, this information can be stored as a sparse $m_n \times k$ matrix, with a constant $k = \text{number of vertices of a cell for homogeneous cell complexes}$. The connection matrix, where j stands for the p -cells and i for the 0-cells is then given by:

$$C_j^i = \begin{cases} +1 & \text{if vertex } \tau_0^i \text{ is a consistently oriented part of cell } \tau_n^j \\ -1 & \text{if vertex } \tau_0^i \text{ is not a consistently oriented part of cell } \tau_n^j \\ 0 & \text{otherwise} \end{cases} \quad (6.3)$$

An example for a connection matrix is depicted in Figure 6.9, where the zeros are omitted, thus revealing the sparse structure of the matrix.

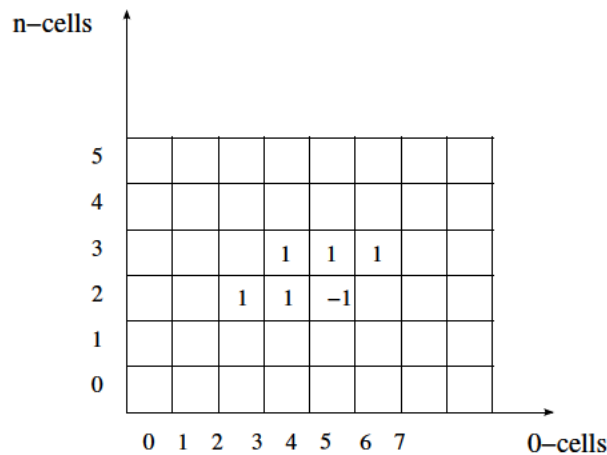


Figure 6.9: Connection matrix C_j^i for a cell complex.

The connection matrix, obtained by a mesh generator, thereby enables the following traversal operations:

- Vertex-on-cell: using a horizontal query
- Cell-on-vertex: using a vertical query

By using the poset mechanism and the stored orientation information to initialize the poset structure accordingly, the following additional traversal mechanism are available:

- Oriented vertex-on-cell
- Oriented edge-on-cell
- Oriented facet-on-cell

Using additional evaluation mechanisms of the connection matrix, adjacency traversal is also possible:

- Vertex-to-vertex
- Edge-to-edge
- Facet-to-facet
- Cell-to-cell

By evaluating the adjacency information, boundary operators can also be derived.

6.4 Separation of Base and Fiber Space

The separation of access to data structures and algorithms by means of iterators has become one of the key elements for the generic programming paradigm. Thereby the implementation complexity of algorithms and data structures is significantly reduced (see Section 5.2.3 for more details). But the traversal mechanisms and data access is combined, making a formal interface between data structures and algorithms difficult.

The concise separation property given by the fiber bundle concept states that the only interface between base space and fiber space is focused on a topological identification mechanism, which finds an actual implementation as a cursor and property map concept [110] in the C++ STL¹. This topological identification mechanism is usually implemented by simple integer numbers.

¹For more information regarding the analysis of this issue see Appendix B.

The following source snippet is an example of this mechanism:

```

val = pmap(*cursor); // accessing a value

pmap(*cursor, val); // storing a value

++cursor; // traversal

```

Cursor and property map concept

Compared to the STL operation realized by iterators:

```

val = *iter; // accessing

++iter; // traversal

```

C++ STL iterator concept

Based on the generic data structure specification and the fiber bundle concepts which have been introduced, the following formal classification can be obtained:

total space	data structure
base space	index space [21], cell complex [1]
fiber space	attached data

The decomposition of the base space is modeled by the concept of a CW-complex, realized by data structures. As an example, Figure 6.10 depicts on the left an array based on the concept of a fiber bundle. Attached to each cell, marked with a dot in the figure, a simple fiber with several sections is shown which conserves the neighborhood of the base space and carries the data of the array. Another example is given in Figure 6.10 on the right where the base space is modeled by a two-dimensional triangular cell complex.

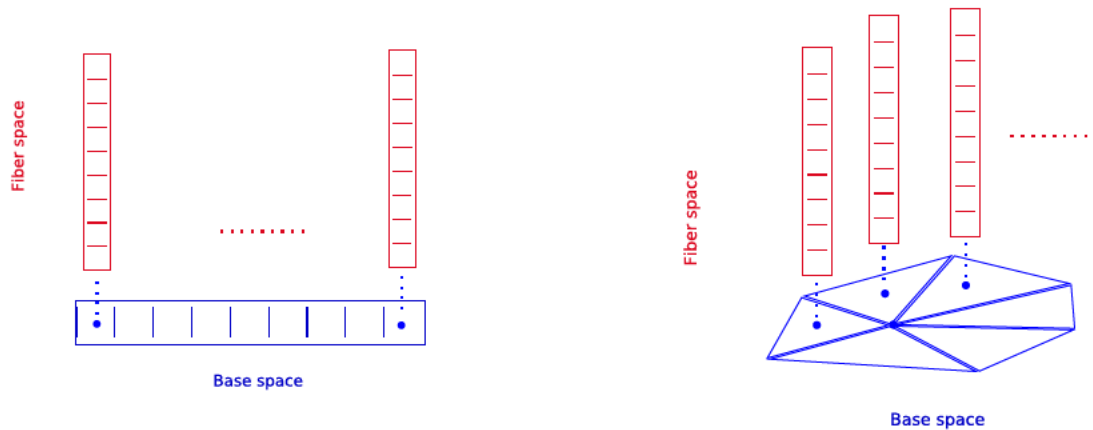


Figure 6.10: Left: a fiber bundle over a 0-cell complex. Right: a fiber bundle over a 2-cell complex.

Organizing data by means of fiber bundles separates properties of the base space (which usually is given by a manifold, but more general topological spaces are possible as well) from the properties of the fiber space. The advantage from the software design view is that both properties can be implemented independently. The same mechanisms can be used to access a vector field given on a tetrahedral mesh, a structured grid, or on a particle set.

For the base space of lower dimensional data structures, such as an array or singly linked list, the only relevant information is the number of elements determined by the so-called index space. Therefore most of the data structures do not separate these two spaces. For backward compatibility with common data

structures the concept of an index space depth is introduced. As can be seen in Figure 6.11, common data structures can be modeled with an index space depth of zero, such as arrays and lists where the content is directly stored at the given location, e.g., an address. An index space depth of zero means that the location of the data structures addressed by an index, e.g. `container[0]` can be used directly as the corresponding value. An index space depth of one or larger means that the index value returned has to be used in another container, e.g., a fiber. In other words, the stored content is only an indirection, e.g., an address of a memory location, which has to be dereferenced to obtain the content. The cursor and property map concept always model an index space depth of one.

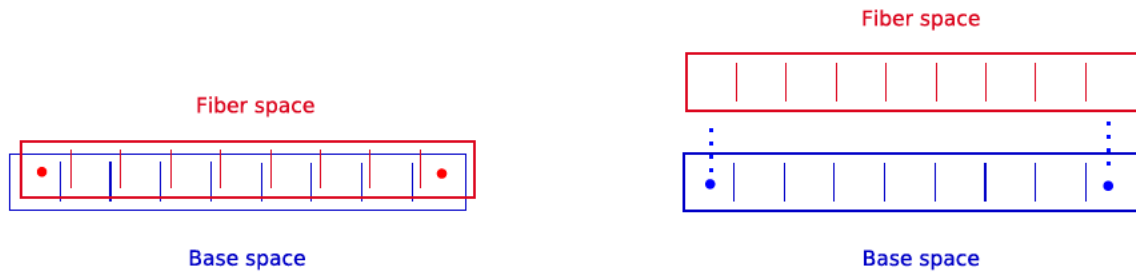


Figure 6.11: Illustration of an index space depth of zero (left) and one (right). The base space is modeled by a 0-cell complex.

The advantages of this approach are similar to those of the cursor and property map but differ in several details. The similarity is that both properties can be implemented independently. But the fiber bundle approach equips the quantity space with more structure, e.g., storing more than one value corresponding to the traversal position as well as preservation of neighborhoods. This feature is especially useful in the area of scientific computing where different data sets have to be managed, e.g., multiple scalar or vector values on vertices, faces, or cells. Based on the structure of the fiber space, vertical and horizontal subsets (sections) can be extracted.

Another important property of the fiber bundle approach is that an equal (isomorphic) base space can be exchanged with another cell complex of the same dimension. The concept of an index space depth ensures backward compatibility of all data structures in the same context as higher dimensional data structures, such as grids and meshes. The following table summarizes the common features and the differences between cursor and property map and fiber bundles:

	cursor and property map	fiber bundles
isomorphic base space	no	yes
traversal possibilities	STL iteration	cell complex
traversal base space	yes	yes
traversal fiber space	no	yes
data access	single data	topological space
fiber space slices	no	yes
backward compatibility	no	yes

6.5 Library-Centric Software Design

Libraries have become a central part of all major programming efforts connected to scientific computing, as introduced in Section 5.4. As a consequence the possible library-centric software design can be regarded as a methodology for designing applications as an assembly of single components with a low degree of coherence and a high degree of orthogonality. However, the following basic principles are essential for writing successful generic components [16]:

- Functions should not depend on the global status but only on the arguments.
- Every function is either general or application-specific.
- Every function that could be made general should be made general.
- The global state should be documented by describing both the semantics of individual variables and the global invariants.

An important step towards library design is the definition of interfaces based only on concept requirements [93, 94] to avoid monolithic application development which always leads to redevelopment of parts or complete applications. Instead already existing concepts and modules which have already proven successful can be used. Essential requirements related to an optimal library development can be summarized as follows:

- A set of libraries has to be complete and therefore must provide a systematic taxonomy [16, 71] to guide the design of an application. Many different types of applications can be written using these libraries and adaptors.
- Libraries should be generic that means that they are usable [17, 130] for a broad range of different applications. Each of the software components is not only written for a very specific purpose, but for a manifold of problems.
- Constraints on performance are required for each of the libraries [93] to obtain an overall high performance application.
- The interoperability of a library is not adversely affected by its completeness [94]. Even if a library is complete by itself, it provides standardized interfaces which guarantee compatibility for data structures which have not been foreseen in its initial design.

In the past the main drawback related to library-centric design was the absence of programming paradigms supporting this type of design. An example can be seen in one of the features most used in programming: loops. The imperative style of using loops offers a great degree of flexibility during the development process, with regard to maintainability and side-effects. Nevertheless, simple loops require local variables to maintain a state, and only the imperative style of quantity access is available. These issues result in codes that lack maintainability, scalability, and lead to unnecessarily error-prone implementation bodies. Another issue which also does not support the concept of component reuse by itself is the object-oriented programming paradigm (Section 5.2), which complicates the interoperability of its software modules. An advantage of imperative and object-oriented programming is that no sophisticated programming techniques have to be taught and learned for one to be able to understand or extend the code. But it also results in an additional drawback, which directly results from the missing expressiveness of the code. Highly optimized code sections can barely be extended by other developers.

Reusability, orthogonality, enhancement capabilities, and performance are all issues which can be eased by using paradigms other than imperative and object-oriented programming. One of the most important issues related to library-centric application design is the shift from these programming paradigms to generic programming and the efficient implementation in programming languages which then offer generality and specialization at the same time. The related work Section 5.4 shows that an important part of generic library design is focused on the selection of the most suitable programming paradigms.

The combination of different programming paradigms fits the scenario of scientific computing exceptionally well. The generic programming paradigm establishes homogeneous interfaces between algorithms and data structures without sub-typing polymorphism. Functional programming eases the specification of equations and offers extendable expressions while retaining the functional dependence of formulae by higher order functions. Also, this type of specification of access, algebraic manipulation, and traversal circumvents the problems of the imperative implementation. The features of meta-programming offer the embedding of domain-specific terms and mechanisms directly into the host language as well as compile-time algorithms to obtain optimal run-time. Developments toward an alternative compilation model and active library design are also an important step [47, 49, 131]. However, reusability of traditional libraries is often extremely limited due to the following issues:

- *Numerical data types.* There are numerous well-known numerical data types which also are often optimized for special applications in order to yield high performance. Only with generic interfaces can these performance-enhancing measures be used in different kinds of applications.
- *Topologies.* Numerical schemes often require different underlying topological data structures. While some applications perform well using structured grids, other applications require unstructured meshes with varying local feature sizes. Although the nature of these topologies is totally different, standardized interfaces for all topological data types have to be provided.
- *Different dimensions.* Special symmetries that are encountered in many problems of scientific computing can be used to reduce the effective dimension of a calculation. Even though all problems can be treated in their full dimension, an enormous gain in performance by using lower dimensional data structures can not be neglected.
- *Equation system assembly.* Most of the solver mechanisms require an initialization of the values of their own interfaces. Therefore, an interface which abstracts these specialties and makes the solvers accessible in a general manner is required. With such an interface the governing equations can be formulated independently of the actual data structures of the solver.
- *Solution of large equation systems.* A lot of problems result in large equation systems which have to be solved. There are solvers available for various special cases, which perform well under certain circumstances but fail to converge sometimes. Therefore, interface design has to guarantee that different solvers can be used.

To circumvent the stated issues, a set of requirements for library-centric application design in the field of scientific computing is given in the following to allow the transformation of the concepts for scientific computing into generically applicable and efficient software components. A transformation into applied concepts has to comply with these concepts.

The fiber bundle concept is used as an organizing tool for a first part of this set of requirements. It separates the application modules into base space and fiber space modules. Base space modules are used for traversal within the cell complex and other cell complex issues, e.g., derivation of cell and complex topology as well as an efficient data structure for storing the connection matrix. Fiber space modules deal with the evaluation of quantities with respect to their differential form or cochain representation with additional semantic information. This means that the appropriate combination and evaluation of cells and their corresponding quantity has to be determined accordingly.

An additional requirement is related to an efficient use of programming paradigms for each of the given tasks. A base layer is identified, which has to implement a formal topological interface for cell and complex properties as well as for traversal, thereby establishing a consistent interface for data structures and quantity storage. The object-oriented and generic programming paradigms are best suited to accomplish this. On top of this base layer, functional expression specification facilities are required, which can be modeled best by the functional programming paradigm. The concept of a DSEL in C++ requires the additional concept of meta-programming resulting in an active library concept to implement and guarantee an overall high performance.

Part III

Applied Concepts

Chapter 7

Overview

During the last decades, several software projects have been developed at our institute and by other research groups (see Section 5.4 for a detailed overview), which have indicated that reuse of single algorithms is difficult. Usually implementations of algorithms make certain assumptions about the way the data they require is represented. But the spirit of generic programming, with its first realization in the STL, has shown that the concept of iterators can enable reusable data structures and algorithms [16, 17, 71] even with an overall high performance comparable to handwritten and hand-optimized implementations.

While the STL data structures are an excellent foundation for common programming tasks, the currently encountered issues from the field of scientific computing call for more practically adapted realizations of concepts. These are presented in this section in order to provide not only orthogonality and expressiveness but also efficiency by the means of a generic scientific simulation environment¹ (GSSE). Usually, orthogonality transcending the initial purpose is complicated by additional complexity of data structures. This results in an increasing number of concepts which have to be met by the data structure, such as the multitude of possible internal representations of graphs in the BGL. This leads to problems whenever algorithms are reused outside the originally intended context, or if the context changes, e.g., a higher dimensional simulation.

The evolution of complexity just described leads to a growing number of software packages for different types of PDE solutions. In addition to this multitude of new packages, they are usually not organized in a way that allows immediate algorithmic reusability. Different projects have contributed components to the field of scientific computing, but up to now, no general set of generic data structures or algorithms suitable for scientific computing in general has been developed. Some works have developed modules, e.g., for generic grid components [18], which was a major contribution to software components for scientific computing with an in-depth analysis of the problem related to algorithms operating on computational grids. It analyzed the relationship between data and algorithms based on topological and combinatorial concepts. A small set of kernel components were developed which greatly ease the specification of algorithms for grid applications. But other issues, e.g., separation guidelines for concepts or functional description of algorithms, were always neglected due to the various diverging requirements in scientific computing. Another important element, as introduced in Section 6.5, is related to the multi-paradigm approach, proposed here that combines object-oriented, generic, and functional programming. To illustrate the chronological evolution of paradigms, the development of applications at our institute and selected libraries are reviewed.

¹The GSSE uses an open source license (Boost [72]) and is available at <http://www.gsse.at>

Name	Year	Paradigm	Further information
MINIMOS	1980	imperative	[132]
S*AP	1989	imperative	[112]
MINIMOS-NT	1996	imperative, OO	[13]
AMIGOS	1998	OO	[118]
WSS	2000	OO	[12]
FEDOS	2004	OO	[69]
GrAL	2000	OO, GP	[18]
CGAL	2001	OO, GP	[120]
BGL	2002	OO, GP	[43]
GSSE	2006	OO, GP, FP	

The current trend, however, is to combine several programming languages, resulting in multi-language applications [133]. Different languages are utilized, each within the field where they perform best. Languages such as Python are used to connect different modules. But problems of interface specification and implementation arise with the combination of several programming languages, further complicating matters. Next, the handling of different languages on different platforms is even more difficult. In the field of scientific computing the performance aspects should be handled orthogonally to the development of applications. Optimizations can thereby be treated separately. With the multi-language approach performance aspects can not be considered orthogonally because of the use of compiled modules which require an interface layer in order to build applications.

Following the guidelines for library-centric application design that have been introduced in Section 5.2 and in Section 6.5, the following sections present advances related to a reusable collection of libraries of the GSSE. The library-centric design is facilitated, as the following criteria are met:

- The environment is complete, so all applications can be written exclusively using its libraries (as well as standard libraries). Indeed, completeness increases usability enormously, because no components have to be added while existing components can be adapted.
- The components of the environment are usable for a broad range of different applications.
- The interoperability of the environment is not affected by its completeness. Even though all the libraries can be used by themselves, they provide standardized interfaces which guarantee compatibility for data structures which have not been foreseen in the initial design.

The following sections deal with these concepts which have to be reflected in a computational approach.

Chapter 8

A Generic Scientific Simulation Environment

The two building blocks which have become apparent are now introduced by two generic libraries accompanied by several additional components, thereby completing the GSSE. Here the most abstract form, reduced to topological concepts, is used to introduce a comprehensive and generic topology library (GTL) based on the object-oriented and generic programming paradigm. It encompasses:

- Properties of a CW-complex translated into mechanisms for cell and complex topology.
- The topological data structure mapped into an efficient implementation of the connection matrix.
- The implementation of the concept of cochains into quantity accessors.
- Implementation of the boundary operator by the means of abstract traversal mechanisms.

In Section 6.1, a common and generic topological classification mechanism was introduced to offer a consistent interface to various types of data structures. All different types of data structures can thereby be interpreted topologically as cell complexes of a certain dimension. The topological data structure requirements demand algebraic properties for collections of cells. The concept of a connection matrix has been introduced to offer computational access to the operations of incidence and adjacency traversal as well as to boundary operations to generate consistent subspaces, the skeletons. The GTL also provides incidence and adjacency traversal operations for various topological elements. The traversal concept allows the formulation of algorithms based on this interface independently of the actual implementation of the topological data structure or the dimension considered. Such a consequent use of the topological interface leads to dimensionally and topologically independent application design.

This approach of a generalized topology make a functional description for algorithms possible which are a natural choice for reuse in the field of scientific computing. The functional specification of algorithms thereby derived is used to develop and implement a minimal base of generic functors which make only minimal assumptions about the structure they operate on, thereby recognizing an advanced equation processing framework and providing high performance as well as high expressiveness by a generic functor library (GFL):

- Functional specification of topological traversal.
- Quantity accessors.
- Functors suitable for discrete mathematics implementing basic arithmetic operations.

The GFL implements the building blocks (functors) for the domain-specific embedded language and is connected to the GTL only by means of topological traversal interfaces and the quantity accessors, thereby providing a clean and concise interface.

Based on these components, the GSSE is specified by a direct implementation of the concepts given. An overview is depicted in Figure 8.1. Meta-programming supports both libraries to enable an overall high performance. The subdivision into a `segment` layer and `domain` layer reflects the modular library design. The `segment` layer combines the topological properties of a cell complex with a simple quantity storage and models a singly connected part of space. Each segment uses a special quantity for coordinate representation, thereby implementing the concept of a chart. No additional concept is used to model geometrical properties. Multi-segment simulation is covered by the fiber bundle concept and consistent base space property of the GSSE. This means that each segment of a domain holds its own quantity storage and is uniquely identifiable. Based on the `segment` concept, the `domain` layer utilizes the concept of an atlas, thereby enabling metrical properties. The atlas is used to represent a consistent view of each segment, based on a globally unique geometrical point list. The domain is also responsible for the evaluation and access of topological elements based on geometrical data from outside. A component for orthogonal range queries is therefore also available, where a data structure is used to arrange the geometrical information of points correspondingly to topological elements.

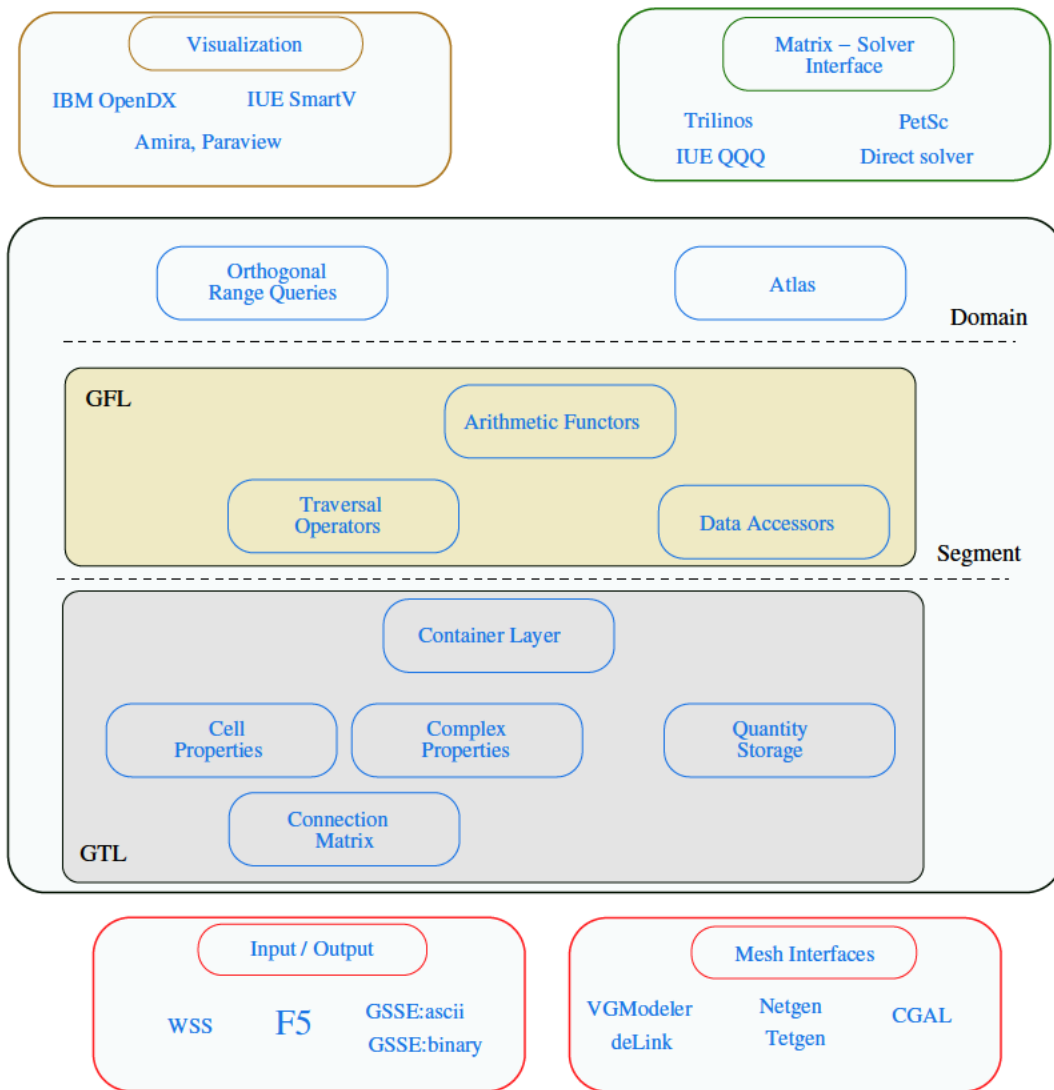


Figure 8.1: Building blocks of the GSSE.

The GSSE is also equipped with a real-time visualization interface, enabling a visual programming concept to ease the actual application development by using abstract quantities to visualize, e.g., boundary areas. The visualization interface can also be used to illustrate matrix properties as well as the evolution of non-linear solution steps. Currently, interfaces to the IBM OpenDX [80], the SmartV developed at our institute, Amira [134], and Paraview [135] are available. The matrix solver interface for various solver packages is implemented by a generic interface, introduced in Section 8.3.1. The input/output module implements the F5 [136] interface and is introduced in Section 8.3.3. Finally, the mesh interface uses various wrapper layers to enable the use of several mesh generators such as VGM [137], deLink [138], Netgen [139], Tetgen [140], CGAL [120], and GrAL [18] to implement the connection matrix. The analysis in Section 6.5 shows that more than 90 percent of the GSSE code consists of general functions.

8.1 The Generic Topology Library

The most basic mechanisms of the GTL are the utilization of cell types and complex types, as explained theoretically in Section 2.3.2. A common topological data structure specification scheme can thereby be derived, as given in Section 6.1.

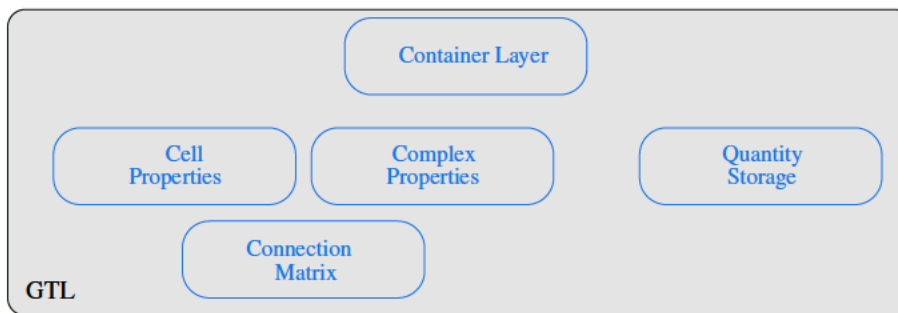


Figure 8.2: The generic topology library of the GSSE.

The GTL transforms each of the given theoretical concepts into a module, where the connection matrix implementation builds the base for the GTL. The poset concept and the corresponding cell topology is implemented by a cell property module, where several posets for different dimensions are stored. Also, several boundary operations for cell traversal are thereby covered. The complex module then implements the local and global cell complex properties, whereby the global cell complex uses procedurally generated incidence and adjacency information and the local cell complex requires the information stored in the connection matrix. The quantity storage module enables an interface to various container structures. Finally, the container layer uses the index space concept and implements an interface to the cell and complex properties. A topological declaration of a simplex and a cuboid object with their respective dimensions is illustrated in the next code snippet¹.

```
cell_type<0, simplex> cell_s;    // vertex
cell_type<3, cuboid> cell_c;    // cuboid
```

Cell type definition

The first specification describes a simple vertex element, a zero-dimensional simplex cell, whereas the second line specifies a cube, a three-dimensional cell. Based on the poset mechanism of the GTL and the concept of a boundary operator (Section 6.2 and Section 6.3), different types of incidence traversal operations are already available. This means that the complete cell topology can be used. However, the cell topology does not contain any cell complex information, such as neighboring cell information.

¹Most of the typedefs are omitted to make the given source snippets as clear and focused as possible.

This information is available by the topology of cell complexes, where only a few complex types are presented in the following code snippet. It should be noted here, however that the complex type is definitely not restricted to these types of cells. Arbitrary types can be used which model the corresponding concepts.

```
complex_type<cell_c, local<3> > complex_2; // doubly linked list / tree
complex_type<cell_t, global> complex_3; // grid
```

Complex type definition

As can be seen, a variety of data structures is thereby uniformly available. Various underlying implementations can be used with a topological formal interface, e.g., the GrAL library for the third line. Basically, the GTL represents a formal topological interface for different types of data structures. Additionally, it provides implementations for the topologies introduced in Section 6.1.

These two parts also integrate a homogeneous access to compile-time and run-time mechanisms. It is important to implement the different evaluation times separately in order to obtain an overall high performance. The following table gives an overview of the different parts of the library with their corresponding evaluation times.

	compile-time	run-time
cell topology, simplex	yes	no
cell topology, cuboid	yes	no
cell topology, polytope	no	yes
complex topology, local	no	yes
complex topology, global	yes	no

An example of the connection matrix, e.g., for a 3-simplex cell complex, is depicted in Figure 8.3: It can

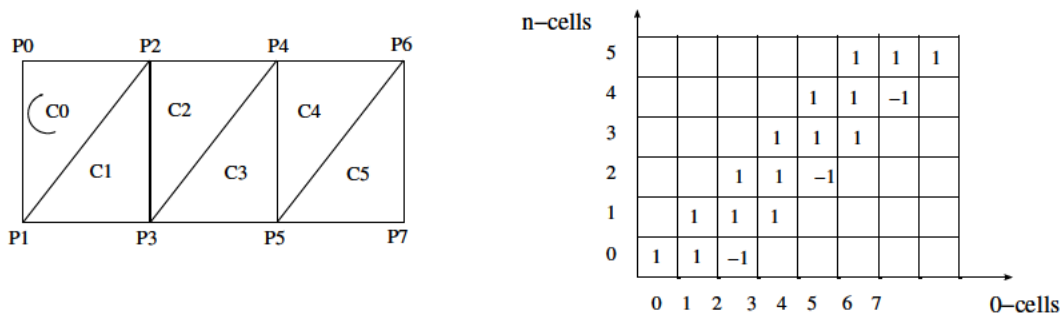


Figure 8.3: Example of the connection matrix C_j^i for a 3-simplex cell complex.

be seen that the connection matrix uses the orientation of the p -cells and is sparse and enables incidence relations (from left to bottom) and adjacency relations (from bottom to left).

The implementation of the container concept is given which combines the complex types (base space properties) with the data types (fiber space properties), where the `indexspace` property is used to introduce the distance of the data from the base space. Finally, an interface to a quantity storage is available, which is connected to the GTL by a unique identifier for each cell.

```
typedef long data_t;
```

```
container_t<complex_1, data_t, indexspace<0> > container;
```

Container type definition

The given example uses an index space depth of zero, which means that the data is directly stored at the cell location.

8.1.1 The 0-Cell Complex

The best known and most widely used data structures in programming can be described by the 0-cell complex types, e.g., the data structures of the C++ STL, depicted in Figure 8.4. The reason for the common topological structure of the C++ STL data structures is the fact that the iterators cannot access any higher dimensional internal structure. On the one hand, this is one of the major advances of the iterator approach of the STL by separating data structure and algorithms. The implementation complexity is thereby significantly reduced. On the other hand, all iterators can only access the zero-dimensional cell of a data structure. Hence it is not possible, e.g., to iterate over the neighbors of a C++ `std::map`.



Figure 8.4: Representation of a 0-cell complex with a topological structure equivalent to a standard container.

The following source snippet introduces the specification of the given topological classification scheme (Section 6.1) with respect to simple well-known data structures, such as arrays or singly linked lists. The tags *global* and *local* in the following code snippet specify the locality of the complex topology.

```

complex_type<cells_t, global > cx; // array
complex_type<cells_t, local<2> > cx; // singly linked list / stream
complex_type<cells_t, local<3> > cx; // doubly linked list / tree
complex_type<cells_t, local<4> > cx; // arbitrary tree

```

Topological data structure definitions

These examples can also be expressed using the STL iterator traits to classify the data structures:

```

complex_type<cells_t, random_access> cx; // simplex, global
complex_type<cells_t, forward> cx; // simplex, local<2>
complex_type<cells_t, bidirectional> cx; // simplex, local<3>

```

Topological data structure definitions, STL iterator traits

The following code snippet demonstrates the equivalence of the STL and the GTL data structures:

```

typedef cell_type<0, simplex> cells_t;
typedef complex_type<cells_t, random_access> complex_t;
typedef long data_t;
container_t<complex_t, data_t, indexspace<0> > container;

```

// is equivalent to the STL container type

```

std::vector<data_t> container;

```

Equivalence of data structures

Here, the separation of the topological structure from the data specification is made explicit, in comparison to the STL, which, unfortunately, combines traversal and data access. Several problems are caused by this combined specification of a data structure². The given index-space concept is used to provide full backward compatibility for the quantity access mechanism, which means that the corresponding traversal mechanism can directly access the obtained iterator.

²Detailed information related to this issue is given in Appendix B

8.1.2 The 1-Cell Complex

Another important representative of a commonly used cell complex is a one-dimensional complex, usually called a graph. Figure 8.5 shows a typical example. A cell of this type of cell complex is called an *edge*. Incidence and adjacency information exists between edges and vertices.

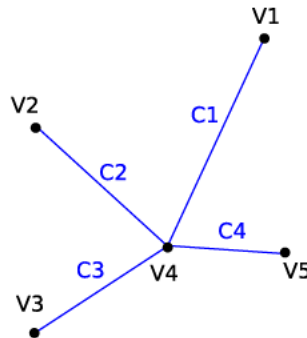


Figure 8.5: Representation of a 1-cell complex with cells (edges, C) and vertices (V).

In the following, the BGL facilities are compared to the traversal mechanisms provided by the GTL. The following example presents the traversal all edges of a 1-cell complex:

```
typedef adjacency_matrix<directedS> Graph;
Graph gr(number_of_vertices);
// edge initialization
edge_iterator eit, eit_end;

for (tie(eit, eit_end) = edges(gr); eit != eit_end; ++eit)
{
    test_source1 += source(*eit, gr);
    test_source2 += target(*eit, gr);
}
```

BGL iteration

The GTL offers the same functionality as is demonstrated in the following code snippet. The `global` tag is used to demonstrate the global structure of the graph, which means that the internal data layout is prepared for dense graph storage. In terms of the BGL, an `adjacency_graph` data structure is used.

```
typedef cell_type<1, simplex>          cells_t;
typedef complex_type<cells_t, global>  complex_t;
typedef long                           data_t;
container_t<complex_t, data_t, indexspace<1> > container(number_of_vertices);
// cell initialization
container_t::cell_on_vertex_it covit, covit_end;

for (tie(covit, covit_end) = cells(container); covit != covit_end; ++covit)
{
    test_source1 += source(*covit, container);
    test_source2 += target(*covit, container);
}
```

GTL's traversal

Related to interoperability, all algorithms developed for the BGL can be used on this structure as well, including the property map concept. The property map can be seen as a specialization of the fiber bundle approach.

8.1.3 The 2-Cell Complex

Commonly used discretization schemes use as base requirement a two-dimensional cell complex, usually called a mesh or grid. Although this cell complex can be projected onto a graph, necessary information is not available for, e.g., a finite volume or finite element method. These schemes require at least cell information, which means that a two-dimensional cell is required. A graph does not contain any cell information except the edges. In the following, the topological specification mechanism is given with respect to two important types of 2-cell complexes, a structured grid and an unstructured mesh. The storage and traversal operators of structured grids benefit greatly from the global complex topology, because all traversal operations can be rendered at compile-time. No additional cell adjacency information has to be stored compared to the unstructured mesh.

cell_type	<2, simplex>	cells_t;
cell_type	<2, cuboid>	cellc_t;
complex_type	<cells_t, global>	complex_structured; // grid
complex_type	<cellc_t, local>	complex_unstructured; // mesh

Grid and mesh definitions

To illustrate a traversal operation offered by the GTL, the following code snippet presents a vertex-on-cell traversal with an additional incident edge traversal. A graphical representation of this traversal is given in Figure 8.6, where an arbitrary cell of the complex is used. Then a vertex-on-cell traversal is initialized, where the marked objects depict the currently evaluated objects. In the first stage of the iteration the vertex v_1 is used, and the iteration is performed over the incident edge E_1 , E_2 . The iteration then continues with the remaining vertices.

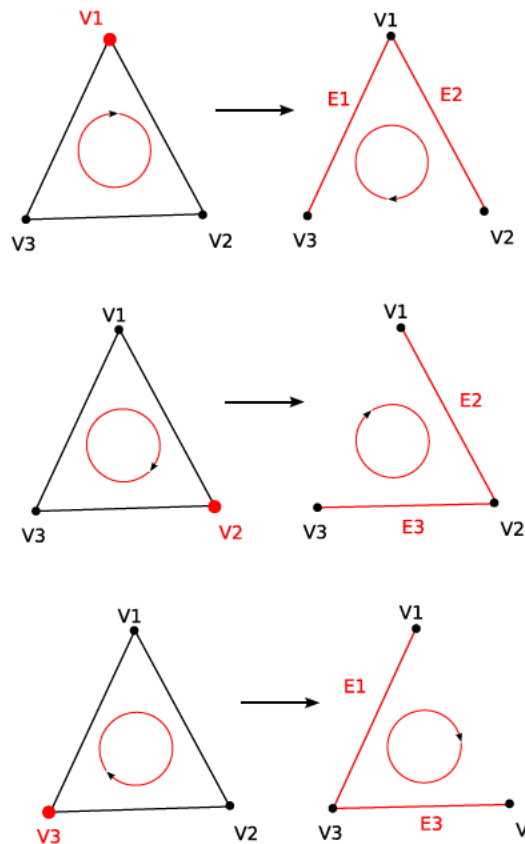


Figure 8.6: Incidence relation and traversal operation.

The given code can be used for both of the 2-cell complex types, the `complex_structured` and `complex_unstructured`. During the loop, an edge-on-vertex traversal is created and initialized with the evaluated vertex.

```

cell_iterator c_it = container.cell_begin();
vertex_on_cell_iterator voc_it = container.voc_begin(*c_it);
vertex_on_cell_iterator voce_it = container.voc_end(*c_it);

for (; voc_it != voce_it; ++voc_it)
{
    edge_on_vertex_iterator eov_it = container.eov_begin(*voc_it);
    edge_on_vertex_iterator eove_it = container.eov_end(*voc_it);

    for (; eov_it != eove_it; ++eov_it)
    {
        //operations on edges
    }
}

```

GTL's traversal mechanisms

Traversal can be used independently of the dimension or type of cell complex. Only three topological properties have to be met by the cell complex: vertices, edges, and cells. All cell complex types which support these three objects can be used for this traversal.

To highlight the interoperability of the GTL, two examples are shown using an implementation of related work of the CGAL and GrAL. The first example presents the traversal mechanisms of the CGAL compared to the GTL. A traversal of all facets related to a cell is shown.

```

// ..... Polyhedron P definition

Facet_iterator fit = P.facets_begin();
for (; fit != P.facets_end(); ++fit)
{
    //operations on a facet
}

```

CGAL traversal for the cell topology

The special geometric object `Polyhedron` in terms of the GTL is implemented by a common container type.

```

// container definition
facet_on_cell_iterator foc_it = container.foc_begin(cell);
facet_on_cell_iterator foce_it = container.foc_end(cell);

for (; foc_it != foce_it; ++foc_it)
{
    //operations on a facet
}

```

GTL's traversal for the cell topology

The CGAL can be used as a complete implementation of the concepts given by the GTL. An interface layer is available to use all CGAL data types within the GTL.

The second example is related to the GrAL. The close relationship between the GrAL and the GTL makes a complete interoperability between these libraries possible. Several different `grid_types` can be used in GrAL, similarly to the GTL container type. The main difference is the orthogonality of the cell types and the complex types in the GTL.

```

typedef grid_types<Triang2D> gt;
Triang2D CC;

for ( gt::CellIterator c(CC); !c.IsDone(); ++c)
{
    // loop over all vertices vc of cell c
    gt::VertexOnCellIterator vc;
    for (vc = (*c).FirstVertex(); ! vc.IsDone(); ++vc)
    {
        // use *vc
    }

    // loop over all neighbors
    for (gt::CellOnCellIterator cc(c); ! cc.IsDone(); ++cc)
    {
        // use cc
    }
}

```

GrAL traversal

As can be seen, the traversal mechanisms are implemented very closely to the `grid_types`. The `random_access` property cannot be used by all traversal operations. This complicates the optimization and performance steps related to application design. The next code snippet presents the GTL implementation. It should be noted that the GTL interface can easily be implemented using the GrAL.

```

// container definition
cell_iterator c_it = container.cell_begin();
cell_iterator ce_it = container.cell_end();

for (; c_it != ce_it; ++c_it)
{
    vertex_on_cell_iterator voc_it = container.voc_begin(*c_it);
    vertex_on_cell_iterator voce_it = container.voc_end(*c_it);
    for (; voc_it != voce_it; voc_it.increment())
    {
        // use *voc_it
    }

    cell_on_cell_iterator coc_it = container.coc_begin(*c_it);
    cell_on_cell_iterator coce_it = container.coc_end(*c_it);
    for (; coc_it != coce_it; ++coc_it)
    {
        // use *coc_it
    }
}

```

GTL's traversal

The interfaces can be exchanged easily and each of these libraries can be used with the GTL interface. Therefore the reuse of source code due to the GTL approach is a major advantage.

8.1.4 Higher-Dimensional Cell Complexes

The 3-cell complex is similar to the 2-cell complex, if simplex or cuboid cell types are used. Therefore the corresponding figures and code snippets are omitted. Based on the given formal definition of finite cell complexes, higher dimensions can be used as well. The theoretical mechanisms introduced in Section 2.3.2 and the implementation of the concepts given in Section 6.1 enable a clean interface for extensions, e.g., a 4-simplex, a pentachoron, depicted in Figure 8.7.

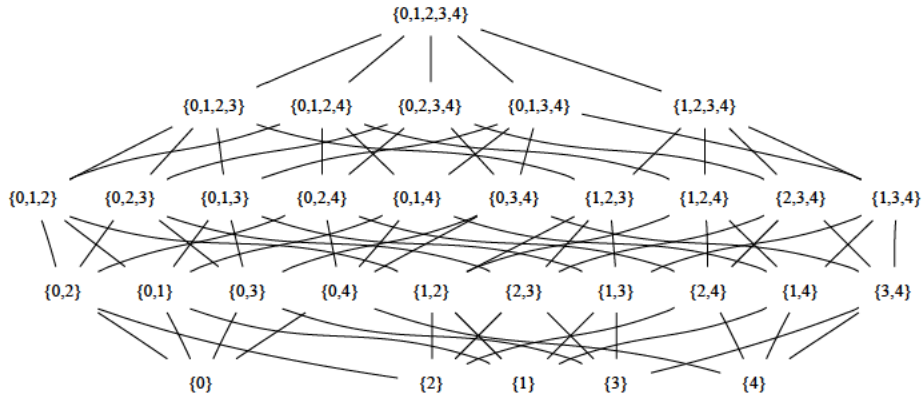


Figure 8.7: The cell poset of a 4-simplex as implemented by the GTL.

The poset of the 4-cube is omitted due to the large number of facets. Renderings of a 4-simplex and a 4-cube are presented in Figure 8.8.



Figure 8.8: Renderings of a 4-simplex (left) and a 4-cube (right).

The following code snippet shows the implementation of an arbitrary topology with the structure of the 4-cell simplex complex. All previously presented algorithms can be used, due to the common traversal mechanisms.

```

typedef cell_type<4, simplex>          cells_t;
typedef complex_type<cells_t, local>    complex_t;
typedef long                            data_t;
container_t<complex_t, data_t, indexspace<1> > container;

```

Four-dimensional simplex complex

The GTL implements all the given data structures from Section 6.1 as well as generic algorithms for simplex and cuboid cell topologies of arbitrary dimensions.

8.1.5 Data Access

Separated by the fiber bundle approach, the stored data corresponding to the topological elements, such as vertices or edges, is available orthogonally by an element identifier, e.g., element number. During initialization, the quantity accessor `q` is bound to a specific domain with its quantity key. To access the underlying data, the `operator()` is evaluated with a vertex of the cell complex as argument and returns a reference to the stored value.

```
string key_quan = "user_quantity";
quan_t q = scalar_quan(domain, key_quan);

quan(vertex) = 1.0;
```

Quantity assignment

The implementation of the fiber bundle approach in the GTL follows:

```
value = container.get_value(base_index, quan_key);
value = container.get_value(*vertex_on_cell, quan_key);
```

Data accessors for the container

As can be seen, the `base_index` selects the respective fiber, whereas the `quan_key` selects the corresponding fiber element. In contrast to the cursor and property map concept, the fiber bundle approach allows traversal operations within fiber space as well:

```
fiber = container.get_fiber(base_index);
for (it = fiber.begin(); it != fiber.end(); ++it)
// ...
```

Fiber access

It is also possible to extract a section, which means that a whole data set attached to a base space can be returned:

```
f_section = container.get_section(quan_key);
for (it = f_section.begin(); it != f_section.end(); ++it)
// ..
```

Section access

8.2 The Generic Functor Library

The GFL is the second important contribution of this work utilizing the concept of a DSEL (Section 5.3) to support functional programming in C++. The GFL is connected to the GTL only by means of topological traversal interfaces and quantity accessors, thereby providing mechanisms to separate discrete mathematics into topological and numerical operations by arithmetic functors. Figure 8.9 presents these building blocks of the GFL.

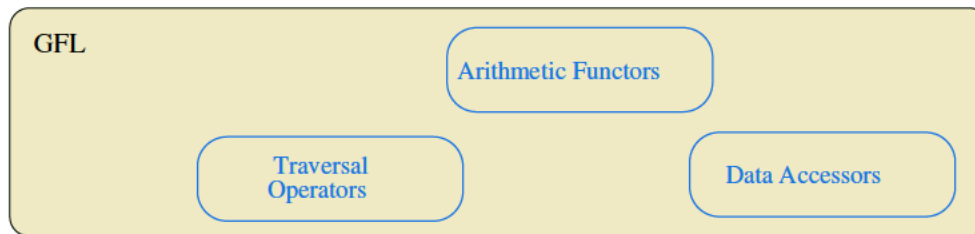


Figure 8.9: The generic functor library of the GSSE.

Based on this collection of functors and using the compile-time functional library Phoenix 2 [44], a highly efficient, maintainable, and scalable DSEL has been created. It is important to note that all functors can be used completely orthogonally with all other functors.

8.2.1 Traversal Operators

As can be seen in Section 8.1, the traversal operations require some additional code to create and initialize the iterators. To ease the application of the GTL, a traversal functor is realized, as given in the next code snippet

```
gsse::traverse<cell_type_1, cell_type_2>
```

where `cell_type_1` represents the base cell under consideration, whereas `cell_type_2` specifies an incident sub-cell, e.g., a vertex or an adjacent cell. This mechanism can use virtually any derived internal structure of a cell or of the whole cell complex to build the requested traversal and avoids the construction and initialization specification. The following source snippet uses a given cell type and derives the boundary of this cell for traversal:

```
gsse::traverse<cell_t, -1>
```

An example of this cell type is a 3-cuboid cell and generates the 3 – 1-cell traversal operation. In other words, this traversal operation generates a facet-on-cell iterator. Simple cell traversal can be specified by the following example which also demonstrates an even shorter specification for this special case:

```
gsse::traverse<cell_t, 0>
```

```
gsse::traverse<cell_t>
```

Figure 8.10 provides examples of a simplex cell and complex type and reviews the possible relationships between topological elements of different dimensions. The first row shows all edges, faces, and cells which are incident with the same base vertex (a)-(c), while the first column shows vertices which are incident with one base edge, face, or cell (d, g, j).

The connection matrix is the underlying data structure for these types of operations.

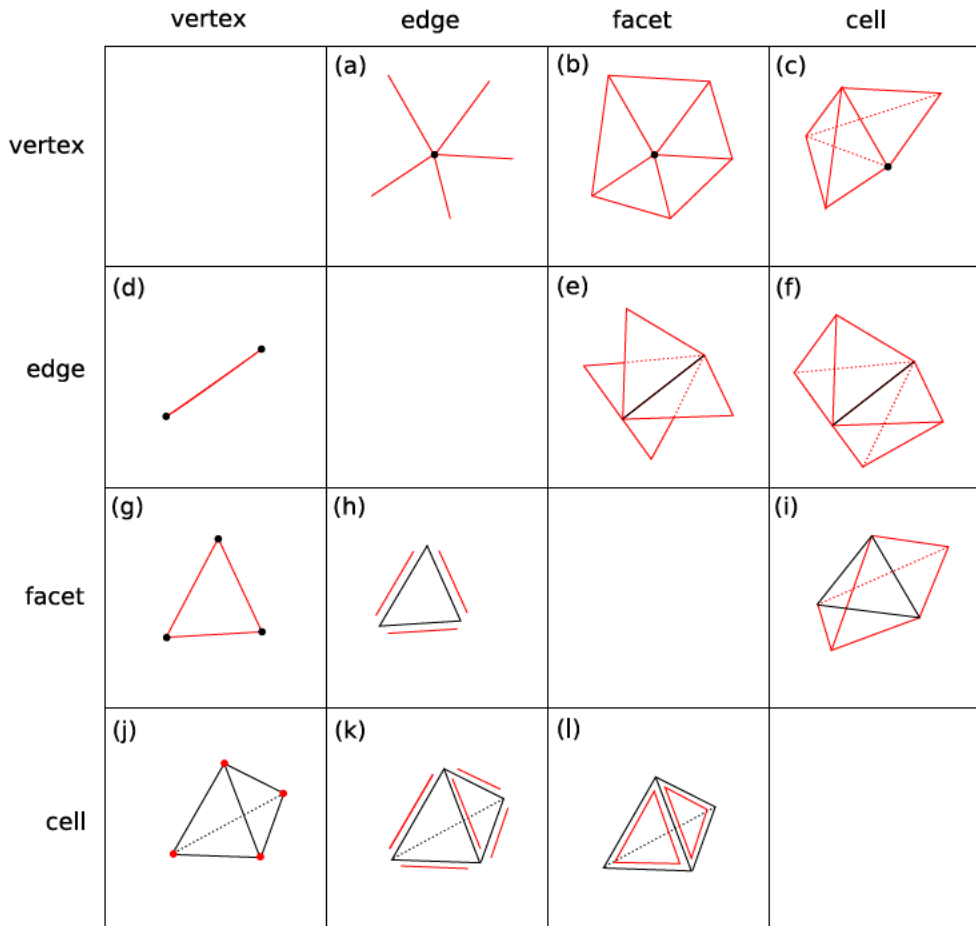


Figure 8.10: Traversal methods induced by the incidence relation. The rows illustrate traversal schemes of the same base element, whereas columns depict traversal schemes of the same traversal element.

8.2.2 Data Accessors

An important concept is the evaluation of quantities related to given topological elements. For example, the quantity storage of discretization schemes requires the evaluation and access of several different dimensional quantity types, e.g., the finite volume method uses edge distances and areas as well as vertex quantities. To create an automatic evaluation of these different types of quantity access, the GFL implements generic data accessors which represent a C++ function object. This means that this object implements, on the one hand, a mechanism for automatic type deduction, and, on the other hand, the operator `()`. In the following code snippet a simple example of the generic use of this accessor is given, where a scalar value is assigned to each vertex in a domain. The quantity accessor creates an assignment which is passed to the `std::for_each` algorithm.

```
string key_quan = "user_quantity";
quan_t quan = scalar_quan(domain, key_quan);

for_each(container.vertex_begin(), container.vertex_end(), quan = 1.0);
```

Simple quantity assignment

8.2.3 Arithmetic Functors

The discretization schemes, as well as several other methods, require standard arithmetic operations. Therefore generic calculations are necessary which can be specified with the corresponding arithmetic operation. The GFL implements comprehensive functor mechanisms. Using these mechanisms as a foundation, the following two algebraic generic components were developed, which are sufficient for all discrete representations of the operations of integration and differentiation. Additionally it is necessary that this generic calculation automatically deduces the neutral element of the given operator for initialization.

```
gsse::calculate<std::plus, traversal_operation >() [ ]  
gsse::calculate<std::minus, traversal_operation >() [ ]
```

Shortcuts are also available for these common operations:

```
gsse::sum < traversal >() [ ]  
gsse::diff< traversal >() [ ]
```

8.2.4 Domain-Specific Embedded Language

A domain-specific embedded language based on the given building blocks of generic traversal, data accessors, and arithmetic functors, as introduced in Section 5.3, is available.

To illustrate the application of the GFL in detail, a simple equation $\sum_{v \rightarrow e} (\Delta_{e \rightarrow v} u) = 0$ is used where u denotes the quantity located on a vertex and Δ denotes the difference, whereas $v \rightarrow e$ states the traversal operation. To express the difference between imperative and functional programming, the example is presented first without the GFL. For both programming paradigms it can be seen that this implementation does not depend on the dimension or type of the cell complex.

```
for (vit = container.vertex_begin(); vit != container.vertex_end(); ++vit)  
{  
    vertex_edge eovit(*vit)  
    for(; eovit.valid(); ++eovit)  
    {  
        edge_vertex voe_it(*eovit);  
        for(; voe_it.valid(); ++voe_it)  
        {  
            equation += lequ(u(*voe_it));  
        }  
        eq += equation;  
    }  
}
```

The same functionality is obtained by using the functional programming paradigm. The main difference is given by the absence of temporary variables within the functional style.

```
for (vit = container.vertex_begin(); vit != container.vertex_end(); ++vit)  
{  
    eq = sum<vertex, edge>  
    [  
        sum<edge, vertex>() [ u ]  
    ](*vit);  
}
```

The complex resulting from this mapping is completed by specifying the current vertex object `*vit` at run-time, which clearly demonstrates the compile-time and run-time border. For a comprehensive explanation the specific parts are separated. The first part specifies a traversal of all vertices in an imperative way.

```
for (vit = (*segit).vertex_begin(); vit != (*segit).vertex_end(); ++vit)
{
  //...
}
```

The following bracket encompass the actual functional expression. The `*vit` represents the currently evaluated terminal object, in this case a dereferenced vertex.

```
eq = sum<vertex, edge>
[
  // ....
](*vit)
```

The heart of the equation is shown in the next code snippet. Here a `sum` is used as the discrete representation of the arithmetic operator. The topological traversal is given by the vertices incident to the edge. This inner part receives the edge information from the `sum<vertex, edge>` and builds a traversal on the vertices on this edge.

```
// ..
sum<edge, vertex>() [ u ]
// ..
```

The quantity accessor `u` supplies the `sum` algorithm with values stored on the vertices. Finally, the summed value obtained from all operations within `sum` is stored in the object `eq`. To show maintainability and scalability, an example of a discretized form of the Laplace equation is given:

$$\mathcal{L}_{\text{ell.fv}}(u) \equiv \sum_{v \rightarrow e} (\Delta_{e \rightarrow v} u) \frac{A}{d} \varepsilon = 0 ,$$

where `u` denotes the solution quantity and Δ again denotes the difference. The geometrical factors A and d , necessary for the finite volume discretization approach, denote the cross section of the flux and the distance between the two edge points, respectively. The final source code reads:

```
eq = sum<vertex, edge>
[
  sum<edge, vertex>() [ u ] * A / d * eps
]
```

With small changes the Laplace equation can be extended to a Poisson equation:

$$\mathcal{L}_{\text{ell.fv}}(u) \equiv \sum_{v \rightarrow e} (\Delta_{e \rightarrow v} u) \frac{A}{d} \varepsilon = V \varrho ,$$

In the next code snippet, the functional character of the implementation approach and the resulting extensibility, is given. Most of the source code remains unchanged; only minor parts have to be added.

```
eq = sum<vertex, edge>
[
  sum<edge, vertex>() [ u ] * A / d * eps
] - V * rho
```

The non-zero right-hand side of the Poisson equation leads to a multiplication with the volume V when the integration is performed using finite volumes.

8.3 Additional Generic Components

As stated, additional generic concepts are available within the GSSE to further ease application design. They all deal with interfaces to various areas of scientific computing.

8.3.1 Generic Solver Interface, GSI

Access to algebraic solver systems requires different mechanisms, if used for a wide variety of discretization schemes. Therefore the GSSE provides a generic solver interface which is based on the introduced traversal and quantity accessors and linearization mechanisms.

The most important parts of the GSI are given in the following short code snippets. First, the compile or run-time specification is illustrated:

```
#include "gsse/matrix_interface/trilinos_solver_interface.hpp"
#include "gsse/matrix_interface/petsc_solver_interface.hpp"
#include "gsse/matrix_interface/qqq_solver_interface.hpp"
```

```
typedef trilinos_solver_interface solver_t;
```

The generic matrix wrapper uses a bijective connection of the assembled quantities and the corresponding domain to provide easy access, where `msi` represents the matrix-solver interface:

```
gsse::gsi::solver_traits<solver_t>::matrix_interface msi();
gsse::gsi::solver_traits<solver_t>::matrix_insert m_ins(msi);
gsse::gsi::solver_traits<solver_t>::matrix_rhs m_rhs(msi);
```

The given quantity accessors are then connected to the solver interface, illustrated in the following:

```
typedef gsse::gsi::solver_traits<solver_t>::type quan_entry_t;
string key_quan = "user_quantity";

quan_t u = scalar_quan(segment, key_quan);

quan_entry_t quan_user (msi, u);
```

thereby creating a quantity `u` which is directly connected to the matrix-solver interface `msi` by the object `quan_user`. The matrix-solver interface also handles the write-back of the solved quantity value. Each quantity is attributed to a segment within a domain.

The use of the final solution procedure with different options is demonstrated in the next code snippet.

```
msi.prepare_solver(); // final instructions for solvers
msi.switch_to_full_output_mode(); // print the solver behavior
msi.set_options_pack4(); // use different types of options

bool solved = msi.solve(); // the solution process
msi.write_back(); // quantity write_back
```

Several algebraic solver software packages are accessible, such as the self-developed direct and iterative solvers [14] and solver packages like Trilinos [15] and PetSc [141], thereby providing a multitude of different solving procedures.

8.3.2 Generic Orthogonal Range Queries

Point location is a necessary requirement for efficient algorithms, such as interpolation, intersection tests, or boundary operators, e.g., for TCAD [119]. Different types of point location mechanisms can be summarized by the concept of orthogonal range queries. The interface of generic orthogonal range queries presented here can be used completely orthogonally, which means that different range query algorithms and libraries can be used. The example is based on the range query concepts and libraries of Mauch [142],

- **Kd-trees, quadtrees and octrees:** The kd-tree outperforms the octree methods in most cases. In addition to the higher execution times, octree methods also have several times the memory footprint of a kd-tree. This fact is due to the differences in how these methods organize their records [142]. Finally, kd-trees can be used as a generic range query method in a wide range of applications.
- **Cell and sparse cell methods:** Sparse cell arrays have execution times that are almost as low as dense cell arrays. Therefore, if the dense cell array has many empty cells, the sparse cell method reduces the amount of required memory storage without affecting the execution time.

Due to the performance dependencies of orthogonal range queries, based on many parameters, such as the dimension of the stored record, the number of records, their distribution, and the query range, it is of utmost importance to have a wide variety of range query modules available. Finally, there is no best method [142] for different application scenarios.

The application by the GSSE for all different types of range query methods is always given by:

```

domain_traits<domain_t>::point_container_t  point_container;

point_container =
  domain.query_pointcontainer( point_t(0,0,0), point_t(0.2, 0.5, 0.5) );

```

8.3.3 Generic File Interface, GFI

The file interface is implemented by means of the fiber bundle data model [1, 136]. By the implementation of fiber bundle within GSSE, the data structures map directly to the hierarchical file format. An example is given by a file resulting from a numerical simulation which has to encompass the time-dependent output of Maxwell's equations, such as **E**, **H**. These fields live on different skeletons of the grid:

- Primary mesh edge fields: **E**
- Primary mesh face fields: **H**

Using the F5 layer the following output information is obtained:

```

/T=1.0/GSSE/Points Group
/T=1.0/GSSE/Points/Cartesian Group
/T=1.0/GSSE/Points/Cartesian/Positions Dataset {40457}
/T=1.0/GSSE/Edges Group
/T=1.0/GSSE/Edges/Points Group
/T=1.0/GSSE/Edges/Points/Positions Dataset {81317}
/T=1.0/GSSE/Edges/Cartesian/E Dataset {81317}
/T=1.0/GSSE/Faces Group
/T=1.0/GSSE/Faces/Points Group
/T=1.0/GSSE/Faces/Points/Positions Dataset {53965}
/T=1.0/GSSE/Faces/Cartesian/H Dataset {53965}

```

The file structure provides a grouping of the different simulation fields according to the topological relations of their skeletons, given here by the points, edges, and faces for one time step. A Cartesian coordinate representation is used.

8.3.4 Finite Element Components

The assembly procedure of finite elements requires additional steps for efficient implementation, as given in Section 3.3. Element-wise assembly is typically used in the finite element method, where all cells are traversed and for each of the cells a local matrix is calculated. This matrix introduces coupling factors between the basis functions. As each basis function is mapped to an element of the underlying cell complex, couplings between functions can be seen as couplings between values on elements. The local matrix entries are written into the global matrix according to a global vertex/cell-numbering scheme. The following source snippet presents the important operations for finite element applications:

```
matrix_type local_m = coord_transformation(coordinates);

gsse::stencil      ( local_m );
assemble_stencil  ( stencil );
```

As stated in Section 3.3, the key for an efficient realization of the finite element method is the transformation of a reference element in a normalized coordinate system (Equation 3.38 - Equation 3.40). To support this operation, GSSE contains a coordination transformation algorithm. C++'s partial specialization feature is used, as given in the following source snippet for a three-dimensional simplex cell.

```
template <typename MatrixType>
class coord_transformation<MatrixType, simplex, 3>
{
    matrix_type operator() (matrix_type coordinates)
    {
        double J11 = coordinates(1, 0) - coordinates(0, 0);
        double J12 = coordinates(1, 1) - coordinates(0, 1);
        double J13 = coordinates(1, 2) - coordinates(0, 2);
        double J21 = coordinates(2, 0) - coordinates(0, 0);
        double J22 = coordinates(2, 1) - coordinates(0, 1);
        double J23 = coordinates(2, 2) - coordinates(0, 2);
        double J31 = coordinates(3, 0) - coordinates(0, 0);
        double J32 = coordinates(3, 1) - coordinates(0, 1);
        double J33 = coordinates(3, 2) - coordinates(0, 2);
        // ....

        double K11 = J22 * J33 - J23 * J32;
        double K21 = J23 * J31 - J21 * J33;
        double K31 = J21 * J32 - J22 * J31;
        double K12 = J13 * J32 - J12 * J33;
        double K22 = J11 * J33 - J13 * J31;
        double K32 = J12 * J31 - J11 * J32;
        double K13 = J12 * J23 - J13 * J22;
        double K23 = J13 * J21 - J11 * J23;
        double K33 = J11 * J22 - J12 * J21;

        double detJ = J11 * J22 * J33 + J21 * J32 * J13 + J12 * J23 * J31
                    - J11 * J23 * J32 - J13 * J22 * J31 - J12 * J21 * J33;

        // ...
    }
};
```

The following pre-calculated element matrices for the Laplace operator are used [113], which are implemented for two- and three-dimensional simplex and cuboid cells and partially given in the following source snippet for a three-dimensional simplex example:

```

template <typename MatrixType>
class stencil<MatrixType, simplex, 3>
{
    stencil()
    {
        Srl1(0, 0) = 1.0/6.0;
        Srl1(1, 0) = -1.0/6.0;
        Srl1(0, 1) = -1.0/6.0;
        Srl1(1, 1) = 1.0/6.0;
        // ...
    }

    MatrixType operator() (const MatrixType& local_m)
    {
        double ga = local_m(0, 0);
        double gb = local_m(1, 1);
        double gc = local_m(2, 2);
        double gd = local_m(1, 0);
        double ge = local_m(2, 1);
        double gf = local_m(2, 0);

        MatrixType result(4, 5);
        result = Srl1 * ga + Srl2 * gb + Srl3 * gc +
                Srl4 * gd + Srl5 * ge + Srl6 * gf;
        return result;
    }
};

```

By utilizing the GSI, the assembly of the local stencil matrix is then given by the following algorithm, where the `index_list` stores the transformation of local node indices to global ones:

```

void assemble_stencil( .. )
{
    for(int i=0; i<index_list.size(); ++i)
    {
        long line = index_list(i);

        for(int j=0; j<index_list.size(); ++j)
        {
            long column = index_list(0, j);
            mat_add(line, column, stencil(i, j)); // GSI interface
        }
        int rhs_pos = index_list.size();
        rhs_add(line, stencil(i, rhs_pos)); // GSI interface
    }
}

```

8.4 Performance Analysis

The trend of increasing clock speeds has led to the problem that CPUs require data at a faster rate than memories are able to supply it.

From a software point of view, the currently available calculation resources of computer systems require the utilization of various paradigms such as object-orientation programming, generic programming, functional programming, and meta-programming, as introduced in Section 5.2, to obtain an overall high performance. The most important parts of how to achieve high performance in C++ can be summarized on the one hand by static parametric polymorphism [94, 143, 144] for a global optimization with inlined function blocks, and on the other hand by lightweight object optimization [41] which enables allocation of simple objects to registers. This has already been demonstrated in the field of numerical analysis, yielding figures comparable to Fortran [47, 100, 145], the previously undisputed candidate for this kind of calculation. The C and Fortran languages do not offer techniques for a variable degree of optimization, such as controlled loop unrolling [143, 146]. Such tasks are left to the compiler. Therefore, libraries have to use special techniques as employed by ATLAS [147] or have to rely on manually tuned code elements which have been assembled by domain experts or the vendors of the microprocessor architecture used. Thereby, a strong dependence on the vendor of the microprocessor is incurred. In short, these methods greatly complicate the development process of orthogonal high performance libraries.

To analyze the performance of the implemented GSSE approach on various computer systems, the maximum achievable performance is summarized in the following table, which shows the CPU types, the amount of RAM, and the compilers used. A balance factor (BF) is evaluated by dividing the maximum MFLOPS measured by ATLAS [147] by the maximum memory band width (MB) in megabytes per second measured with STREAM [148]. This factor states the relative cost of arithmetic calculations compared to memory access.

Type	Speed	RAM	GCC	MFLOPS	MB/s	Balance factor
P4	1 x 2.8 GHz	2 GB	4.0.3	2310.9	3045.8	0.7
AMD64	1 x 2.2 GHz	2 GB	3.4.4	3543.0	2667.7	1.2
IBMP655	8 x 1.5 GHz	64 GB	4.0.1	16361.7	4830.4	3.4
G5	4 x 2.5 GHz	8 GB	4.0.0	24434.0	3213.3	7.6
Intel Core E6600	2 x 2.4 GHz	2 GB	4.1.2	12404.0	4047.2	3.1
AMD X2 5600+	2 x 2.8 GHz	2 GB	4.1.2	17773.45	3692.2	4.8

8.4.1 GTL's Performance

The first analysis concerns to the topological traversal for different dimensional cell complex types. First a typical representative of a 0-cell complex, e.g., an array, is traversed. The C++ STL containers, such as vector or list, are representatives and are schematically depicted in Figure 8.11, where the points represent the cells on which data values are stored.

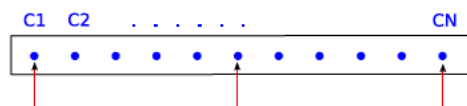


Figure 8.11: Topological traversal of vertices.

To investigate the abstraction penalty of the generic code a simple C implementation is used without any generic overhead compared to the GTL. The next code snippet presents the C source code for a three-dimensional 0-cell complex (cube) and is followed by the generic approach:

```

for (i3 = 0; i3 < sized3; i3++)
  for (i2 = 0; i2 < sized2; i2++)
    for (i1 = 0; i1 < sized1; i1++)
    {
      //operations, use i1, i2, i3
    }

```

C approach for 0-cell traversal

```

vit1 = container.vertex_begin();
vit2 = container.vertex_end();
for (;vit1 != vit2; ++vit1)
{
  //operations, use *vit1
}

```

Generic approach for 0-cell traversal

Figure 8.12 and Figure 8.13 present results obtained on four computer systems. As can be seen, the performance is comparable on different systems without incurring an abstraction penalty from the highly generic code when compared to a simple C implementation.

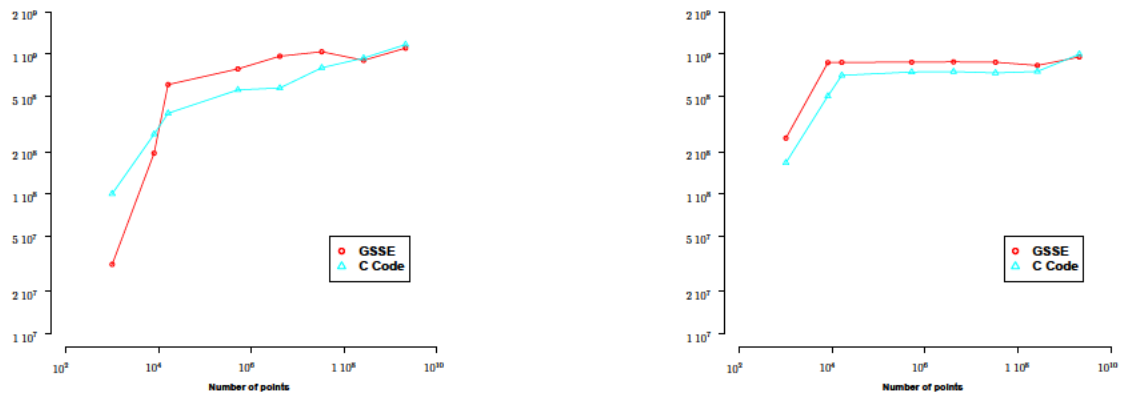


Figure 8.12: 0-cell traversal on the P4 (left) and the AMD64 (right); the units are iterations per second.

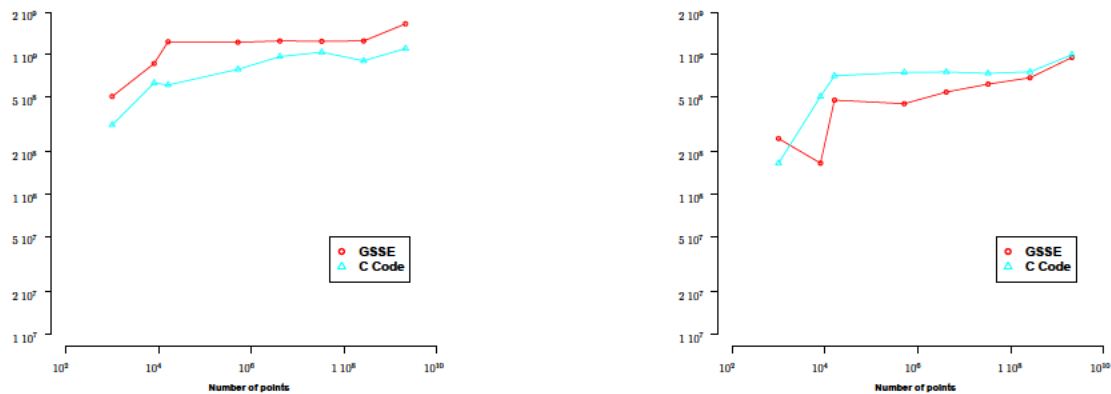


Figure 8.13: 0-cell traversal on the G5 (left) and the IBM (right); the units are iterations per second

To provide a more complex example, the traversal mechanisms of the BGL data structures are compared to the approach given here. Figure 8.14 presents a 1-cell complex type with an edge traversal mechanisms.

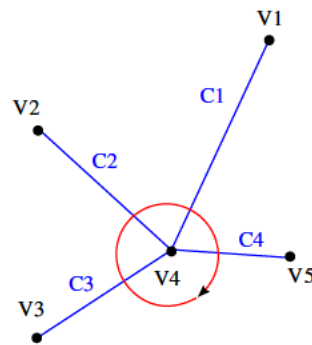


Figure 8.14: Topological traversal of cells for a one-dimensional cell complex.

The source code snippets from Section 8.1.2 are used as the corresponding implementations.

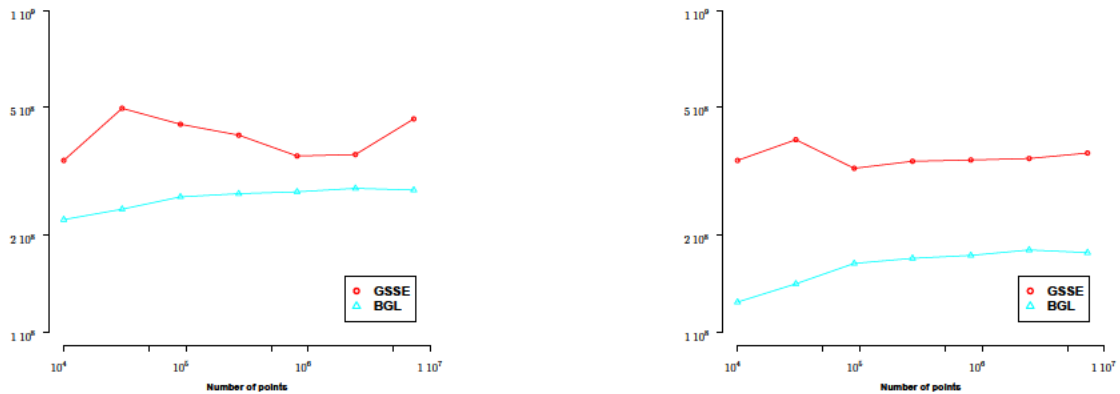


Figure 8.15: Incidence traversal for the BGL and the GTL approach on the P4 (left) and the AMD64 (right); the units are iterations per second

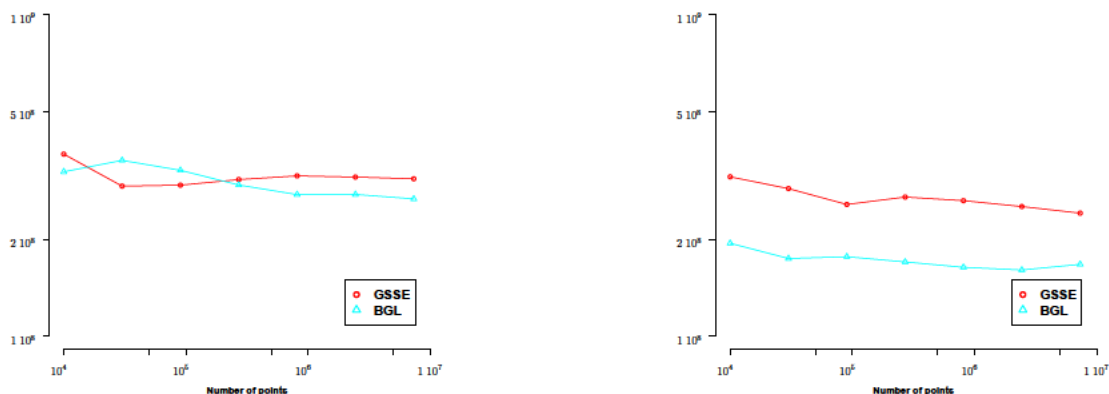


Figure 8.16: Incidence traversal for the BGL and the GTL approach on the G5 (left) and the IBM (right); ; the units are iterations per second

Figure 8.15 and Figure 8.16 present the run-time results for edge-on-vertex traversal. The overall run-time behavior of the GTL is comparable, at times even superior, to that of the BGL.

8.4.2 GFL's Performance

The second analysis is related to the analysis of the abstraction penalty of the generic functor library. Due to the use of meta-programming and its evaluation at compile-time, the calculation associated with the specified equations can be highly optimized for the whole application by the compiler [149]. As a base foundation the Blitz++ benchmark system is used which uses a vector operation (DAXPY) $f = \alpha \cdot x + y$, evaluated with different vector sizes for x, y . The influence on different machines can be clearly separated into the system-related differences and the compiler-based differences. Therefore, the first benchmarks are related to different compiler versions of the GCC. Next, a comparison of all machines is given.

For a rigorous analysis, different implementations in C++ are analyzed and compared to a hand-optimized Fortran 77 implementation on different computer architectures:

- The GNU GCC `valarray` which is a standardized data-structure representing a mathematical vector.
- Next the Blitz++ Version 0.9 [150] library, is utilized, which introduced high performance calculation comparable to Fortran 77 directly in C++.
- Finally, a naive C++ implementation based on a `std::vector<T>` is used that creates two temporary objects, one for addition and one for assignment.
- The GFL of the GSSE.

Figure 8.17 and Figure 8.18 present the given benchmarks for the P4 and the AMD X2.

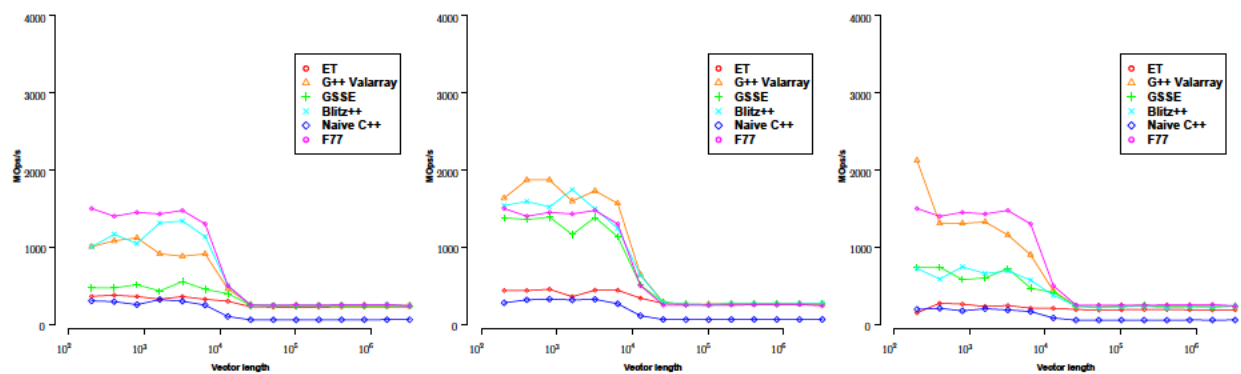


Figure 8.17: Evolution of compiler enhancements on the P4 for the DAXPY benchmark. Left: GCC 4.0.4; middle: GCC 4.1.2; right: GCC 4.2.1

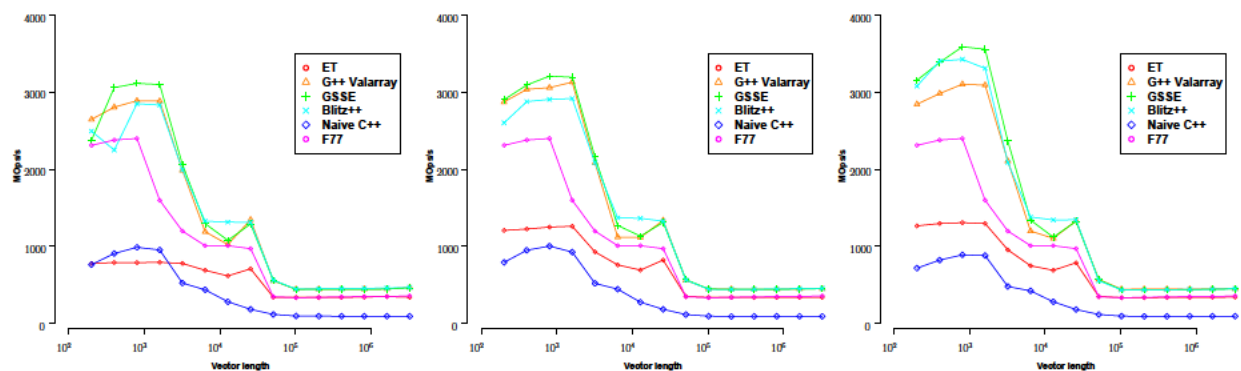


Figure 8.18: Evolution of compiler enhancements on the AMD X2 for the DAXPY benchmark. Left: GCC 4.0.4; middle: GCC 4.1.2; right: GCC 4.2.1

As can be seen, the GSSE performs well with recent compilers such as the GCC 4.2.1 compiler generation. Another important point can be seen. The overall computer performance has doubled within the last two years.

To analyze the dependence on the machine in more detail, the best possible compiler generation is used on all the machines and the results are provided in Figures 8.19 and 8.20.

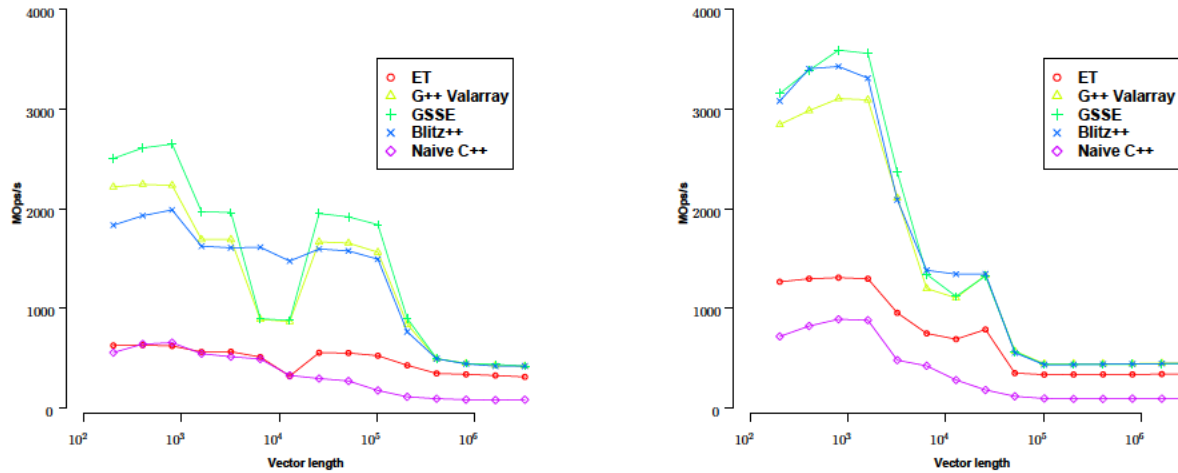


Figure 8.19: Best compiler performance for the Intel Core (left) and the AMD X2 (right), both using the GCC 4.2.1.

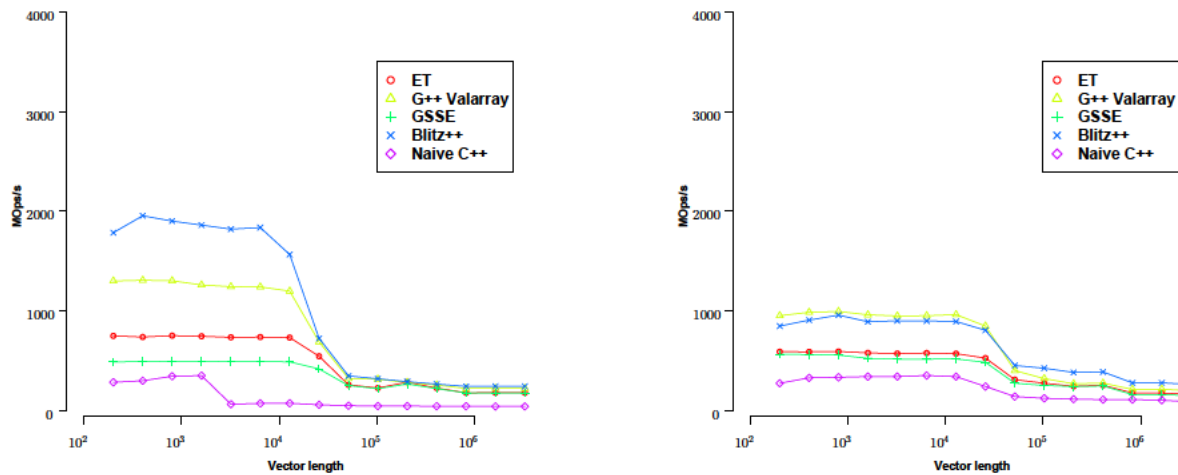


Figure 8.20: Best compiler performance for the G5 (left) and the IBM (right), both using the system compiler.

Although the machine information of the Intel Core is superior to the AMD X2 system, the overall benchmarks show that the GSSE performs best on the AMD X2 with a recent compiler. A wide variety of optimization possibilities are thereby available. Recent developments and research have focused on the process of automation for this type of optimization for several architectures and compilers by the introduction of a concept called composer [151].

Chapter 9

Generic Application Design

Generic application design deals with the conceptual categorization of computational domains, the reduction of algorithms to their minimal conceptual requirements, and strict performance guarantees. Here, the assembly of the given practical building blocks (see Part II) and the transformation of GSSE components into actual applications are presented. Several examples are given to demonstrate that the proposed concepts are functional and that the implementation effort is greatly reduced. The application areas are divided into different areas, such as wave equation, diffusion simulation, and electrostatic simulation. A system of coupled non-linear equations from the field of TCAD's device simulation is used to demonstrate the approach, and several results for the investigated equations are presented. All examples use automatic model quantity evaluation, while communication is accomplished using the GFI module. Each of the applications presented is implemented following the same patterns:

- Specification of required quantities by corresponding identifier, e.g., `strings`.
- Use of a linear or non-linear solution scheme by the selection of the appropriate solver scheme.
- Implementation of pre-processing sections.
- Implementation of the application code.
- Implementation of post-processing sections.

The actual application code requires normally not more than hundreds of source lines for each application, and the pre-processing and post-processing facilities can be reused. Related to the finite element scheme, GSSE contains pre-calculated stencil matrices for triangles, rectangles, tetrahedra, and cubes. Cell topologies (posets) are stored for from one-dimensional up to four-dimensional simplices and cuboids and can thereby be optimized at compile-time. Compile-time as well as run-time algorithms are available to generate higher dimensional posets. Interpolation algorithms are available for linear, quadratic, and exponential methods.

9.1 Visual Programming

Based on the concepts of topological manifolds, the separation facilities of the fiber bundle approach, and the correlated programming concepts introduced in Section 5.2, new possibilities for implementing the programming concepts can be derived. One of these concepts is visual programming, which is not only the illustration of a final structure, but can also be used in abstract ways, such as a convergence analysis of non-linear algebraic solver steps, or to define the model by identifying different types of boundary conditions. In addition to a real-time interface to update the visualization at each time step, the following abstract quantities are necessary, such as *equation type*, *coupling coefficients*, *domain identifier*, and *boundary conditions*. Based on these abstract quantities, the application design can not only be simplified, but the search for errors can also be enhanced by this type of programming.

A complete example of an abstract process which greatly benefits from the support of visual programming is given next. Each step of the process of solving a non-linear equation is available to be examined for errors in the implementation. The visualization of the calculation is available in real time, making it possible to observe the evolution even of a non-linear solution process. It proved to be invaluable for the adjustment of simulation parameters. The leftmost part of Figure 9.1 shows the initial potential, while the rightmost depicts the final solution. The center image shows an intermediate result that has not yet fully converged.

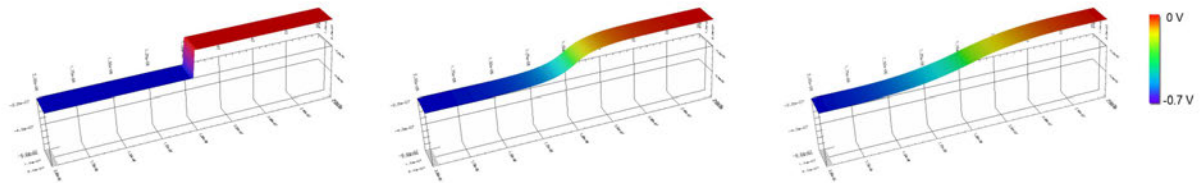


Figure 9.1: Potential of a PN diode during different stages of the Newton iteration. From initial (left) to the final result (right).

In contrast, an equation system which does not converge is given in Figure 9.2, where small oscillations can be observed. This problem can be caused by wrong input parameters, or by an inappropriate time stepping procedure.

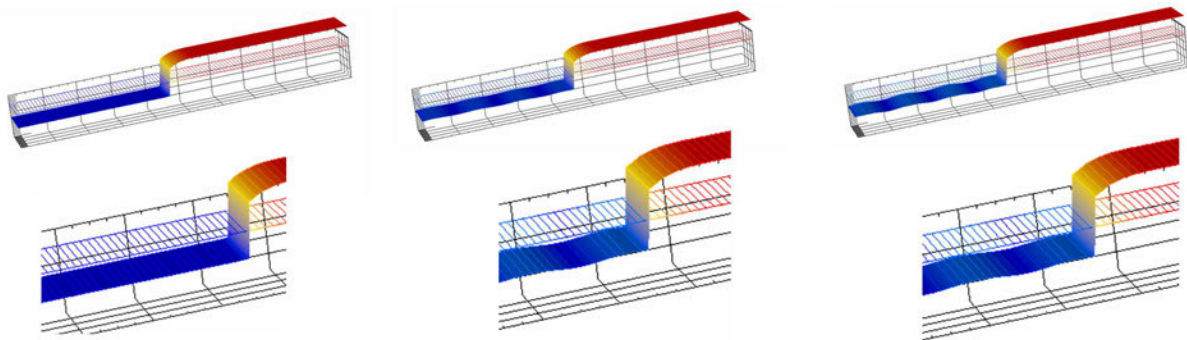


Figure 9.2: Visualization of non-converging process. Here the potential is illustrated where small oscillations can be observed from left to right.

Figure 9.3 depicts the complete breakdown of the solution procedure. As can be seen, visual programming greatly supports the development of applications.

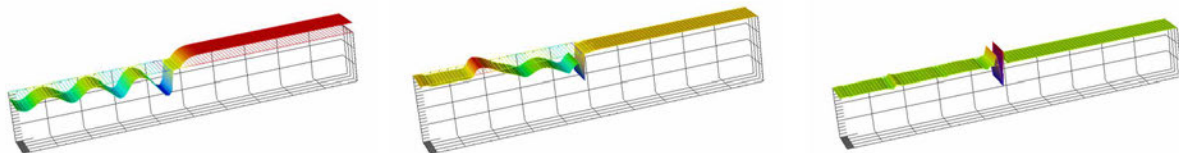


Figure 9.3: Illustration of a complete breakdown of the solution procedure (from left to right).

9.2 Wave Equation

An example using the discrete concepts of electromagnetics in conjunction with the comprehensive operations of the GSSEs provided. It translates the continuous formulation of Maxwell's equations given in Section 2.2 with internally oriented \mathbf{E} , \mathbf{B} and externally oriented \mathbf{H} , \mathbf{D} fields, including their linking relations

$$\text{curl}(\mathbf{E}) = -\partial_t \mathbf{B} \quad (9.1)$$

$$\text{curl}(\mathbf{H}) = \partial_t \mathbf{D} \quad (9.2)$$

into a discrete setting, by applying the discrete concepts of chains and cochains (Section 2.4 and Section 2.7). Figure 9.4 presents the discrete form of Faraday's and Amperé's law.

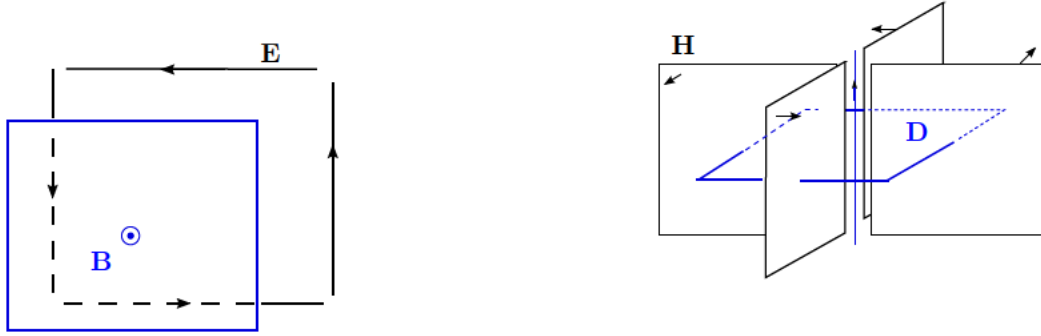


Figure 9.4: The left figure depicts Faraday's law by the corresponding projection onto a finite cell, whereas the right figure illustrates Amperé's law.

Using a projection of the averaged field components onto 2-cells, local representatives of the global quantities are obtained. See Section 3.4.3 for more details. The following equation expresses this fact:

$$\begin{aligned} \Phi_{B,x|(i+1/2,j,k)}^{n+1} &= \Phi_{B,x|(i+1/2,j,k)}^n + \frac{\Delta x \Delta t}{\Delta \tilde{y} \Delta \tilde{z}} \quad (9.3) \\ &\left(\frac{1}{\varepsilon_{|(i+1/2,j,k+1/2)}} \Psi_{D,x|(i+1/2,j,k+1/2)}^{n+1/2} - \frac{1}{\varepsilon_{|(i+1/2,j,k-1/2)}} \Psi_{D,x|(i+1/2,j,k-1/2)}^{n+1/2} \right) \\ &\quad - \frac{1}{\varepsilon_{|(i+1/2,j+1/2,k)}} \Psi_{D,x|(i+1/2,j+1/2,k)}^{n+1/2} + \frac{1}{\varepsilon_{|(i+1/2,j-1/2,k)}} \Psi_{D,x|(i+1/2,j-1/2,k)}^{n+1/2} \end{aligned}$$

By utilizing the quantity accessor mechanisms and the traversal operations of the GSSE, the equation can be rewritten as a discrete formulation, where the the electric field quantity is located on edges E_e and the magnetic field quantity is located on facets B_f . The permittivity, permeability, and spatial resolution are omitted to emphasize the topological traversal capabilities:

$$\mathcal{L}_{B.fdm}(B) \equiv \frac{B_f - B_f^{\text{old}}}{\Delta t} = \Delta_{f \rightarrow e}[E_e], \quad (9.4)$$

where $f \rightarrow e$ denotes the traversal of all edges incident to a face. The topological traversal mechanism is presented in the following code snippet, where the constitutive laws are used to interpolate the corresponding quantities.

```
// ..
H += delta_t * sum<facet, -1>() [ E ]
E += delta_t * sum<edge, +1>() [ H ]
// ..
```

The x -component and y -component of the final vector field \mathbf{E} is depicted in Figure 9.5 and Figure 9.6 for a three-dimensional calculation. The x - y plane with a spatial dimension of 10×10 units at the bottom uses a simple harmonically oscillating quantity and is also used as a Dirichlet contact. Neumann boundary conditions are applied to the remaining boundary planes.

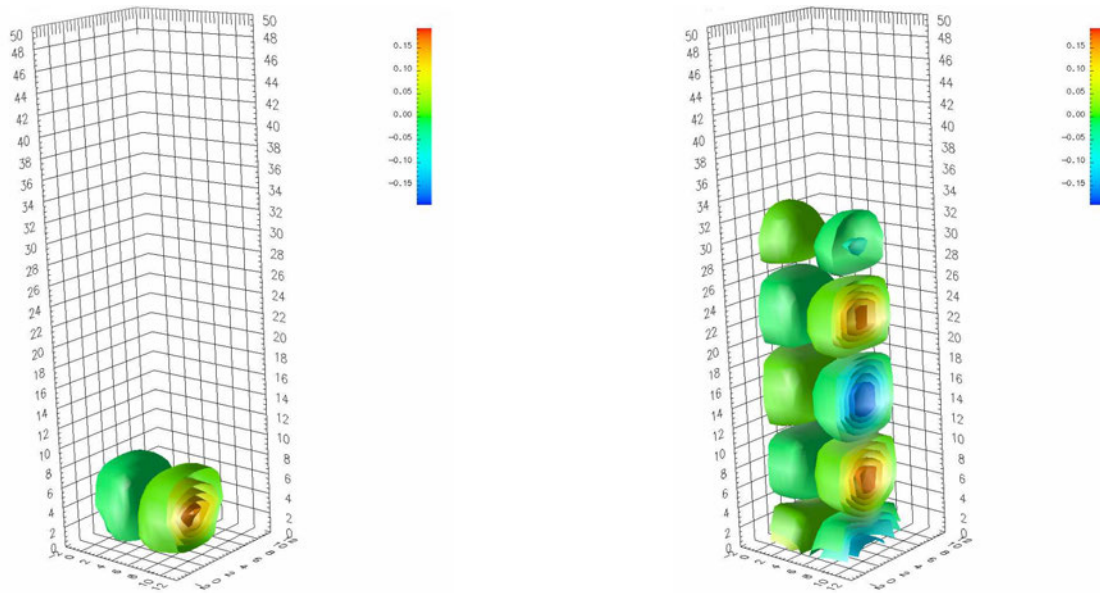


Figure 9.5: Illustration of the x -component of \mathbf{E} with a harmonic oscillating source in the x - y plane at two different time steps.

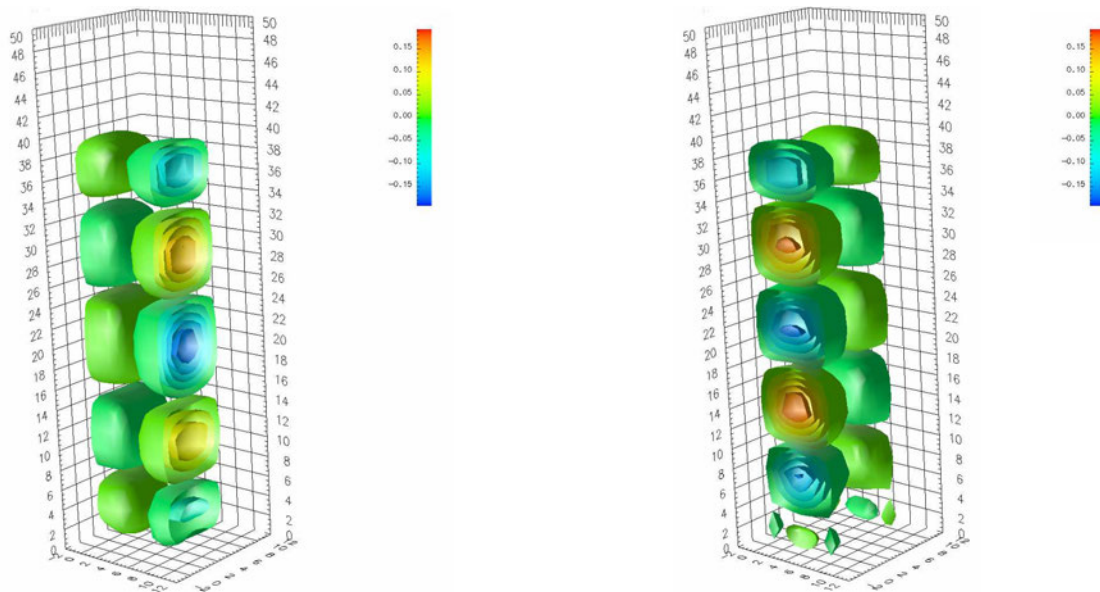


Figure 9.6: Wave equation with a harmonic oscillating source in the x - y plane where the source is switched of (left figure). The y -component of \mathbf{E} is depicted on the right side.

9.3 Diffusion Simulation

The accurate simulation of the dopant atoms and their spatial distribution within devices is an important issue in modern TCAD. The characteristics of the manufactured device is determined by these issues, e.g., if devices rely on very shallow junctions to minimize the so-called short-channel effects [152]. Diffusion does not only occur in place of the introduction of dopant atoms, the so-called ion-implantation step, but also to electrically activate the implanted atoms. A comprehensive study of currently developed diffusion simulation methods is given in [69]. The simulations given here use the finite volume discretization method with very basic models to document the potential of GSSE as a mathematical framework for generic parabolic PDEs. The transformation of the given equations into executable C++ code and the corresponding alternative time discretization steps are the main parts, whereas the results show the feasibility.

9.3.1 The Equations

This example introduces a parabolic type of a PDE. As introduced in Section 3.1, time can be modeled by different numerical approximations. Here, an implicit and an explicit method are presented [67]. The following differential equation is used to describe a generic parabolic PDE, e.g., a diffusion process with boundary conditions:

$$\operatorname{div}(u) + \alpha \partial_t u = 0 \quad \text{on } \Omega \quad (9.5)$$

$$\partial_n u = 0 \quad \text{on } \Gamma_1 \quad (9.6)$$

$$u = f(x, y, z) \quad \text{on } \Gamma_2 \quad (9.7)$$

By the use of the concepts presented the discretized formulation by the finite volume method is rendered similar to a generic elliptic equation, extended by an additional time derivative

$$\mathcal{L}_{\text{p-fvm}}(u) \equiv \sum_{v \rightarrow e} (\Delta_{e \rightarrow v} u) \frac{A}{d} + \frac{u - u_{\text{old}}}{\Delta t} = 0,$$

where Δt denotes the time step and u_{old} stands for the solution function u in the prior time step. This scheme is known as the implicit Euler time discretization. The explicit time discretization, which avoids the solution of an equation system but becomes unstable if time steps are too large, is formulated analogously:

$$\mathcal{L}_{\text{p-fvm}}(u) \equiv \sum_{v \rightarrow e} (\Delta_{e \rightarrow v} u_{\text{old}}) \frac{A}{d} + \frac{u - u_{\text{old}}}{\Delta t} = 0.$$

The GSSE diffusion application is available for two and three dimensions, as well as for structured grids and unstructured meshes, which are demonstrated in the following by two different examples.

9.3.2 Transformation into the GSSE

By using the finite volume method and the specification mechanism enabled by the GSSE, the next code snippet shows a possible implementation. Here, the boundary evaluation is also shown where the conditional statements are based on abstract quantities, as illustrated in Section 9.1. First, the implicit time discretization and the corresponding specification is given. The `domainType` tag is also available by the GSSE.

```

if (domain.get_type(*vit) == domaintype_interior_neumann)
{
  equation = ( sum<vertex,edge>()
              [
                sum<edge,vertex>() [ u ] * area / dist
              ] + (u - u_old ) / delta_t
              )(*vit);
}
else if (domain.get_type(*vit) == domaintype_dirichlet)
{
  equation = ( u - u_bnd )(*vit);
}

```

The Dirichlet boundary condition is specified by the abstract boundary condition quantity `u_bnd`. An explicit time discretization is presented next, where the reusability of developed components can be easily observed. Almost the entire implementation can be reused, whereas the functional paradigm enables the exchange of only one quantity accessor.

```

equation = ( sum<vertex,edge>()
            [
              sum<edge,vertex>() [ u_old ] * area / dist
            ] + (u - u_old) / delta_t
            )(*vit);

```

9.3.3 Results

In two dimensions, a structured grid (see Figure 9.7) is used, and the influence of the boundary conditions is shown. The left side of the simulation domain ($x = 0$) is used as a constant source of diffusion particles, thereby creating a steady flow into the area. All other boundaries are modeled by a Neumann boundary condition.

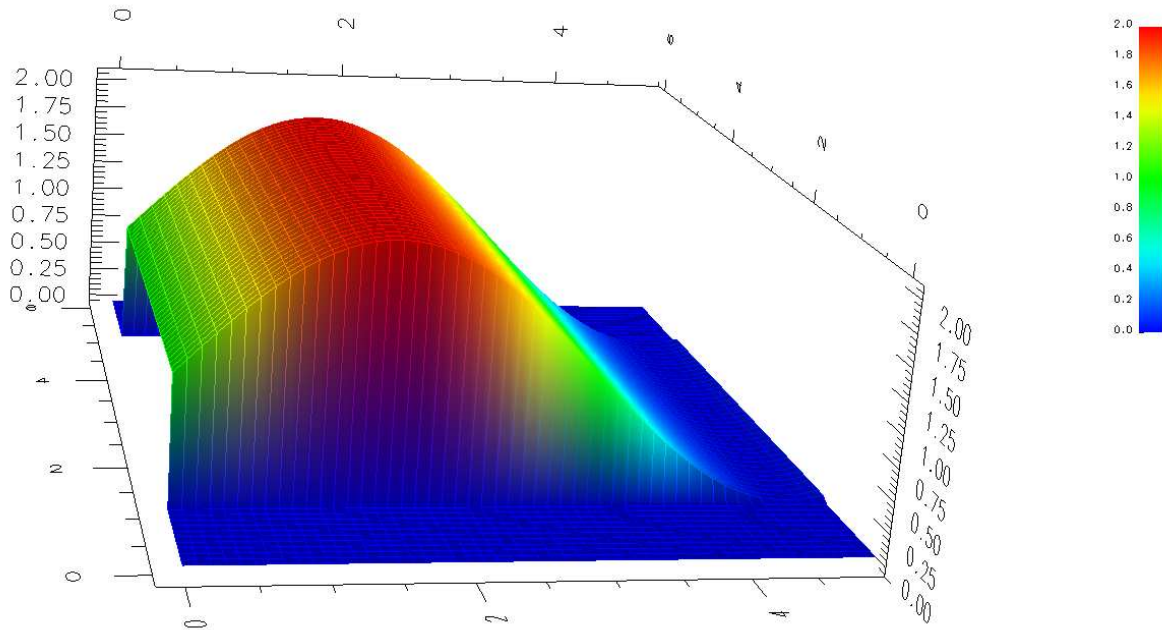


Figure 9.7: Two-dimensional structured grid with an initial doping profile (concentration in cm^{-1}).

The simulation results for two subsequent time steps are given in Figure 9.8 and Figure 9.9.

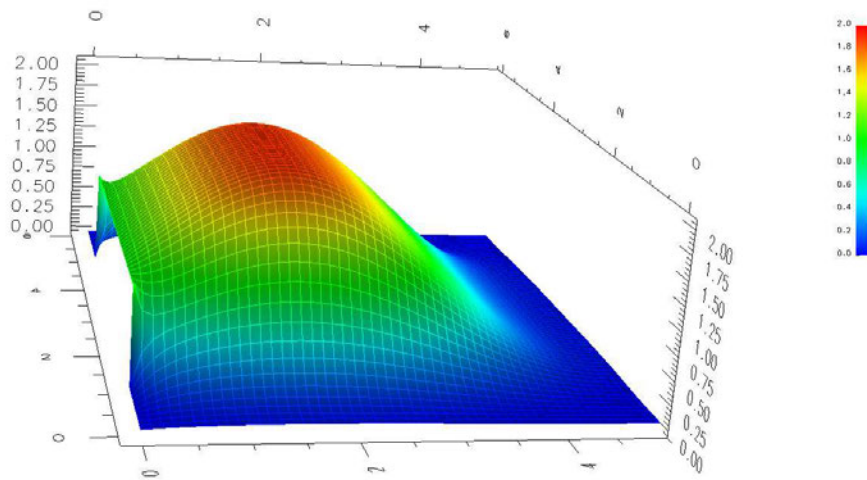


Figure 9.8: Two-dimensional diffusion simulation for a structured grid after 200 time steps.

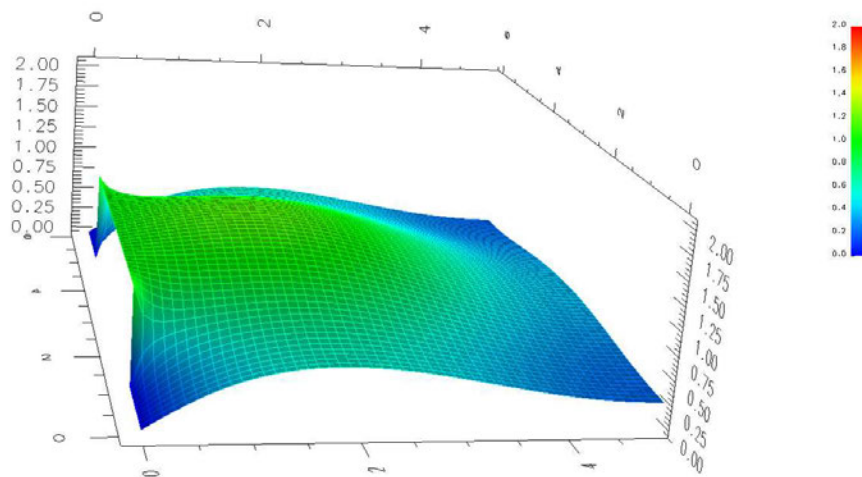


Figure 9.9: Two-dimensional diffusion simulation for a structured grid after 2000 time steps.

The three-dimensional unstructured mesh example, depicted in Figure 9.10, is based on an analytic ion-implantation module, which also generated the point cloud automatically [153]. The corresponding mesh generation, input preparation for the analytic ion implantation and dual mesh calculation, and simulation times are given in the following table (AMD X2, see Section 8.4 for more details):

Task	Number of points	Time [s]
mesh generation	$3 \cdot 10^3$	2.7
input preparation		18.1
diffusion simulation		18.2

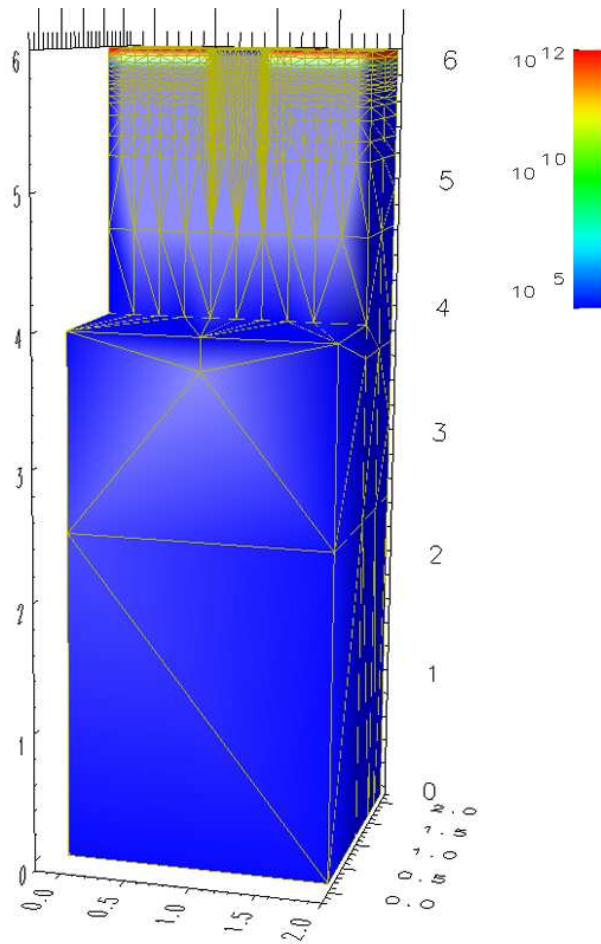


Figure 9.10: Three-dimensional device structure with an initial phosphorus doping profile (concentration in cm⁻¹).

Two subsequent diffusion steps are depicted in Figure 9.11, which are used as a rapid annealing step after ion implantation [69].

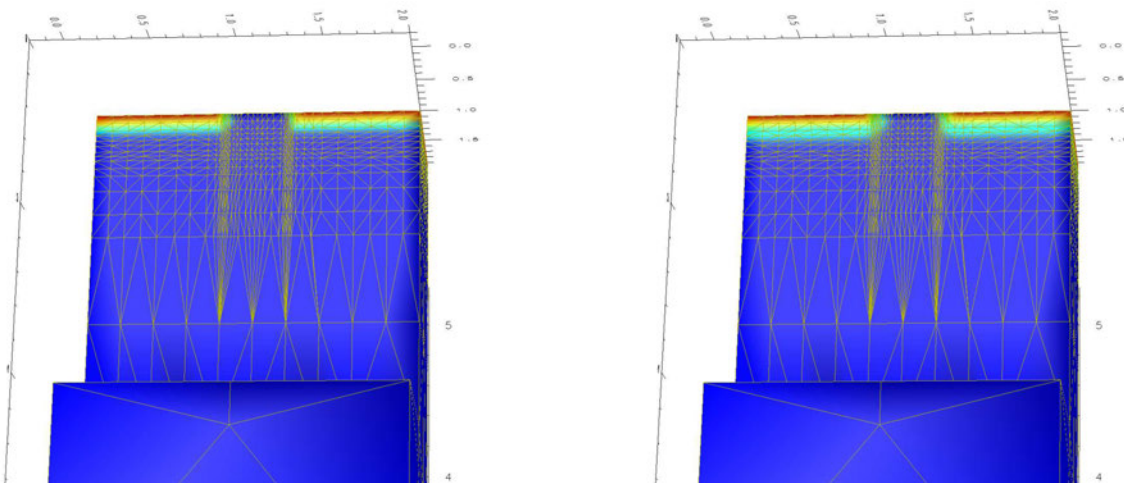


Figure 9.11: Three-dimensional diffusion simulation for a device structure with an initial doping profile. Two subsequent simulation steps are depicted.

9.4 Device Simulation

Semiconductor devices have become an ubiquitous commodity and one is used to expect a constant increase of device performance at higher integration densities and falling prices. It is this quest for ever decreasing device dimensions and faster switching speeds that results in ever growing requirements on simulation methodology and thereby on the application design. While computer performance is steadily increasing the additional complexity of these simulation models easily outgrows this gain. This example presents a non-linear problem, discretized by the finite volume method, to show the high expressiveness of the resulting application code.

9.4.1 The Equations

The drift diffusion model which can be derived from Boltzmann's equation by applying the method of moments [132] has been used very successfully in semiconductor device simulations. Equations 9.8 show the resulting current relations. These equations are solved self-consistently by Poisson's equation, given in Equation 9.9.

$$\begin{aligned}\operatorname{div}(\mathbf{J}_p) &= \operatorname{div}(q p \mu_p \operatorname{grad}(\varphi) - q \mu_p U_{\text{th}} \operatorname{grad}(p)) = -q R \\ \operatorname{div}(\mathbf{J}_n) &= \operatorname{div}(q n \mu_n \operatorname{grad}(\varphi) + q \mu_n U_{\text{th}} \operatorname{grad}(n)) = q R\end{aligned}\quad (9.8)$$

$$\operatorname{div}(\operatorname{grad}(\varepsilon \varphi)) = -q(p - n + C) \quad (9.9)$$

To this end, the equations are discretized using the Scharfetter-Gummel [154] scheme, resulting in a non-linear equation system of the form [132]

$$J_{n,ij} = \frac{q \mu_n U_{\text{th}}}{d_{ij}} (n_j B(\Lambda_{ij}) - n_i B(-\Lambda_{ij})) \quad (9.10)$$

$$\Lambda_{ij} = \frac{\varphi_j - \varphi_i}{U_{\text{th}}} \quad B(x) = \frac{x}{e^x - 1} \quad (9.11)$$

The discretization of the given problem using finite volumes yields:

$$\operatorname{div}(x) \approx \sum_{v \rightarrow e} x \frac{A}{V} \quad \operatorname{grad}(x) \approx \frac{1}{d_{e \rightarrow v}} \Delta x \quad (9.12)$$

In order to make maintenance of the code as easy as possible and to achieve a maximum of flexibility, it is important to keep the code expressive.

9.4.2 Transformation into the GSSE

The transformation of the presented equations into C++ code is presented in the following source snippet.

By this transformation a link between the still continuous formulation of the equation and a specific chain complex of the simulation domain is formed. As can be observed, the implementation above makes no assumptions about the dimension of the problem and is therefore suitable for arbitrary dimensions. It should be noted that any scaling in the mathematical sense was not carried out in order to highlight the expressiveness of GSSE in this simple example.

The Bernoulli function, given in Equation 9.11, is mapped to `Bern`, where the discretized differential operators are easily recognized as the `diff` and `sum` constructs. This dimension-independent formulation is made possible only by the combination of the separation of algorithms from data sources by the use of

```

// Poisson equation
//
equation_pot =
(
  sum<vertex, edge>()
  [
    diff<edge, vertex>()
    [
      pot
    ] * area / dist
  ] + ( p -n + C ) * ( vol * q / eps )
) (*vit);

// Continuity equation for electrons
//
equation_n =
(
  sum<vertex, edge>()
  [
    diff<edge, vertex>
    (
      -n_quan*Bern(diff<edge, vertex>() [ pot ]/U_th),
      -n_quan*Bern(diff<edge, vertex>() [-pot ]/U_th)
    ) * q * mu_n * U_th * area/dist
  ]
) (*vit);

// Continuity equation for holes
equation_p =
(
  sum<vertex, edge>()
  [
    diff<edge, vertex>
    (
      p_quan*Bern(diff<edge, vertex>() [-pot]/U_th),
      p_quan*Bern(diff<edge, vertex>() [ pot]/U_th)
    ) * q * mu_p * U_th * area/dist
  ]
) (*vit);

```

traversal mechanisms (Section 8.1) and by the employment of functional objects (Section 8.2). Here, it is important to highlight that the entire application is available at compile-time and can thereby be highly optimized at once. Calculation results based on the techniques presented are provided in the following section.

9.4.3 Results

To demonstrate that the proposed computational scheme is indeed operational a two-dimensional PN-diode is analyzed. Two neighboring regions of different doping, one doped with acceptors, the other doped with donors, result in the well-known rectifying PN diode. Such a two-dimensional silicon PN-diode is simulated based on the given code section. The complete application uses no more than 200 lines of source code for all different dimensions and topological input structures. Figure 9.12 depicts the input structure for two and three dimensions.

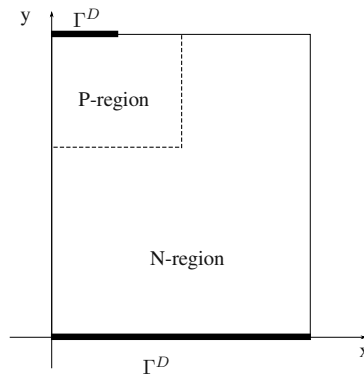


Figure 9.12: Domain for a two-dimensional PN-junction diode.

The concentration of acceptors has a maximum of $N_A = 10^{16} \text{cm}^{-3}$, the donors concentration has a maximum of $N_D = 10^{19} \text{cm}^{-3}$. The netto doping profile ($N_D - N_A$) of the device presented in Figure 9.12 is illustrated in Figure 9.13.

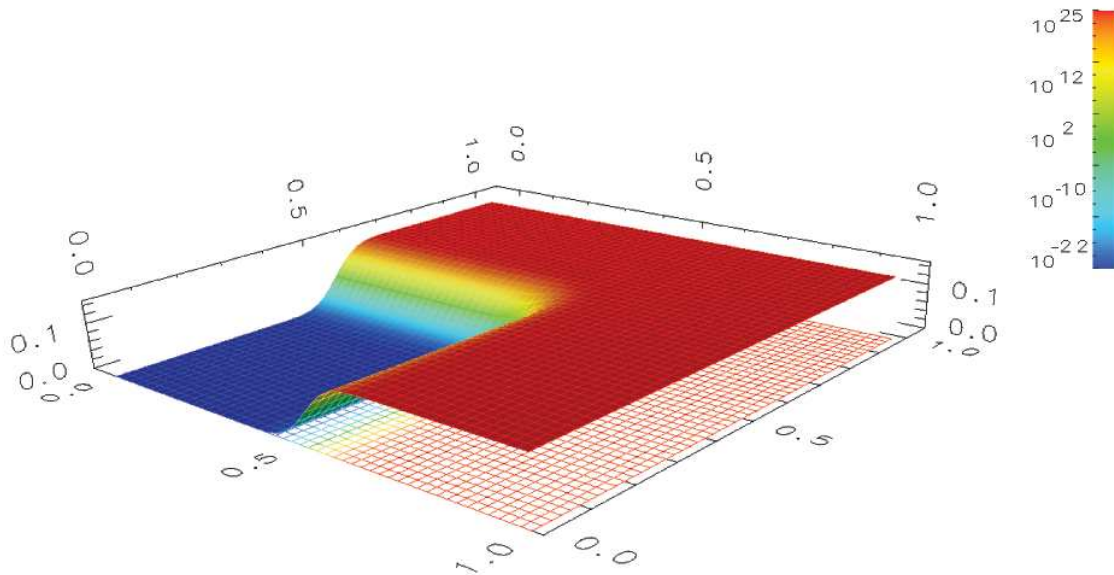


Figure 9.13: Netto doping concentration of the two-dimensional PN diode. Donors are given in red, acceptors are blue; units are given in parts per cubic meter.

The shape of the initial barrier present without external bias and for reverse mode can be seen in Figure 9.14. When operated in reverse the current is blocked, as is illustrated by the potential when the initial barrier is increased, as shown in Figure 9.15. In forward operation mode, current can run freely, which is reflected by the lowering of the initial potential barrier, as shown in Figure 9.16. The space-charge distribution for reverse mode can be seen in Figure 9.17 and for forward mode in Figure 9.18.

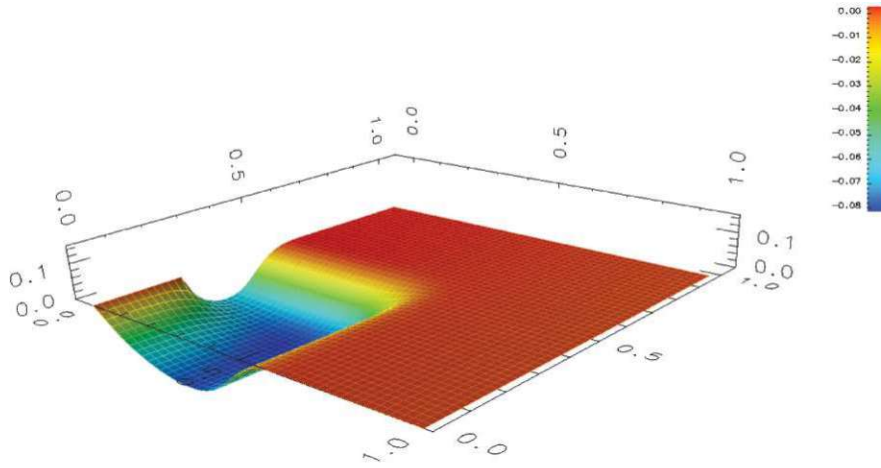


Figure 9.14: Two-dimensional diode with potential distribution for equilibrium mode; units given in Volt.

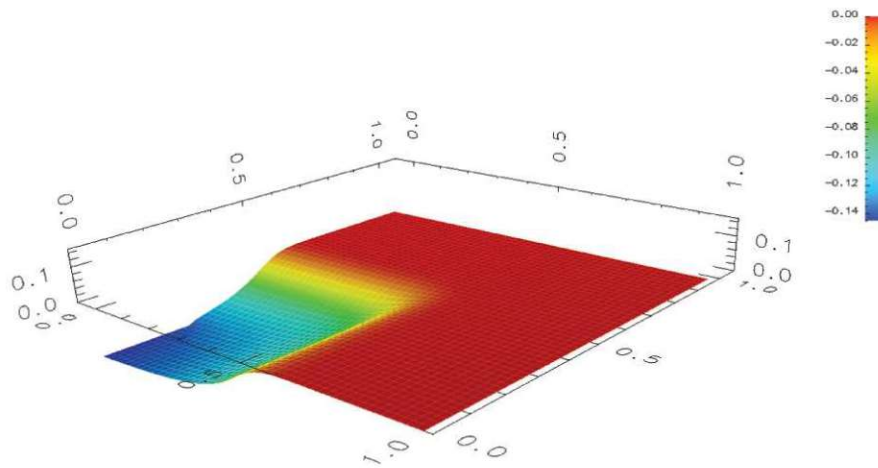


Figure 9.15: Two-dimensional diode with potential distribution for reverse mode; units given in Volt.

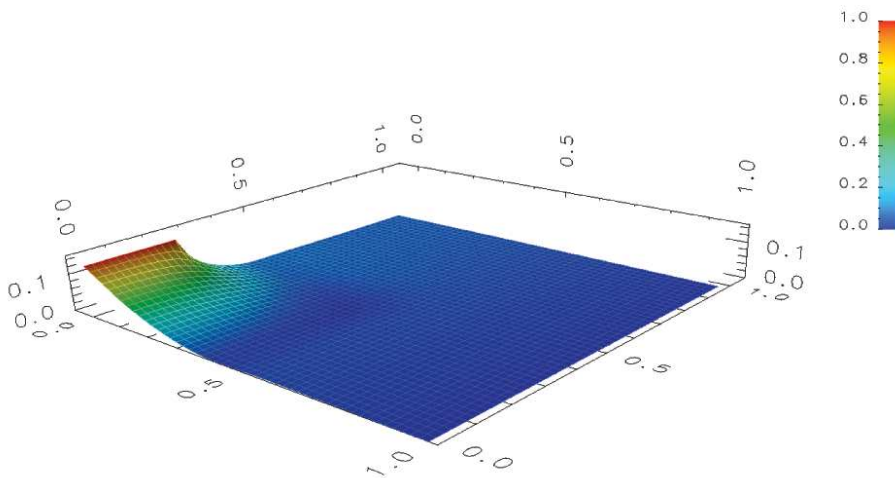


Figure 9.16: Two-dimensional diode with potential distribution for forward mode; units given in Volt.

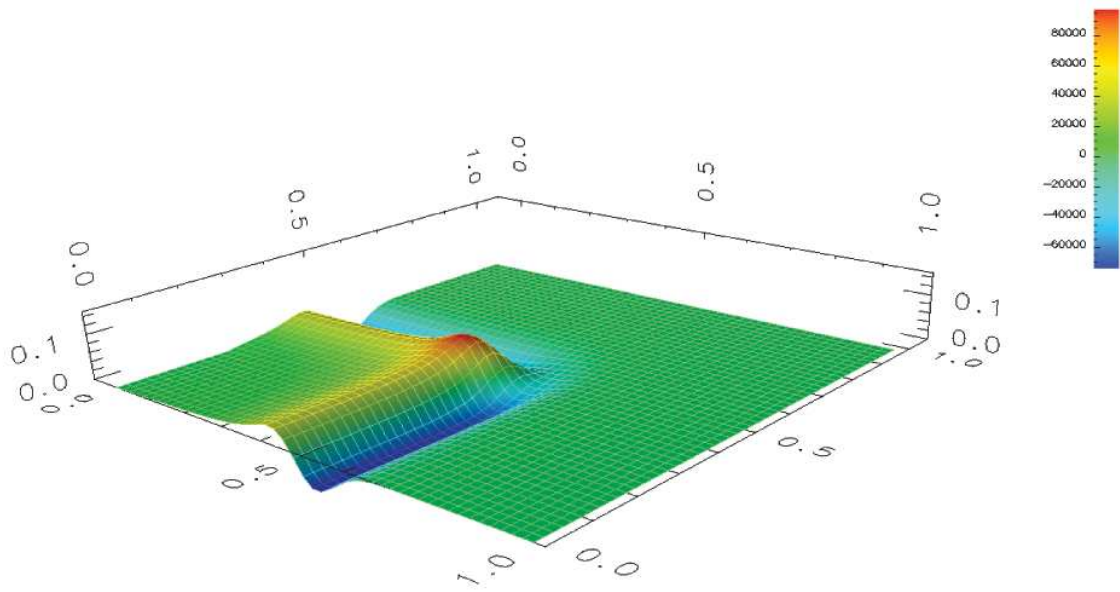


Figure 9.17: Two-dimensional diode with charge distribution for reverse mode; units given in parts per cubic meter.

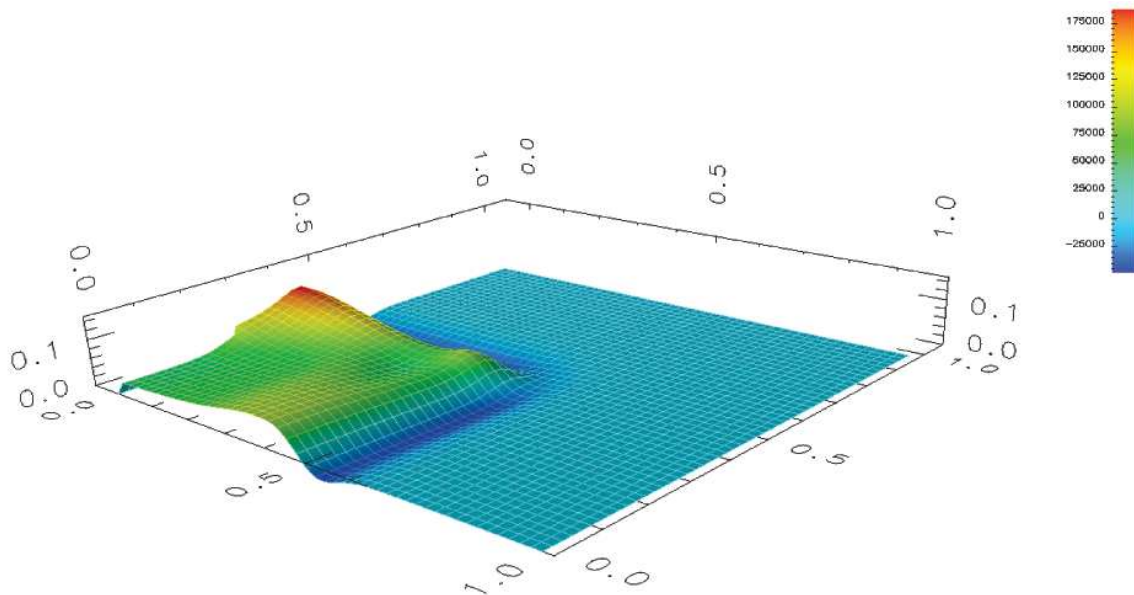


Figure 9.18: Two-dimensional diode with charge distribution for forward mode; units given in parts per cubic meter.

Chapter 10

Smart Analysis Package

Downscaling of integrated circuits to the deep submicron regime and beyond increases the influence of interconnects on circuit behavior drastically. Parasitic effects are becoming more and more important as devices get faster and line widths smaller. These effects become the limiting factor for further improvements of circuit speed. An important part of interconnect analysis is the extraction of parameters, e.g. capacitance extraction [155], or resistance analysis.

The high performance SAP package [113, 114], reviewed in Section 5.4.1, has been developed for these tasks, especially for thermal (STAP) and capacitive (SCAP) problems. This package was manually tuned to yield excellent performance and portability but requires a considerable amount of maintenance by the development group. The here presented approach is based on GSSE modules. Two examples from this area are analyzed in the following sections. A comprehensive performance analysis is also given, whereby the next section introduces an important extension of the smart analysis package known as structure modeling.

10.1 Structure Modeling

To overcome the difficulties encountered in the modeling of three-dimensional structures, the Vienna Geometry Modeler (VGM[137]) was developed. It enables an efficient means for structure generation for interconnect simulation, and can be seen as a successor of LAYGRID [114] from the SAP-package. The creation of objects is based on a constructive solid geometry (CSG) approach which models complex three-dimensional shapes using simple objects. The agglomerations of these simple objects then provide suitable simulation structures. To give a brief overview of the syntax and mannerism of VGM, a comparison of a LAYGRID file and a VGM input file (Figure 10.1) are presented.

An additional feature of the VGM specification is that tapering angles of the given sidewalls can be easily adjusted. Only the corresponding skewness parameters have to be adjusted, given in the following lines:

```
//                                     xskew  yskew
tapered { VB1 dummy { 4.6 2.6 0.9}{0.4 0.4 0.7} {0.0} {0.0}}
tapered { VB1 dummy { 4.6 2.6 0.9}{0.4 0.4 0.7} {0.2} {0.2}}
```

```

lengthunit { 1.0 um }

mask { MASK0
  rectangle { FX0 SIOX { 0.0 0.0 } { 10.5 6.5 } }
}
mask { MASK1
  rectangle { FX1 SIOX { 0.0 0.0 } { 10.5 6.5 } }
  rectangle { FX2 ALX { 0.0 2.5 } { 6.0 1.5 } }
  rectangle { FX2C ALX { 0.0 2.5 } { 1.0 1.5 } }
}
mask { MASK2
  rectangle { FX3 SIOX { 0.0 0.0 } { 10.5 6.5 } }
  rectangle { FX4 AVX { 4.6 2.6 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 5.5 2.6 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 4.6 3.5 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 5.5 3.5 } { 0.4 0.4 } }
}
mask { MASK4
  rectangle { FX3 SIOX { 0.0 0.0 } { 10.5 6.5 } }
  rectangle { FX4 AVX { 4.6 2.6 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 5.5 2.6 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 4.6 3.5 } { 0.4 0.4 } }
  rectangle { FX4 AVX { 5.5 3.5 } { 0.4 0.4 } }
}
mask { MASK5
  rectangle { FX9 SIOX { 0.0 0.0 } { 10.5 6.5 } }
  rectangle { FX10 ALX { 4.5 2.5 } { 6.0 1.5 } }
  rectangle { FX10C ALX { 9.5 2.5 } { 1.0 1.5 } }
}
mask { MASK6
  rectangle { FX11 SIOX { 0.0 0.0 } { 10.5 6.5 } }
}
layerstructure {
  origin { 0 0 0 }
  plane { ----- }
  layer { MASK0 0.5 }
  plane { ----- }
  layer { MASK1 0.5 }
  plane { ----- }
  layer { MASK2 0.25 }
  plane { ----- }
  layer { MASK4 0.25 }
  plane { ----- }
  layer { MASK5 0.5 }
  plane { ----- }
  layer { MASK6 0.5 }
  plane { ----- }
}
  
```

```

line { OB dummy { 0 0 0 } { 10 6.5 2.5 } }
line { ODL dummy { -10 0 0 } { 10 6.5 2.5 } }
line { OXR dummy { 10 0 0 } { 10 6.5 2.5 } }
tapered { LB1 dummy { -1 2.5 0.5 } { 7 1.5 0.5 } { 0.0 } { 0.0 } }
tapered { LB2 dummy { 4.5 2.5 1.5 } { 6.5 1.5 0.5 } { 0.0 } { 0.0 } }

tapered { VB1 dummy { 4.6 2.6 0.9 } { 0.4 0.4 0.7 } { 0.0 } { 0.0 } }
tapered { VB2 dummy { 5.5 2.6 0.9 } { 0.4 0.4 0.7 } { 0.0 } { 0.0 } }
tapered { VB3 dummy { 4.6 3.5 0.9 } { 0.4 0.4 0.7 } { 0.0 } { 0.0 } }
tapered { VB4 dummy { 5.5 3.5 0.9 } { 0.4 0.4 0.7 } { 0.0 } { 0.0 } }

solid { LT1 dummy { OB & LB1 } }
solid { LT2 dummy { OB & LB2 } }

solid { O1 SiO2 { OB-LB1-LB2-VB1-VB2-VB3-VB4 } }
solid { CF1 Al { ODL & LB1 } }
solid { CF2 Al { OXR & LB2 } }
solid { LF Al { LT1 + LT2 + VB1 + VB2 + VB3 + VB4 } }

contact { Contact1 electric 1.0 V { CF1 } }
contact { Contact2 electric 0.0 V { CF2 } }
  
```

Figure 10.1: A comparison between the LAYGRID and VGM syntax.

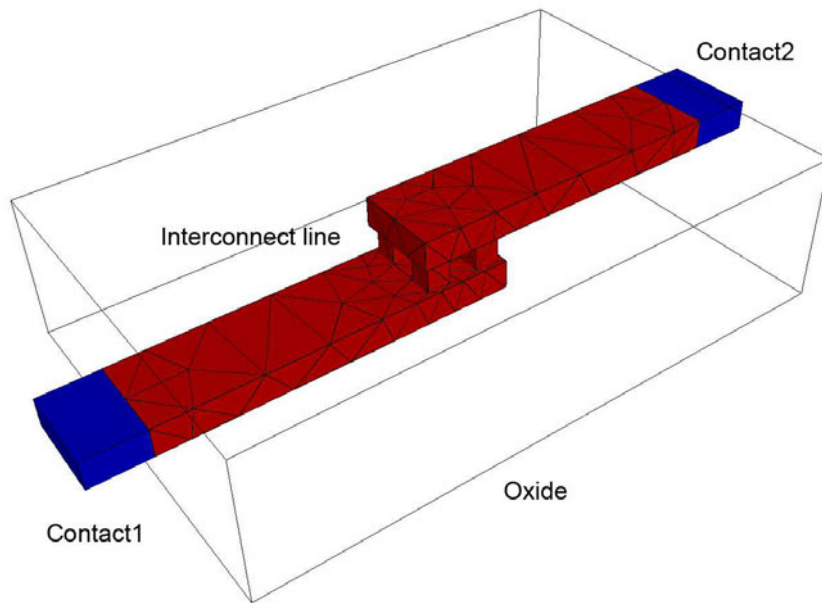


Figure 10.2: Resulting structure from the given input specification.

10.2 Basic Equations and Discretization

Maxwell's equations, given in Section 2.2, are the basic equations for the examples given here. For the stationary electrostatic case, the time derivatives vanish and $\text{curl}(\mathbf{E}) = 0$ is obtained. Such an equation can be expressed by a scalar potential $\mathbf{E} = -\text{grad}(\varphi)$ which is subsequently used to derive the capacitance and resistance analysis [114].

10.2.1 Capacitance Analysis

For the given stationary case with linear dielectric, the ratio of charge Q and voltage U of conductors is constant and is called capacitance $C = \frac{Q}{U}$. Due to the fact that the inner part of the conductors is free of electric field, the charge is distributed only on their surfaces. The charge distribution is derived from \mathbf{D} by means of a surface charge σ as given in Section 2.2 and Equation 2.13. By integration of the charge on the surface of the conductor Γ_i the following is obtained:

$$Q = \int_{\Gamma_i} \sigma dA = \int_{\Gamma_i} \varepsilon \mathbf{E} \cdot d\mathbf{A} \quad (10.1)$$

Then the capacitance C can be derived. Another way can be described by the energy method, where the stored energy is given by $W = \frac{C U^2}{2}$ expressed in local field terms:

$$W = \frac{1}{2} \int_{\Omega} \mathbf{E} \cdot \mathbf{D} d\Omega = \frac{1}{2} \int_{\Omega} \varepsilon \mathbf{E} \cdot \mathbf{E} d\Omega \quad (10.2)$$

Here, the calculation domain covers the complete dielectric part and expands, theoretically, into infinite space. Therefore it is important to emphasize that most of the field energy is contained near the conductors, and so only a small part of the integration domain has to be covered [114].

Both of these calculation mechanisms require the calculation of the electric field \mathbf{E} , where the charge integration requires only the field at the surface of the conductor, which can be expressed by a scalar potential φ . The Maxwell equation $\text{div}(\mathbf{D}) = \rho$, and the fact that an isolator does not carry electric charge ($\rho = 0$), the following equations are used to determine a potential distribution as well as for the extraction of the capacity of a given domain:

$$\text{div}(\varepsilon \text{grad}(\varphi)) = 0 \quad \text{on } \Omega \quad (10.3)$$

$$\mathbf{n} \cdot \varepsilon \text{grad}(\varphi) = 0 \quad \text{on } \Gamma \quad (10.4)$$

$$\varphi = \varphi_k \quad \text{on } \Gamma_{c,k} \quad (10.5)$$

where Ω is a bounded open domain with boundary Γ , and conductor boundaries $\Gamma_{c,k}$, and φ_k represent conductor surface potentials.

As introduced in Section 3.3, the finite element formulation is represented by the following equation system for N -elements:

$$\sum_{j=1}^N \varphi_j \int_{\tau_3^j} \text{grad}(\phi_p(\mathbf{x})) \varepsilon \text{grad}(\phi_q(\mathbf{x})) d\Omega = \sum_{j=1}^N \sigma_j \int_{\partial\tau_3^j} \phi_p \phi_q d\Gamma \quad (10.6)$$

where the following expression for the charge distribution is used $\sigma = \sum_{j=1}^N \phi_{q,j} \sigma_j$. This problem is linear and the global system matrix is simply obtained by assembling Π with the corresponding node index transformation into the global matrix.

10.2.2 Resistance Analysis

The electric resistance can be described by the global law $R = \frac{U}{I}$. The resistance of a conductor can then be calculated by using a potential on the boundary of this conductor and calculating the current by an integration of an area of conductor Γ_i :

$$I = \int_{\Gamma_i} \mathbf{J} \cdot d\mathbf{A} \quad (10.7)$$

or by calculating the electrical power loss:

$$P = \frac{U^2}{R} = \int_{\Omega_i} \mathbf{E} \cdot \mathbf{J} d\Omega \quad (10.8)$$

Similar to the capacitance calculation, the electric field \mathbf{E} has to be calculated. The lack of sources of the electrical current density $\text{div}(\mathbf{J}) = 0$ in the electrostatic system is also retained. Then the following equation for the current density is obtained, where Ω is the inside of a current-carrying conductor:

$$\text{div}(\gamma \text{grad}(\varphi)) = 0 \quad \text{on } \Omega \quad (10.9)$$

$$\mathbf{n} \cdot \gamma \text{grad}(\varphi) = 0 \quad \text{on } \Gamma \quad (10.10)$$

$$\varphi = \varphi_k \quad \text{on } \Gamma_k \quad (10.11)$$

where Ω is a bounded open domain. Γ is identified by the part of the boundary of Ω which is fully covered by isolators and not carrying any contacts, because there is no current flow from the conductor into the isolator $\mathbf{J} \cdot \mathbf{n} = 0$. The contact areas Γ_k normally carry a potential φ_k . The finite element equation system is given for N -elements by:

$$\sum_{j=1}^N \varphi_j \int_{\tau_3^i} \text{grad}(\phi_p(\mathbf{x})) \cdot \gamma \text{grad}(\phi_q(\mathbf{x})) d\Omega = \sum_{j=1}^N \mathbf{J}_j \int_{\partial\tau_3^i} \phi_p \phi_q d\Gamma \quad (10.12)$$

10.2.3 Finite Element Solution Procedure

To obtain the global solution, a procedure where all the local matrices have to be inserted into the global system matrix \mathbf{A} is used, called assembly. Equation 10.6 as well as Equation 10.12 evaluated on all elements of a domain can then be expressed by a matrix notation:

$$\mathbf{A} \mathbf{x} = \mathbf{b} \quad (10.13)$$

where \mathbf{A} is the global system matrix, \mathbf{x} the solution vector, and \mathbf{b} the right-hand side. A transformation of the local indices to global indices is required to obtain consistent system matrix entries.

The right-hand side matrix \mathbf{b} is obtained by contributions of elements on the boundary of the domain, thereby only containing values not equal to zero for boundary contact nodes. A detailed notation of Equation 10.13 is given by [114]:

$$\left[\begin{array}{ccc|ccc} a_{11} & \cdots & a_{1,N_A} & a_{1,N_A+1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{N_A,1} & \cdots & a_{N_A,N_A} & a_{N_A,N_A+1} & \cdots & a_{N_A,N} \\ \hline a_{N_A+1,1} & \cdots & a_{N_A+1,N_A} & a_{N_A+1,N_A+1} & \cdots & a_{N_A+1,N} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{N,1} & \cdots & a_{N,N_A} & a_{N,N_A+1} & \cdots & a_{N,N} \end{array} \right] \cdot \left\{ \begin{array}{c} \underline{x}_1 \\ \vdots \\ \underline{x}_{N_A} \\ \underline{x}_{N_A+1} \\ \vdots \\ \underline{x}_N \end{array} \right\} = \left\{ \begin{array}{c} 0 \\ \vdots \\ 0 \\ \underline{b}_{N_A+1} \\ \vdots \\ \underline{b}_N \end{array} \right\} \quad (10.14)$$

where the upper half corresponds to the inner nodes whereas the lower half can be identified by the boundary nodes. The unknown values are underlined. Expressed as block matrices the equation system reads:

$$\begin{bmatrix} \mathbf{A}_A & \mathbf{A}_B \\ \mathbf{A}_B^T & \mathbf{A}_D \end{bmatrix} \cdot \begin{Bmatrix} \mathbf{x}_A \\ \mathbf{x}_D \end{Bmatrix} = \begin{Bmatrix} 0 \\ \mathbf{b}_D \end{Bmatrix} \quad (10.15)$$

To obtain the unknown values \mathbf{x}_A inside the domain, the equation system has to be solved:

$$\mathbf{A}_A \mathbf{x}_A = -\mathbf{A}_B \mathbf{x}_D \quad (10.16)$$

The electric charge is then obtained by:

$$\mathbf{b}_D = \mathbf{A}_B^T \mathbf{x}_A + \mathbf{A}_D \mathbf{x}_D \quad (10.17)$$

and solving:

$$\mathbf{B} \sigma = \mathbf{b} \quad (10.18)$$

where \mathbf{B} is built up from the local element matrices of the boundary elements.

10.2.4 Transformation into GSSE

The transformation of the given expression is supported by GSSE's finite element components, given in Section 8.3.4. The following set of operations is required to obtain the solution:

1. Computation of the stencil matrices $\Pi(\tau_3^i)$ and matrix \mathbf{b} .
2. Assembly of the global system matrix \mathbf{A} and matrix \mathbf{b} .
3. Solution of the algebraic equation system $\mathbf{A} \mathbf{x} = \mathbf{b}$.

Where the first step is already implemented in the GSSE. The second and third step are given in the following source snippet for two- and three-dimensional simplex and cuboid cell complexes:

```
for (cit = domain.cell_begin(); cit != domain.cell_end(); ++cit)
{
    vertex_on_cell_iterator iter(*cit);
    matrix_type coordinates;
    for (i=0; iter.valid(); ++iter, ++i)
    {
        coordinates(i) = domain.get_coordinate(*iter);
    }
    matrix_type local_m = coord_transformation(coordinates);

    matrix_type stencil (local_m);

    assemble_stencil(stencil, matrix_interface);
}

matrix_interface.solve();
```

Finite element assembly procedure

The final line invokes the solver using the generic solver interface. Because of the library-centric application design approach of the GSSE and the derived applications, the `matrix_type` can be implemented by any matrix data structures, e.g., the high performance Blitz++ [150] matrices, or the C++ `valarray`. A performance comparison is given in Section 8.4.

10.3 Applications

The following examples are a selection of examples with short comments providing a brief description.

10.3.1 Capacitance Analysis

This example not only provides a capacitance extraction for the structure illustrated in Figure 10.3 but also shows how different mesh generation techniques influence the whole simulation process. A LAYGRID mesh and the VGM mesh were used as a calculation base. It is important to highlight that the given structure complicates the original LAYGRID approach, due to the offset of the second layer, depicted in Figure 10.4 on the left side. By a prismatic mesh generation, this issue can result in a mesh with more than two to three orders of magnitude difference in the number of mesh elements. Therefore the example was appropriately adapted to enable the comparison with the LAYGRID mesh approach.

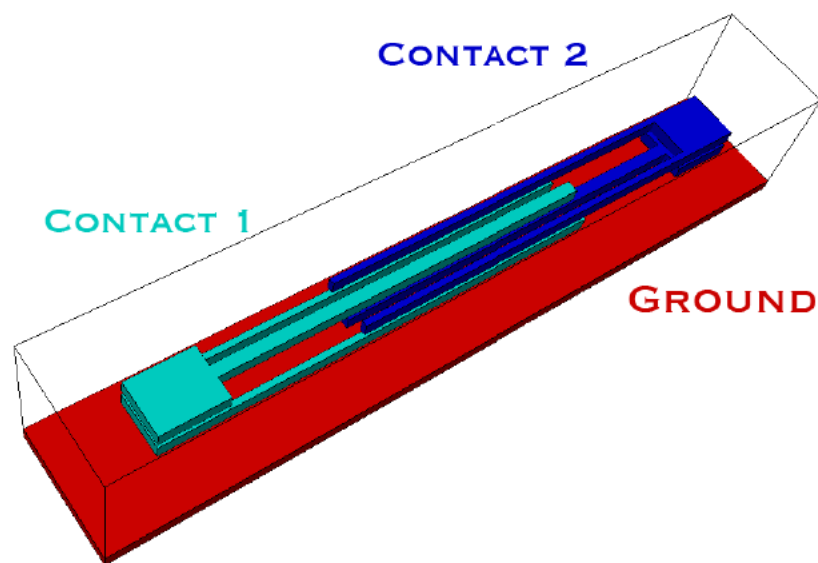


Figure 10.3: A structure suitable for capacitance simulation, created by VGM.

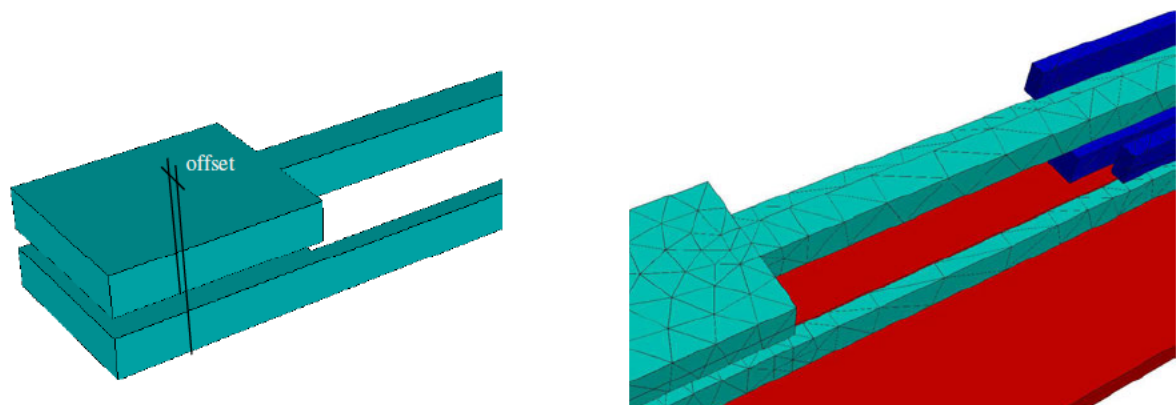


Figure 10.4: The offset of the second layer of contact 1. Right: the corresponding mesh.

The potential distribution for the two contacts is given in Figure 10.5. A smooth potential is required to extract the capacity correctly. Therefore care has to be taken in regard to spatial discretization of the structure, as shown in Figure 10.4 on the right side.

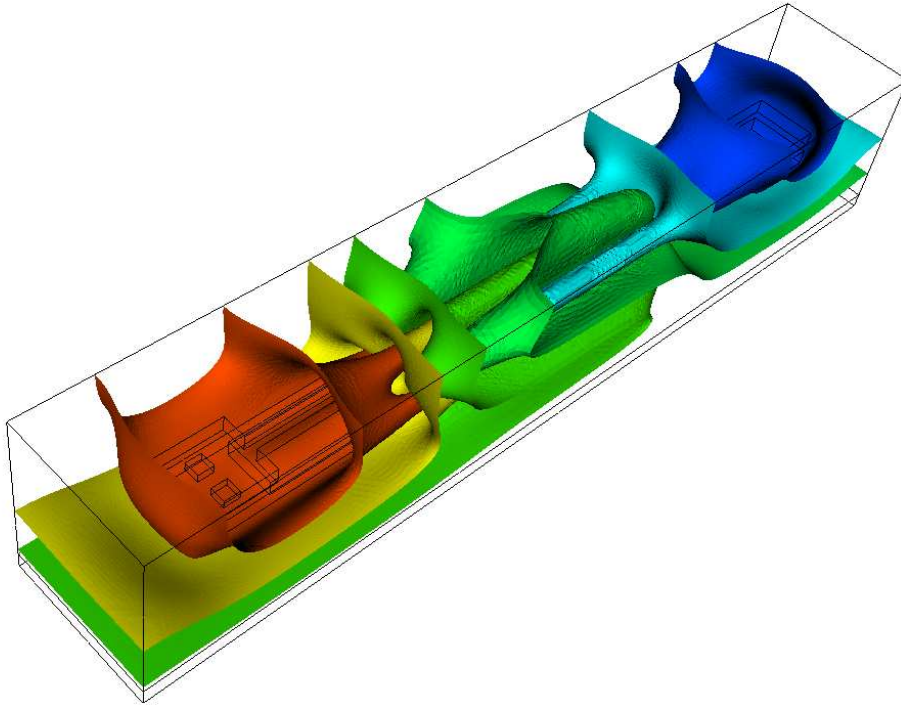


Figure 10.5: Illustration of isosurfaces of the potential distribution.

A capacitance extraction analysis is given in the following table, where the left column always represents the LAYGRID mesh and the right column the VGM mesh. The refinement column states the required overall mesh refinement steps to obtain the targeted accuracy. The x marks examples, where no meshes could be generated.

Refinement	Capacity [pF]		Number of points		Calculation time [s]		Relative error	
0	5.58	6.51	$4 \cdot 10^4$	$1 \cdot 10^4$	7.70	1.10	5.08	22.59
2	5.35	5.44	$2 \cdot 10^6$	$9 \cdot 10^5$	2282	326	0.07	2.44
4	x	5.31	x	$5 \cdot 10^7$	x	4920	x	0.0

10.3.2 Resistance Analysis for Cu-DD Architecture

This example deals with the resistance analysis of different types of via structures related to their cross-section in a copper-dual-damascene architecture, illustrated in Figure 10.6. It is shown how the thickness of the barrier layer influences the structure's resistance [114].

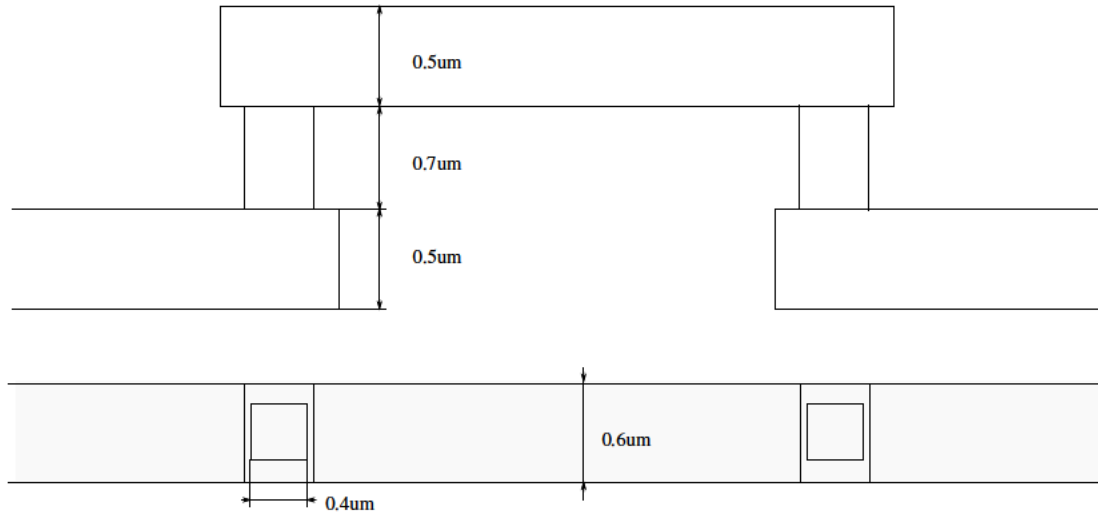


Figure 10.6: Interconnect structure for resistance analysis.

Figure 10.7 illustrates the different material types of the structure under investigation as well as the TiN barrier layer.

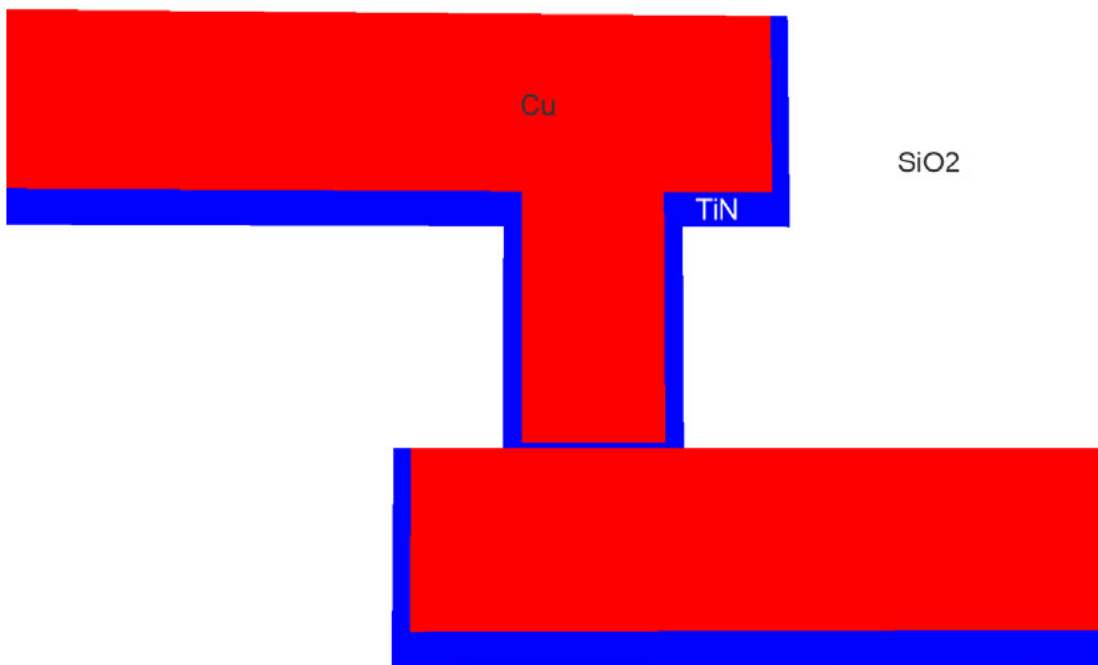


Figure 10.7: Interconnect structure for resistance analysis.

To give a short overview of the new syntax and mannerism of VGM a comparison of a LAYGRID file and a VGM input file (Figure 10.8) is presented.

```

lengthunit { 1.0 um }
mask { MASK0
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
mask { MASK1
  rectangle { CBLI TiN { 1.995 -1.005 } { 0.61 5.01 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
mask { MASK2
  rectangle { BLI Cu { 2.0 -1.0 } { 0.6 5.0 } }
  rectangle { CBLI TiN { 1.995 -1.005 } { 0.61 5.01 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } }
  rectangle { CT1 Cu { 2.0 4.0 } { 0.6 1.0 } } }
mask { MASK3
  circle { VIA1 Cu { 2.3 4.5 } { 0.05 } }
  circle { CVIA1 Cu { 2.3 4.5 } { 0.0525 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
mask { MASK4
  rectangle { CTLI TiN { 1.995 3.995 } { 0.61 6.01 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
mask { MASK5
  rectangle { TLI Cu { 2.0 4.0 } { 0.6 6.0 } }
  rectangle { CTLI TiN { 1.995 3.995 } { 0.61 6.01 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } }
  rectangle { CT2 Cu { 2.0 10.0 } { 0.6 1.0 } } }
mask { MASK6
  rectangle { CTLI TiN { 1.995 3.995 } { 0.61 6.01 } }
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
mask { MASK7
  rectangle { OX SiO2 { -4 0 } { 14 10 } } }
contact { CONT1 cap vol { CT1 } }
contact { CONT3 cap vol { CT2 } }
layerstructure {
  origin { 0 0 0 }
  plane { ----- }
  layer { MASK0 1 }
  plane { ----- }
  layer { MASK1 0.005 }
  plane { ----- }
  layer { MASK2 0.5 }
  plane { ----- }
  layer { MASK3 0.695 }
  plane { ----- }
  layer { MASK4 0.005 }
  plane { ----- }
  layer { MASK5 0.5 }
  plane { ----- }
  layer { MASK6 0.005 }
  plane { ----- }
  layer { MASK7 1 }
  plane { ----- }
}
tapered {TL Cu {2.0 4.0 1.2 }{0.6 7.0 0.5}{-0.01}{0.0}}
pyramid {VIA1 Cu {2.3 4.5 0.503}{0.05 0.9} {26}{1.0}{0.0}}
tapered {BL Cu {2.0 -1.0 0.0 }{0.6 6.0 0.5}{-0.01}{0.0}}
tapered {CTL TiN {1.995 3.995 1.195}{0.61 7.01 0.51}{-0.01}{0.0}}
pyramid {CVIA1 TiN {2.3 4.5 0.45 }{0.0525 1 }{26}{1.0}{0.0}}
tapered {CBL TiN {1.995 -1.005 -0.005}{0.61 6.01 0.505}{-0.01}{0.0}}

line {OX1 SiO2 {-4 0 -1 } { 14 10 5 }}
line {OXC1 SiO2 {0 10 -1 } { 10 10 5 }}
line {OXC2 SiO2 {0 -10 -1 } { 10 10 5 }}

solid {LCM Cu {VIA1 + TL}}
solid {LF Cu {LCM & LCM - BL}}
solid {CCM TiN {CTL + CVIA1}}
solid {C1TM TiN {CCM - LCM - BL}}
solid {CB2TM TiN {CBL - BL}}
solid {OXCm SiO2 {OX1 - CCM - CBL - BL}}
solid {CVTL TiN {OX1 & C1TM}}
solid {FCVBL TiN {OX1 & CB2TM}}
solid {FBL Cu {OX1 & BL }}
solid {FTL Cu {OX1 & LF }}
solid {CT1 Cu {OXC1 & LF }}
solid {CT2 Cu {OXC2 & BL }}

contact {CONTACT1 electric 1.0 V { CT1 }}
contact {CONTACT2 electric 0.0 V { CT2 }}

```

Figure 10.8: A comparison between the LAYGRID and VGM syntax.

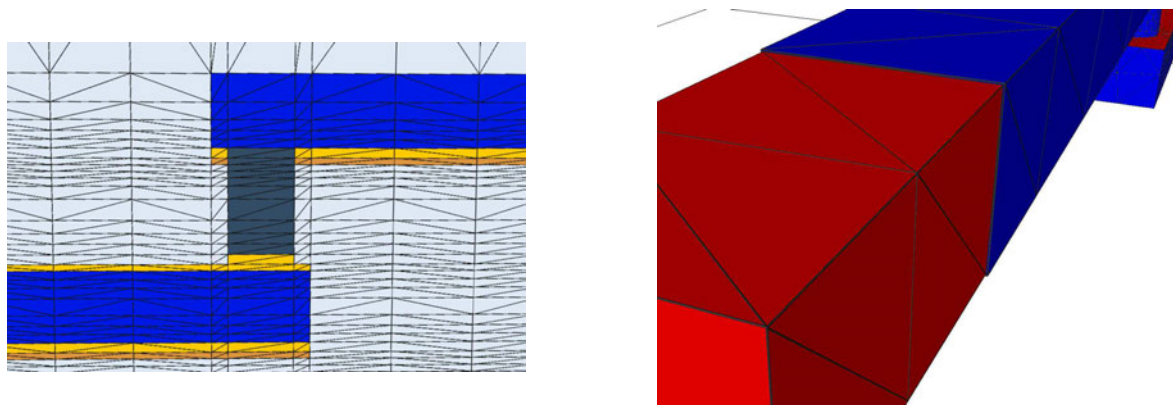


Figure 10.9: Comparison of the structures created and the corresponding meshes for LAYGRID (left) and VGM (right). With the VGM approach different ratios between the interconnect line (red) and the covering layer (blue) can be easily modeled. Here a ratio between the thickness of the interconnect line and the covering layer is given by 1/10000. As can be seen, the thin layer does not impose additional points for the interconnect line.

An analysis related to the influence of the spatial expansion of the via is given in Figure 10.10.

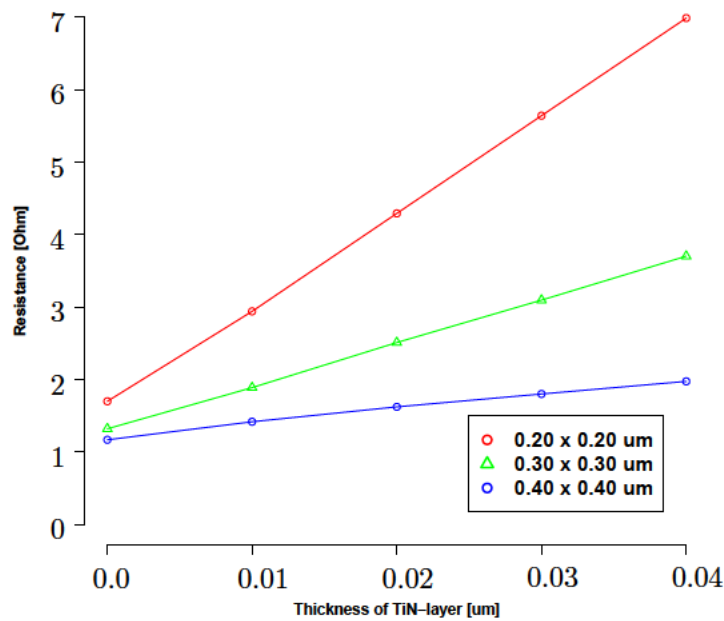


Figure 10.10: Resistance progression related to the spatial expansion of the via part.

The influence of the barrier thickness which respect to the overall resistance of the structure can be clearly observed in small cross-sections of the via.

10.4 Performance Analysis

An analysis and comparison taking into account the performance of a whole-application are presented. It should be noted that for high precision simulations it is essential to model the simulation domain as exactly as possible. The accuracy and efficiency of a finite element or a finite volume simulation strongly depends on the quality of the tessellation of the domain. Figure 10.11 presents the structure under investigation with a coarse mesh for the following performance analysis. In order to obtain sufficiently accurate results, the mesh size typically has to be in the order of 10^4 to some 10^5 nodes.

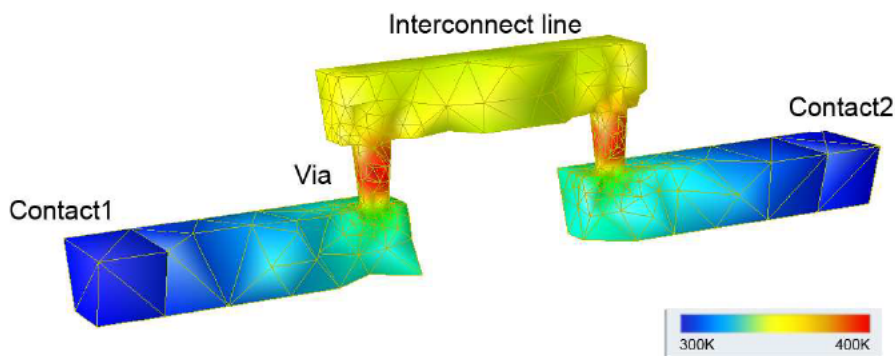


Figure 10.11: Temperature distribution due to self-heating in a tapered interconnect line with cylindrical vias.

Based on these observations of restrictions due to limited resources, the influence of a problem's size on the overall performance is also illustrated. The given test structure with different levels of mesh refinement is used for analysis. By resolving a three-dimensional simulation domain, the large number of points easily leads to severe problems caused by memory bandwidth restrictions (Figure 10.12).

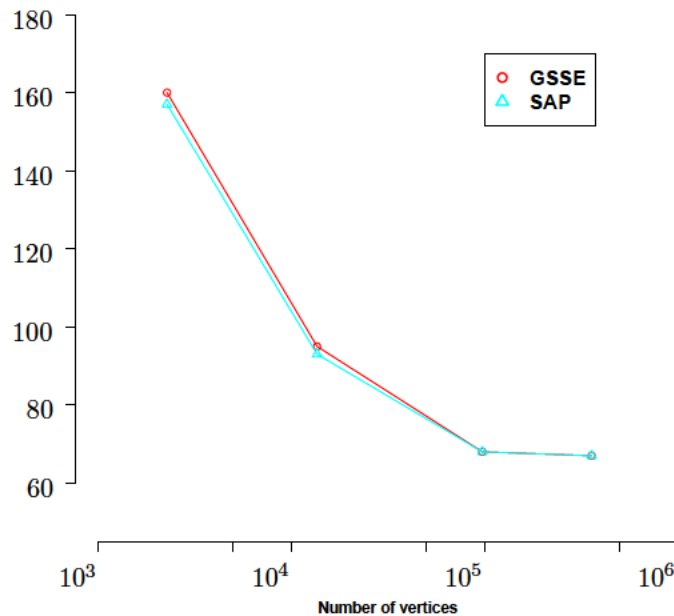


Figure 10.12: Comparison of the finite element assembly times in equations per millisecond, GSSE and SAP.

As can be seen, the high performance manually tuned SAP shows the same run-time compared to the GSSE code module.

10.5 Application Design Concepts

The main difference between the imperative SAP and the library-centric application design approach followed by the GSSE is the use of several orthogonal modules, as given in Section 8.3.4 for the assembly procedure. The emphasized expressiveness of the source code is not obscured by the overall high performance. To give an approximate measure for the effort resulting from the chosen application design, the necessary code parts of GSSE and SAP are analyzed by the required number of source lines [93], where all components of the GSSE require 7400 lines of source code:

Application part	GSSE	SAP
input/output	10	5481
segment analysis	558	3096
assembly	13	5866
solver	23	4523
total number:	604	18966

As can be seen, the development effort is reduced dramatically at an equal run-time performance even in the absence of manually tuned code parts. Instead, the compiler and various high performance libraries are used.

To emphasize not only the high performance library-centric application design proposed here, but also to demonstrate the difference of programming style with respect to extensibility, maintenance, and reusability, a comparison of the finite element algorithms by the GSSE

```
void assemble_stencil( .. ) {
    for(int i=0; i<index_list.size(); ++i) {
        long line = index_list(i);

        for(int j=0; j<index_list.size(); ++j) {
            long column = index_list(0, j);
            mat_add(line, column, stencil(i, j));
        }
        int rhs_pos = index_list.size();
        rhs_add(line, stencil(i, rhs_pos));
    }
}
```

and the corresponding matrix assembly section from SAP is given. The necessary combination of memory management and program flow, e.g., `assemble_reset_mat()` can be clearly seen. This part of code also lacks reusability due to the pre-conditions required by the algorithm.

```
static void assemble_elem(...) {
    Finite *fin;
    int k, therm_idx, j;
    double *b;
    Subproblem *sp;
    //..
    for(k=0; k<sp->region_n; k++) {
        Region *r;
        Gridelements *g;
        ModelInfo *m;
        AssElem ae;

        if(sp->region_status[k]!=1) continue;
        r=fin->region+k;
        g=&r->grid;
        m=t->modelInfo+k;
        ae.region=k;
        ae.node_map=sp->node_map[k];
        ae.model=m;

        for(j=0; j<g->elem_n; j++) {
            assemble_reset_mat(t);
            ae.elem=GridelementsGet(g, j);
            get_elem_nodes(fin, &ae);
            if(m->ass_S) {
                m->ass_S(t, &ae);
                matrix_entry(t, S, &ae);
            }
            if(m->ass_b) {
                m->ass_b(t, &ae);
                b_entry(t, S, &ae);
            }
        }
    }
}
```

Chapter 11

Summary and Outlook

I have presented theoretical, practical, and applied concepts related to the field of scientific computing and, therefore, introduced possible means to enable the reuse of data structures and algorithms. Customized algorithms can be implemented easily where they prove most valuable with the corresponding programming paradigm. By dimensional independent programming, not only is all relevant physics included, but also the demand on computer resources is kept to a minimum. These concepts have been applied and it was proven that this approach leads to a drastic reduction of the amount of newly developed modules and application parts. The following table briefly summarizes the impact of the library-centric application design approach by estimating the total number of lines of code for all developed applications at our institute:

1980	100.000 lines of code	imperative	Fortran
1990	300.000 lines of code	imperative	C, Fortran
2000	600.000 lines of code	imperative, OO	C, C++
2006	20.000 lines of code	OO, GP, FP, MP	C++

Application maintenance is easily achieved by these few software components, whereby most of the application code is reused by other applications, resulting in a great extensibility and stability.

The concepts collected, refined, and formalized here are a solid foundation for future work. The formalization of guidelines allows a clean decomposition of applications into general-purpose components. Following precisely specified interfaces interoperability of the original modules as well as future developments adhering to the interfaces is guaranteed. Therefore, the basic building blocks supplied here can serve as a crucial part for the rapid development of the state-of-the-art in scientific computing. The easy extension of these building blocks to provide additional functionality of complex topics such as geometric algebra calculus should also be counted among the major strengths of the presented environment. Besides the development of extensions of the already extensive GSSE the assembly of generic modules containing algorithms and data structures to form applications addressing a large variety of different problems will be the focus of future work.

Appendix A

Common Mathematical Terms

Basic mathematical terms are reviewed. The readers' familiarity with sets and subsets [2] is required in the following. All sets which contain only one element (singleton) can be used to describe a vertex.

Definition 44 (Relation) *A relation R between sets A and B is a collection of ordered pairs (a, b) such that $a \in A$ and $b \in B$, if $(a, b) \in R$.*

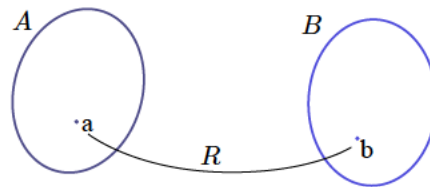


Figure A.1: A relation R of a set A and a set B .

Definition 45 (Function) *A function or mapping f is a rule that assigns to each element $a \in A$ exactly one element $b \in B$. So f maps a into b . This is denoted by $f(a) = b$.*

A permutation of a finite set is usually given by its action on the elements of the set. An example is given by the basic set $\{1, 2, 3, 4, 5\}$ with a transformation into $\{5, 4, 3, 2, 1\}$ where the notation states that the permutation maps $1 \rightarrow 5$, $2 \rightarrow 4$ and so on. Thereby all permutations of a finite set can be placed into two sets.

Definition 46 (Parity) *A permutation of a finite set can be expressed as either an even or an odd number of transpositions, but not both. In the former case, the permutation is called even, in the latter it is odd.*

Definition 47 (Image) *Let A and B be sets, f be the function $f : A \rightarrow B$, and $a \in A$. Then the image of a under f , denoted by $\text{im } f$, is the unique member $b \in B$ that f associates with a .*

$$f(A) = \{b = f(a) \mid a \in A\} \tag{A.1}$$

Definition 48 (Composite Function) *If f and g are functions with $f : A \rightarrow B$ and $g : B \rightarrow C$, then there is a natural function mapping A into C , the composite function, consisting of f followed by g . It can be written $g(f(a)) = c$ the composition of functions is denoted by $g \circ f$.*

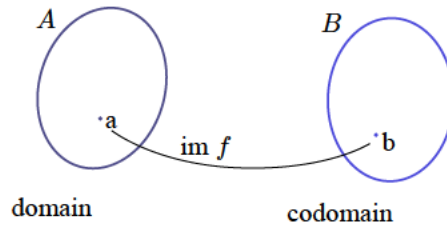


Figure A.2: The image of a under f .

Definition 49 (Preimage) The preimage (fiber, inverse image) of a set $B \subseteq Y$ under f is the subset of X defined by

$$f^{-1}(B) = \{x \in X | f(x) \in B\} \quad (\text{A.2})$$

The preimage of a singleton, $f^{-1}(y)$, is a fiber or a level set. A fiber can also be the empty set \emptyset . The union set of all fibers of a function is called the *total space*, E .

Definition 50 (Group) A group $(G, *)$ is a set G with a binary operation $* : G \times G \rightarrow G$, such that the following axioms are satisfied:

- $*$ is associative
- $\exists e \in G$ such that $e * x = x * e$ for all $x \in G$. The element e is an identity element for $*$ on G .
- $\forall a \in G, \exists a' \in G$, such that $a' * a = a * a' = e$. The element a' is an inverse of a with respect to the operation $*$.

Definition 51 (Abelian) A group G is Abelian, if its binary operation $*$ is commutative.

Definition 52 (Subgroup) A subset $H \subseteq G$ of a group $(G, *)$ is a subgroup of G if H is a group and closed under the operation $*$. The subgroup consists of the identity element of G , whereas $\{e\}$ is the trivial subgroup of G . All other subgroups are nontrivial.

Definition 53 (Injective) A mapping $f : A \rightarrow B$ is called injective if there is at most one value $b = f(a) \in B$ for each element $a \in D$. This means that every preimage in B has an image in A but there may be elements in A which are not obtained as images.

Definition 54 (Surjective) A mapping $f : A \rightarrow B$ is called surjective if $b = f(a)$ has at least one solution $b \in B$ for every $a \in A$. Several identical images in A may have different preimages in B .

Definition 55 (Bijective) A mapping $f : A \rightarrow B$ that is both injective and surjective is called bijective. It maps every preimage from B to a distinct image in A . There are no elements in A that cannot be obtained as images.

Definition 56 (Vector Space) A vector space \mathcal{V} over a field \mathbb{F} is a set associated with two binary operations. The elements of \mathcal{V} are called vectors, while the elements of \mathbb{F} are called scalars. The binary operations are addition

$$+ : \mathcal{V} \times \mathcal{V} \rightarrow \mathcal{V} \quad (\text{A.3})$$

which is commutative as well as associative and which possesses a neutral as well as an inverse element, and scalar multiplication

$$\cdot : \mathbb{F} \times \mathcal{V} \rightarrow \mathcal{V} \quad (\text{A.4})$$

which is distributive with addition. \mathcal{V} is closed under its two binary operations.

Definition 57 (Linear Mapping) A mapping $f : \mathcal{U} \rightarrow \mathcal{V}$ is called linear, if the following relation holds

$$f(\lambda u + \mu v) = \lambda f(u) + \mu f(v) \quad (\text{A.5})$$

A mapping

$$g : L \times M \rightarrow N \quad (\text{A.6})$$

that is linear in both of its arguments, as in the following relations

$$\varphi(\lambda u + \mu v, w) = \lambda \varphi(u, w) + \mu \varphi(v, w) \quad (\text{A.7})$$

$$\varphi(u, \lambda w + \mu x) = \lambda \varphi(u, w) + \mu \varphi(u, x) \quad (\text{A.8})$$

is called bilinear.

The extension of this notion to more than two arguments is called a multi-linear mapping.

Definition 58 (Homomorphism) A map f of a group G into a group G' is a homomorphism if $f(ab) = f(a)f(b)$, $\forall a, b \in G$. For any groups G and G' , there is always at least one homomorphism $f : G \rightarrow G'$, namely the trivial homomorphism defined by $f(g) = e'$, $\forall g \in G$, where e' is the identity element in G' .

Definition 59 (Kernel) Let $f : G \rightarrow G'$ be a homomorphism. The subgroup $f^{-1}(\{e'\}) \subseteq G$, consisting of all elements of G mapped by f into the identity e' of G' , is the kernel of f , denoted by $\ker f$.

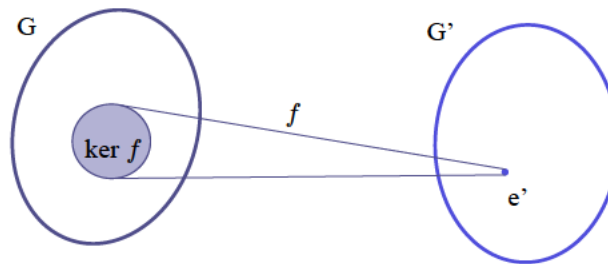


Figure A.3: The kernel of f .

Appendix B

STL Iterator Analysis

The topological specification mechanisms of Section 6.1 and the separation properties of the fiber bundle concept are used to analyze the C++ STL iterator concept in detail. With the advent of the C++ STL the separation of access to data structures and algorithms by means of iterators has become one of the key elements for the generic programming paradigm. Thereby the implementation complexity of algorithms and data structures is significantly reduced. The value access requirements in existing iterator categories are given by:

output iterator	*iter = a
input iterator	*iter is convertible to T
forward iterator	*iter is T&
random access iterator	iter[n] is convertible to T

Here, `iter` represents the iterator and `T` the corresponding data type of the evaluated iterator. But unfortunately, it combines traversal and data access. An example of this problem of this iterator concept is, e.g., the well-known example of the `std::vector<bool>` which can actually be modeled by a random access container [110] due to the container data structure. But the return type is not `bool&` and can thereby only be modeled by the requirement of input and output iterators. An alternative approach was suggested [110] by separating traversal and access, where the access hierarchy is described by:

- Readable iterator
- Writable iterator
- Swappable iterator
- Lvalue iterator

and by the traversal hierarchy:

- Incrementable iterator
- Single Pass iterator
- Forward Traversal iterator
- Bidirectional Traversal iterator
- Random Access Traversal iterator

But up to now this concept is not widely used. The proposed new iterator concept [156] and another version [157] with a different focus on these problems, were accepted into the TR1¹ of C++. Additionally, the backward and forward compatibility of the new iterator categories are a major problem [158] and are not included in the new C++0x standard [159].

¹Library Technical Report 1 (TR1) is a draft document specifying additions to the C++ Standard Library

This concept is briefly illustrated in the following source snippet:

```
vector<bool>          container;
vector<bool>::iterator it;
property_map         pm(container);

it = container.begin();
++it;                //iteration
bool value = pm(*it); //data access
```

Separated iteration and data access

As explained in Section 6.1, there is a formal and distinguishable definition possible for all of these data structural properties. Based on this formal classification the combination of traversal and data access is analyzed in more detail. The following list gives an overview of the basic iterator traits [17] for the basic STL iterators:

iterator category	specification	property
input/output		data property
forward	local(2)	topological property
bidirectional	local(3)	topological property
random access	global	topological property

Problems are encountered if the new iterator categories [159] are integrated into the topological specification.

iterator category	specification	property
incrementable	local(2)	topological property
single pass		data property
forward	local(2)	data and topological property

The combinatorial property of the underlying space of these three categories is the same: a 0-cell complex with a local topological structure such as:

data structure	dimension	cell type	complex topology
single linked list / stream	0	simplex	local(2)

Only the incrementable property can be described by a topological property, whereas the other two categories are data-dependent. The former iterator properties [17] only use two different categories which specify the behavior of data, namely the input and output properties.

Appendix C

Cell Properties of Discretization Schemes

As introduced in Section 3, the finite volume and finite element methods use different cell types for their approximations. Finite volumes write the balance equations in terms of global quantities, whereas finite elements, making use of spread cells with spread boundaries, is forced to use field functions defined locally over the whole domain. Despite this difference, neither finite volume nor finite element discretize the operator representing the local version of the balance equation. Instead they both resort to a global version since a topological equation directly applies to regions with finite extension. A detailed geometrical analysis [33] based on the identification of the weighting functions was given.

The weighted residual formulation of the finite element method starts from the continuous formulation of a weighted domain:

$$\int_{w_{p+1}} d\omega^p = \int_{\partial w_{p+1}} \omega^p \quad (\text{C.1})$$

The relationship between the two concepts, the continuous concept of p -differential forms (Section 2.7) and the corresponding weighted domains, and the discrete concept related to chains and cochains (Section 2.4) can be used to algebraically handle the cells from a cell complex and the discrete projection of a physical field. The weighted domain and the p -form are then directly related to the p -chains and p -cochains. When d is used as a divergence operator, the following expression is obtained in the discrete case:

$$\int_{\tau_3} w \operatorname{div}(\mathbf{D}) = \int_{\partial\tau_3} w \mathbf{D} - \int_{\tau_3} \operatorname{grad}(w) \cdot \mathbf{D} \quad (\text{C.2})$$

where the 3-cell τ_3 can be taken as the support of the weight function w . With the expression for the boundary of a weighted three-dimensional geometric object, the following formal definition can be obtained:

$$\int_{\partial(w \tau_3)} \mathbf{D} := \int_{\partial\tau_3} w \mathbf{D} - \int_{\tau_3} \operatorname{grad}(w) \cdot \mathbf{D} \quad (\text{C.3})$$

where the $w \tau_3$ represents a weighted 3-cell. This "boundary" includes actually an integral of the whole 3-cell τ_3 , and not only on $\partial\tau_3$, except in the particular case of a constant weight function on its supporting cell. A corresponding interpretation of the weight function w as a continuous counterpart of a chain yields a concept called spread cell, compared to a "crisp" cell, considered so far by a constant weight function. If a weight function is constant on a cell and zero outside, the second term of the equation vanishes and the finite element method corresponds with a finite volume method. Due to the fact that these spread cells overlap, this assembly of cells does not model the concepts introduced for a consistent cell complex.

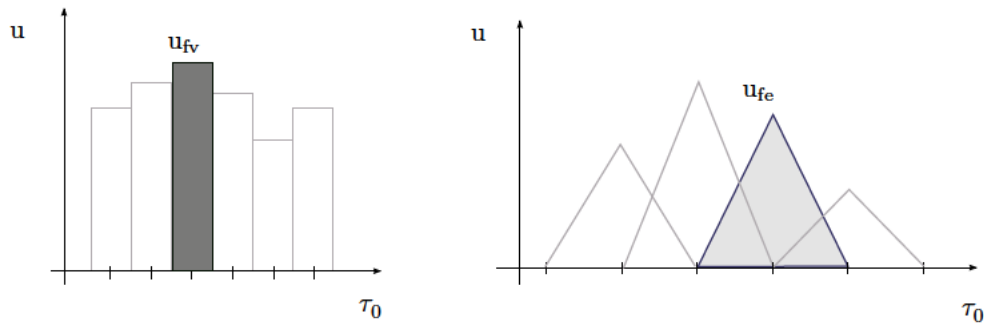


Figure C.1: Geometrical interpretation of the basis functions for the (left) finite volume method and (right) for the finite element method. The finite volume method describes consistent crisp cells which can be interpreted as a cell complex, whereas the finite element method uses spread cells which do not conform with the properties of a consistent cell complex.

Figure C.1 depicts the different approaches of the finite volume and finite element method. The x-axis is divided into one-dimensional cells, marked with vertical lines. The finite volume method uses crisp cells which are aligned accordingly with the dual complexes of the given cells of the underlying cell complex. The finite element method uses spread cells which do not build a consistent cell complex. In this example, each cell, depicted by the basis function, is spread over two cells. From the point of view of the dual complex, which is not used by the finite element method, the cells extend beyond the area of a dual cell. The important part of the integration by parts in the FE method can be thereby explained with the necessity to operate with the boundary of the FEM spread cells in order to express topological equations.

Bibliography

- [1] W. Benger, “Visualization of General Relativistic Tensor Fields via a Fiber Bundle Data Model,” Dissertation, Freie Universität Berlin, 2004.
- [2] K. Jänich, *Topologie*. Heidelberg: Springer, 2001.
- [3] T. Archer and A. Whitechapel, *Inside C#*, 2nd ed. Redmond, WA, USA: Microsoft Press, 2002.
- [4] W. Bright, “The D Programming Language,” *Dr. Dobbs’s J.*, vol. 27, no. 2, pp. 36–40, 2002.
- [5] I. Kazi, H. Chen, B. Stanley, and D. Lilja, “Techniques for Obtaining High Performance in Java Programs,” *ACM Computing Surveys*, vol. 32, no. 3, pp. 213–240, 2000.
- [6] J. Gosling, B. Joy, G. Steele, and G. Bracha, *The Java Language Specification, Third Edition*. Addison Wesley, 2005.
- [7] S. P. Jones and J. H. (Eds.), “Haskell 98: A Non-strict, Purely Functional Language,” Tech. Rep., February 1999. [Online]. Available: <http://haskell.org/onlinereport/>
- [8] B. Stroustrup, *The C++ Programming Language*. New York: Addison Wesley, 1995.
- [9] D. Gregor, J. Järvi, J. Siek, B. Stroustrup, and G. R. A. A. Lumsdaine, “Concepts: Linguistic Support for Generic Programming in C++,” *SIGPLAN Not.*, vol. 41, no. 10, pp. 291–310, 2006.
- [10] R. Minixhofer, “Integrating Technology Simulation into the Semiconductor Manufacturing Environment,” Dissertation, Technische Universität Wien, 2006.
- [11] S. Halama, C. Pichler, G. Rieger, G. Schrom, T. Simlinger, and S. Selberherr, “VISTA — User Interface, Task Level, and Tool Integration,” *IEEE Trans. on Techn. Comp. Aided Design*, vol. 14, no. 10, pp. 1208–1222, 1995.
- [12] T. Binder, A. Hössinger, and S. Selberherr, “Rigorous Integration of Semiconductor Process and Device Simulators,” *IEEE Trans. Comp.-Aided Design of Intl. Circ. and Systems*, vol. 22, no. 9, pp. 1204–1214, 2003.
- [13] I μ E, *MINIMOS-NT 2.1 User’s Guide*, Institut für Mikroelektronik, Technische Universität Wien, Austria, 2004, <http://www.iue.tuwien.ac.at/software/minimos-nt>.
- [14] S. Wagner, T. Grasser, and S. Selberherr, “Performance Evaluation of Linear Solvers Employed for Semiconductor Device Simulation,” in *Proc. Conf. on Sim. of Semiconductor Processes and Devices*, 2004, pp. 351–354.
- [15] M. Heroux, R. Bartlett, V. H. R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams, “An Overview of Trilinos,” Sandia National Laboratories, Tech. Rep. SAND2003-2927, 2003.

- [16] D. R. Musser and A. A. Stepanov, “Generic Programming,” in *Proc. of the ISSAC’88 on Symb. and Alg. Comp.* London, UK: Springer, 1988, pp. 13–25.
- [17] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library.* Boston, MA, USA: Addison-Wesley, 1998.
- [18] G. Berti, “Generic Software Components for Scientific Computing,” Dissertation, Technische Universität Cottbus, 2000.
- [19] A. J. Zomorodian, “Topology for Computing,” in *Cambridge Monographs on Applied and Computational Mathematics*, 2005.
- [20] D. Butler and M. Pendley, “A Visualization Model Based on the Mathematics of Fiber Bundles,” *Computers in Physics*, vol. 3, no. 5, pp. 45–51, September/October 1989.
- [21] D. M. Butler and S. Bryson, “Vector Bundle Classes From Powerful Tool for Scientific Visualization,” *Computers in Physics*, vol. 6, pp. 576–584, November/December 1992.
- [22] A. Hatcher, *Algebraic Topology.* Cambridge University Press, 2002.
- [23] G. Deschamps, “Electromagnetics and Differential Forms,” *Proc. of the IEEE*, vol. 69, no. 6, pp. 676–696, 1981.
- [24] E. Tonti, “Finite Formulation of the Electromagnetic Field,” in *Geometric Methods in Computational Electromagnetics, PIER 32*, F. L. Teixeira, Ed. Cambridge, Mass.: EMW Publishing, 2001, pp. 1–44.
- [25] P. Bochev and M. Hyman, “Principles of Compatible Discretizations,” in *Proc. of IMA Hot Topics Workshop on Compatible Discretizations*, vol. IMA 142. Springer, 2006, pp. 89–120.
- [26] P. A. Markowich, C. Ringhofer, and C. Schmeiser, *Semiconductor Equations.* Wien-New York: Springer, 1990.
- [27] T. Barth and M. Oehlberger, “Finite Volume Methods: Foundation and Analysis,” 2004. [Online]. Available: citeseer.ist.psu.edu/barth04finite.html
- [28] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method.* Berkshire, England: McGraw-Hill, 1987.
- [29] C. Johnson, *Numerical Solutions of Partial Differential Equations by the Finite Element Method.* Cambridge, UK: Cambridge University Press, 1987.
- [30] K. S. Yee, “Numerical Solution of Initial Boundary Value Problems involving Maxwell’s Equations in Isotropic Media,” *IEEE Trans. Antennas and Propagation*, vol. 14, no. 1, pp. 302–307, 1966.
- [31] E. Tonti, “The Reason for Analogies between Physical Theories,” *Appl. Math. Modelling*, vol. 1, no. 1, pp. 37–50, 1976/77.
- [32] ———, “On the Geometrical Structure of Electromagnetism,” in *Gravitation, Electromagnetism and Geometrical Structures.* Bologna: Pitagora, 1996, pp. 281–308.
- [33] C. Mattiussi, “An Analysis of Finite Volume, Finite Element, and Finite Difference Methods using some Concepts From Algebraic Topology,” *J. Comput. Phys.*, vol. 133, no. 2, pp. 289–309, 1997.
- [34] ———, “The Finite Volume, Finite Element, and Finite Difference Methods as Numerical Methods for Physical Field Problems,” *Advances in Imaging and Electron Physics*, vol. 113, pp. 1–146, 2000.

- [35] ———, “The Geometry of Time-Stepping,” in *Geometric Methods in Computational Electromagnetics*, PIER 32, F. L. Teixeira, Ed. Cambridge, Mass.: EMW Publishing, 2001, pp. 123–149.
- [36] ———, “A Reference Discretization Strategy for the Numerical Solution of Physical Field Problems,” *Advances in Imaging and Electron Physics*, vol. 121, pp. 143–279, 2002.
- [37] J. M. Hyman and M. J. Shashkov, “Mimetic Discretizations for Maxwell’s Equations and Equations of Magnetic Diffusion,” in *Proc. of the Fourth Int. Conf. on Mathematical and Numerical Aspects of Wave Propagation, Golden, Colorado, June 1-5, 1998*, J. A. DeSanto, Ed. SIAM, 1998, pp. 561–563.
- [38] C. Ghezzi, M. Jazayeri, and D. Mandrioli, *Fundamentals of Software Engineering*. Upper Saddle River, NJ, USA: Prentice-Hall, 1991.
- [39] K. Bruce, L. Cardelli, G. Castagna, G. T. Leavens, and B. Pierce, “On Binary Methods,” *Theor. Pract. Object Syst.*, vol. 1, no. 3, pp. 221–242, 1995.
- [40] J. G. Siek and A. Lumsdaine, “The Matrix Template Library: A Unifying Framework for Numerical Linear Algebra,” in *ECOOP Workshop*, 1998, pp. 466–467. [Online]. Available: citeseer.ist.psu.edu/siek98matrix.html
- [41] J. Siek and A. Lumsdaine, “Mayfly: A Pattern for Lightweight Generic Interfaces,” July 1999. [Online]. Available: citeseer.ist.psu.edu/siek99mayfly.html
- [42] J. G. Siek and A. Lumsdaine, “The Generic Graph Component Library,” *Dr. Dobb’s J. of Software Tools*, September 2000.
- [43] J. Siek, L.-Q. Lee, and A. Lumsdaine, *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.
- [44] *Boost Phoenix 2*, Boost, 2006. [Online]. Available: <http://spirit.sourceforge.net/>
- [45] D. Abrahams and A. Gurtovoy, *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond (C++ in Depth Series)*. Addison-Wesley, 2004.
- [46] A. Priesnitz, “Multistage Algorithms in C++,” Dissertation, Universität Göttingen, 2005.
- [47] T. L. Veldhuizen and D. Gannon, “Active Libraries: Rethinking the Roles of Compilers and Libraries,” in *Proc. of the SIAM Workshop on Obj.-Oriented Methods for Inter-Operable Sci. and Eng. Comp. (OO’98)*. SIAM, 1998.
- [48] T. L. Veldhuizen, “Stage-Preserving Embeddings of Languages,” in *The 16th Nordic Workshop on Programming Theory (NWPT’04)*, October 2004, pp. 5–8.
- [49] T. Veldhuizen, “Tradeoffs in Metaprogramming,” in *Proc. of the 2006 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. New York, NY, USA: ACM Press, 2006, pp. 150–159.
- [50] K. Czarnecki and U. Eisenecker, *Generative Programming: Methods, Tools, and Applications*. New York, NY, USA: Addison-Wesley, 2000.
- [51] G. Berti, “GrAL - The Grid Algorithms Library,” in *ICCS ’02: Proc. of the Conf. on Comp. Sci.*, vol. 2331. London, UK: Springer, 2002, pp. 745–754.
- [52] J. C. Maxwell, *A Treatise on Electricity & Magnetism*. New York: Dover Publications, 1873.

- [53] P. Gross and P. R. Kotiuga, *Electromagnetic Theory and Computation: A Topological Approach*. Cambridge University Press, 2004.
- [54] B. A. Davey and H. A. Priestley, *Introduction to Lattices and Order*. Cambridge, 1990.
- [55] J. Hocking and G. Young, *Topology*. Dover Publications, New York: Addison-Wesley, 1961.
- [56] T. Dey, H. Edelsbrunner, and S. Guha, “Computational Topology,” in *Advances in Discrete and Computational Geometry*, ser. Contemporary Mathematics, J. E. G. B. Chazelle and R. Pollack, Eds., Providence, RI, USA, 1998.
- [57] Y. Chow, “Fiber Bundles, Sheaf Theory, and Generalization of Differentiable Manifolds in Physics,” *Intl. J. of Quantum Chemistry*, vol. 17, no. 1, pp. 85–97, 1980.
- [58] K. Kodaira, *Complex Manifolds and Deformation of Complex Structures*. Springer, 1986.
- [59] J. A. Chard and V. Shapiro, “A Multivector Data Structure for Differential Forms and Equations,” *Math. Comput. Simulation*, vol. 54, no. 1-3, pp. 33–64, 2000.
- [60] H. Flanders, *Differential Forms*. New-York: Academic Press, 1963.
- [61] K. Warnick, R. Selfridge, and D. Arnold, “Teaching Electromagnetic Field Theory Using Differential Forms,” *IEEE Trans. on Education*, vol. 40, no. 1, pp. 53–68, 1997.
- [62] F. H. Branin, “The Algebraic-Topological Basis for Network Analogies and for Vector Calculus,” in *Symposium on Generalized Networks (12–14 April 1966)*. Polytechnic Institute of Brooklyn, 1966, pp. 453–491.
- [63] J. M. Hyman and M. J. Shashkov, “Natural Discretizations for the Divergence, Gradient, and Curl on Logically Rectangular Grids,” *Comput. Math. Appl.*, vol. 33, no. 4, pp. 81–104, 1997.
- [64] —, “Adjoint Operators for the Natural Discretizations of the Divergence, Gradient, and Curl on Logically Rectangular Grids,” *Appl. Numer. Math.*, vol. 25, no. 4, pp. 413–442, 1997.
- [65] J. M. Hyman, M. J. Shashkov, and S. Steinberg, “The Numerical Solution of Diffusion Problems in Strongly Heterogeneous Non-Isotropic Materials,” *J. Comput. Phys.*, vol. 132, no. 1, pp. 130–148, 1997.
- [66] J. M. Hyman and M. J. Shashkov, “The Approximation of Boundary Conditions for Mimetic Finite Difference Methods,” *Computers & Mathematics with Applications*, vol. 36, pp. 79–99, 1998.
- [67] H. R. Schwarz, *Numerische Mathematik*. Stuttgart: B. G. Teubner, 1997.
- [68] S. Larsson and V. Thomee, *Partial Differential Equations with Numerical Methods*. Springer, 2003.
- [69] H. Ceric, “Numerical Techniques in Interconnect and Process Simulation,” Dissertation, Technische Universität Wien, 2004.
- [70] M. McIlroy, “Mass-Produced Software Components,” *Proc. NATO Conf. on Software Engineering, Garmisch, Germany*, 1968.
- [71] M. Jazayeri, “Component Programming - A Fresh Look at Software Components,” in *Proc. Software Engineering (ESEC’95)*. Berlin,: Springer, W. Schäfer and P. Botella, Eds., 1995, pp. 457–478.
- [72] *Boost C++ Libraries*, Boost. [Online]. Available: <http://www.boost.org>

- [73] *Boost Fusion 2*, Boost, 2006. [Online]. Available: <http://spirit.sourceforge.net/>
- [74] *Boost Xpressive*, Boost, 2006. [Online]. Available: <http://www.boost.org/>
- [75] *Boost Spirit*, Boost, 2006. [Online]. Available: <http://spirit.sourceforge.net/>
- [76] R. Garcia, J. Järvi, A. Lumsdaine, J. Siek, and J. Willcock, “A Comparative Study of Language Support for Generic Programming,” in *Proc. of the 18th Annual ACM SIGPLAN*. New York, NY, USA: ACM Press, 2003, pp. 115–134.
- [77] —, “An Extended Comparative Study of Language Support for Generic Programming,” *J. of Functional Programming*, vol. 17, no. 2, pp. 145–205, March 2007. [Online]. Available: <http://dx.doi.org/10.1017/S0956796806006198>
- [78] R. B. Haber, B. Lucas, and N. Collins, “A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids,” in *VIS '91: Proc. of the 2nd Conf. on Visualization '91*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 298–305.
- [79] E. Codd, “A Relational Model of Data for Large Shared Data Banks,” *Commun. ACM*, vol. 13, no. 6, pp. 377–387, 1970.
- [80] *IBM Visualization Data Explorer*, 3rd ed., IBM Corporation, Yorktown Heights, NY, USA, Feb. 1993.
- [81] E. Bethel, “Interoperability of Visualization Software and Data Models is Not an Achievable Goal,” in *VIS '03: Proc. of the 14th IEEE Visualization 2003 Conf. (VIS'03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 114.
- [82] W. Benger, G. Ritter, and R. Heinzl, “The Concepts of VISH,” in *Proc. of the 4th High-End Visualization Workshop*, Tyrol, Austria, June 18–22 2007, pp. 28–41.
- [83] R. Wirdemann and T. Baustert, *Rapid Web Development mit Ruby on Rails*. München: Hanser, 2006.
- [84] E. N. Volanschi, C. Counsel, G. Muller, and C. Cowan, “Declarative Specialization of Object-Oriented Programs,” in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.* New York, NY, USA: ACM Press, 1997, pp. 286–300.
- [85] R. Affeldt, H. Masuhara, E. Sumii, and A. Yonezawa, “Supporting Objects in Run-Time Bytecode Specialization,” in *Proc. of the Symp. on Part. Eval. and Semantics-Based Prog. Manip.* New York, NY, USA: ACM Press, 2002, pp. 50–60.
- [86] H. M. Andersen and U. P. Schultz, “Declarative Specialization for Object-Oriented-Program Specialization,” in *PEPM '04: Proc. of the 2004 ACM SIGPLAN Symp. on Part. Eval. and Semantics-Based Prog. Manip.* New York, NY, USA: ACM Press, 2004, pp. 27–38.
- [87] D. F. Bacon, S. L. Graham, and O. J. Sharp, “Compiler Transformations for High-Performance Computing,” in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.*, vol. 26, 1996, pp. 345–420.
- [88] D. Gay and B. Steensgaard, “Fast Escape Analysis and Stack Allocation for Object-Based Programs,” in *CC '00: Proc. of the 9th Conf. on Compiler Constr.* London, UK: Springer, 2000, pp. 82–93.

- [89] J. Järvi, D. Gregor, J. Willcock, A. Lumsdaine, and J. G. Siek, “Algorithm Specialization in Generic Programming - Challenges of Constrained Generics in C++,” in *PLDI '06: Proc. of the ACM SIGPLAN 2006 Conf. on Programming Language Design and Implementation*. New York, NY, USA: ACM Press, June 2006.
- [90] T. Geraud and A. Duret-Lutz, “Generic Programming Redesign Pattern,” in *Proc. of the 5th Conf. on Pattern Lang. of Progr. (EuroPLoP 2000)*, Irsee, Germany, 2000. [Online]. Available: citeseer.ist.psu.edu/geraud00generic.html
- [91] J. Järvi, J. Willcock, and A. Lumsdaine, “Concept-Controlled Polymorphism,” in *GPCE '03: Proc. of the 2nd Conf. on Generative Prog. and Comp. Eng.* New York, NY, USA: Springer, 2003, pp. 228–244.
- [92] A. Alexandrescu, *Modern C++ Design: Generic Programming and Design Patterns Applied*. Boston, MA, USA: Addison-Wesley, 2001.
- [93] D. Gregor, J. Järvi, M. Kulkarni, A. Lumsdaine, D. Musser, and S. Schupp, “Generic Programming and High-Performance Libraries,” *Intl. J. of Parallel Prog.*, vol. 33, no. 2, June 2005.
- [94] J. G. Siek and A. Lumsdaine, “Concept Checking: Binding Parametric Polymorphism in C++,” in *Proc. of the First Workshop on C++ Template Programming*, Erfurt, Germany, 2000. [Online]. Available: citeseer.nj.nec.com/siek00concept.html
- [95] A. Dekker, “Lazy Functional Programming in Java,” *ACM SIGPLAN Notices*, vol. 41, no. 3, pp. 30–39, March 2006.
- [96] M. Naftalin and P. Wadler, *Java Generics and Collections*. O'Reilly & Associates, 2006.
- [97] L. Prechelt, “An Empirical Comparison of Seven Programming Languages,” *Computer*, vol. 33, no. 10, pp. 23–29, 2000. [Online]. Available: citeseer.ist.psu.edu/article/prechelt00empirical.html
- [98] P. Wegner, “Concepts and Paradigms of Object-Oriented Programming,” *SIGPLAN OOPS Mess.*, vol. 1, no. 1, pp. 7–87, 1990.
- [99] J. Backus, “Can Programming be Liberated from the Von Neumann style?: A Functional Style and its Algebra of Programs,” *Commun. ACM*, vol. 21, no. 8, pp. 613–641, August 1978. [Online]. Available: <http://portal.acm.org/citation.cfm?id=359579>
- [100] T. L. Veldhuizen, “Expression Templates,” *C++ Report*, vol. 7, no. 5, pp. 26–31, June 1995, reprinted in *C++ Gems*, ed. Stanley Lippman.
- [101] *Boost Lambda Library*, Boost. [Online]. Available: <http://www.boost.org>
- [102] B. McNamara and Y. Smaragdakis, “Functional Programming in C++ using the FC++ Library,” *SIGPLAN*, vol. 36, no. 4, pp. 25–30, Apr. 2001.
- [103] K. Driesen and U. Hölzle, “The Direct Cost of Virtual Function Calls in C++,” *SIGPLAN Not.*, vol. 31, no. 10, pp. 306–323, 1996.
- [104] *GNU Compiler Collection (GCC)*. [Online]. Available: <http://gcc.gnu.org/>
- [105] S. C. Johnson, “YACC: Yet Another Compiler Compiler,” in *UNIX Programmer's Manual*. New York, NY, USA: Holt, Rinehart, and Winston, 1979, vol. 2, pp. 353–387.
- [106] O. Bagge, “CodeBoost: A Framework for Transforming C++ Programs,” Master's thesis, University of Bergen, P.O.Box 7800, N-5020 Bergen, Norway, March 2003.

- [107] J. Siek and A. Lumsdaine, “Software Engineering for Peak Performance,” *C++ Report*, pp. 23–27, May 2000.
- [108] T. Veldhuizen, “Guaranteed Optimization for Domain-Specific Programming,” in *Domain-Specific Program Generation*, ser. Lecture Notes in Computer Science, vol. 3016. Springer, 2004, pp. 306–324.
- [109] R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser, “Performance Aspects of a DSEL for Scientific Computing with C++,” in *Proc. of the POOSC Conf.*, Nantes, France, July 2006, pp. 37–41.
- [110] D. Abrahams, J. Siek, and T. Witt, “New Iterator Concepts,” ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++, Tech. Rep. N1477 03-0060, 2003.
- [111] S. Selberherr, A. Schütz, and H. Pötzl, “MINIMOS—A Two-Dimensional MOS Transistor Analyzer,” *IEEE Trans. Electron Dev.*, vol. ED-27, no. 8, pp. 1540–1550, 1980.
- [112] R. Sabelka and S. Selberherr, “A Finite Element Simulator for Three-Dimensional Analysis of Interconnect Structures,” *Microelectronics J.*, vol. 32, no. 2, pp. 163–171, 2001.
- [113] R. Bauer, “Numerische Berechnung von Kapazitäten in dreidimensionalen Verdrahtungsstrukturen,” Dissertation, Technische Universität Wien, 1994.
- [114] R. Sabelka, “Dreidimensionale Finite Elemente Simulation von Verdrahtungsstrukturen auf Integrierten Schaltungen,” Dissertation, Technische Universität Wien, 2001.
- [115] C. Fischer, “Bauelementsimulation in einer computergestützten Entwurfsumgebung,” Dissertation, Technische Universität Wien, 1994.
- [116] R. Klima, “Three-Dimensional Device Simulation with Minimos-NT,” Dissertation, Technische Universität Wien, 2002.
- [117] S. Wagner, “Small-Signal Device and Circuit Simulation,” Dissertation, Technische Universität Wien, 2005.
- [118] M. Radi, “Three-Dimensional Simulation of Thermal Oxidation,” Dissertation, Technische Universität Wien, 1998.
- [119] T. Binder, “Rigorous Integration of Semiconductor Process and Device Simulators,” Dissertation, Technische Universität Wien, 2002.
- [120] A. Fabri, “CGAL - The Computational Geometry Algorithm Library,” 2001. [Online]. Available: <http://citeseer.ist.psu.edu/fabri01cgal.html>
- [121] S. Pion and A. Fabri, “A Generic Lazy Evaluation Scheme for Exact Geometric Computations,” in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.*, Portland, OR, USA, October 2006, pp. 75–84.
- [122] M. Haverlaen, H. Friis, and T. Johansen, “Formal Software Engineering for Computational Modeling,” *Nordic J. of Computing*, vol. 3, no. 6, pp. 241–270, 1999.
- [123] C. S. Rafferty and R. K. Smith, “Solving Partial Differential Equations with the Prophet Simulator,” Bell Laboratories, Lucent Technologies, 1996. [Online]. Available: <http://www-tcad.stanford.edu/prophet/math.pdf>
- [124] W. Bangerth, R. Hartmann, and G. Kanschat, deal.II *Differential Equations Analysis Library*, Technical Reference. [Online]. Available: <http://www.dealii.org/>

- [125] P. Castillo, R. Rieben, and D. White, “FEMSTER: An Object-Oriented Class Library of High-Order Discrete Differential Forms,” *ACM Trans. Math. Softw.*, vol. 31, no. 4, pp. 425–457, 2005.
- [126] A. Logg, T. Dupont, J. Hoffman, C. Johnson, R. C. Kirby, M. G. Larson, and L. R. Scott, “The FEniCS Project,” Chalmers Finite Element Center, Tech. Rep. 2003-21, 2003.
- [127] K. Czarnecki and U. W. Eisenecker, “Components and Generative Programming,” in *ESEC/FSE-7: Proc. 7th ESEC*. London, UK: Springer, 1999, pp. 2–19.
- [128] R. Pozo, “Template Numerical Toolkit for Linear Algebra: High Performance Programming with C++ and the Standard Template Library,” *Intl. J. of High Performance Computing Applications*, vol. 11, no. 3, pp. 251–263, Fall 1997.
- [129] W. Bangerth, R. Hartmann, and G. Kanschat, “deal.II – A General Purpose Object-Oriented Finite Element Library,” Institute for Scientific Computation, Texas A&M University, Tech. Rep. ISC-06-02-MATH, 2006.
- [130] J. A. Lewis, S. M. Henry, D. G. Kafura, and R. S. Schulman, “An Empirical Study of the Object-Oriented Paradigm and Software Reuse,” in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.* New York, NY, USA: ACM Press, 1991, pp. 184–196.
- [131] T. L. Veldhuizen, “Five Compilation Models for C++ Templates,” in *1st Workshop on C++ Template Programming*, October 2000. [Online]. Available: <http://oonumerics.org/tmpw00/>
- [132] S. Selberherr, *Analysis and Simulation of Semiconductor Devices*. Wien–New York: Springer, 1984.
- [133] H. P. Langtangen and X. Cai, “Mixed Language Programming for HPC Applications,” in *Proc. of the PARA Conf.*, Umea, Sweden, June 2006, p. 154.
- [134] *Amira Advanced 3D Visualization and Volume Modeling*, Mercury Computer Systems. [Online]. Available: <http://www.amiravis.com/>
- [135] *Paraview*, Paraview. [Online]. Available: <http://www.paraview.org/>
- [136] *FiberHDF5*, Fiber Bundle HDF5 Library. [Online]. Available: <http://www.fiberbundle.net/>
- [137] R. Heinzl and T. Grasser, “Generalized Comprehensive Approach for Robust Three-Dimensional Mesh Generation for TCAD,” in *Proc. Conf. on Sim. of Semiconductor Processes and Devices*, Tokio, September 2005, pp. 211–214.
- [138] P. Fleischmann and S. Selberherr, “Enhanced Advancing Front Delaunay Meshing in TCAD,” in *Proc. Conf. on Sim. of Semiconductor Processes and Devices*, 2002, pp. 99–102.
- [139] J. Schöberl, “NETGEN - An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules,” *Comput. Visual. Sci.*, vol. 1, pp. 41–52, 1997.
- [140] H. Si, “On Refinement of Constrained Delaunay Tetrahedralizations,” *15th Intl. Meshing Roundtable*, September 2006.
- [141] S. Balay, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang, “PETSc Users Manual,” Argonne National Laboratory, Tech. Rep. ANL-95/11 - Revision 2.3.2, September 2006. [Online]. Available: <http://www.mcs.anl.gov/petsc/docs>

- [142] S. Mauch, "Efficient Algorithms for Solving Static Hamilton-Jacobi Equations," Dissertation, California Institute of Technology, Purdue, CA, 2003.
- [143] L. Lee and A. Lumsdaine, "Generic Programming for High Performance Scientific Applications," in *JGI '02: Proc. of the 2002 joint ACM-ISCOPE Conf. on Java Grande*. New York, NY, USA: ACM Press, 2002, pp. 112–121.
- [144] C. E. Oancea and S. M. Watt, "Parametric Polymorphism for Software Component Architectures," in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.* New York, NY, USA: ACM Press, 2005, pp. 147–166.
- [145] T. L. Veldhuizen, "Using C++ Template Metaprograms," *C++ Report*, vol. 7, no. 4, pp. 36–43, May 1995, reprinted in *C++ Gems*, ed. Stanley Lippman.
- [146] J. Siek and A. Lumsdaine, "The Matrix Template Library: Generic Components for High-performance Scientific Computing," *Computing in Science and Engineering*, vol. 1, no. 6, pp. 70–78, Nov/Dec 1999.
- [147] R. Whaley and J. Dongarra, "Automatically Tuned Linear Algebra Software," in *Proc. of the 1998 ACM/IEEE Conf. on Supercomputing (CDROM)*. IEEE Computer Society, 1998, pp. 1–27, cD-ROM Proc.
- [148] *Stream - Sustainable Memory Bandwidth in High Performance Computers*, Boost. [Online]. Available: <http://www.cs.virginia.edu/stream/>
- [149] J. G. S. Todd L. Veldhuizen, "Combining Optimizations, Combining Theories," Indiana University, Tech. Rep. 582, May 2003. [Online]. Available: <http://www.cs.indiana.edu/cgi-bin/techreports/TRNNN.cgi?trnum=TR582>
- [150] T. L. Veldhuizen, "Arrays in Blitz++," in *Proc. Symp. on Comp. in Obj.-Oriented Parallel Env.*, ser. Lecture Notes in Computer Science. Springer, 1998.
- [151] S. Schupp, M. Zalewski, and K. Ross, "Rapid Performance Prediction for Library Components," in *Proc. 4th. ACM Workshop on Software and Performance (WOSP 2004), Redwood City*. ACM Press, 2004, pp. 69–73.
- [152] T. Hiramoto, T. Nagumo, T. Ohtou, and K. Yokoyama, "Device Design of Nanoscale MOS-FETs Considering the Suppression of Short Channel Effects and Characteristics Variations," *IEICE Transactions on Electronics*, vol. E90-C, no. 4, pp. 836–841, 2007.
- [153] R. Heinzl, "A Three-Dimensional Analytical Ion Implantation Tool Using the Wafer-State-Server," Master Thesis, Technische Universität Wien, 2006.
- [154] D. Scharfetter and H. Gummel, "Large-Signal Analysis of a Silicon Read Diode Oscillator," *IEEE Trans. Electron Dev.*, vol. 16, no. 1, pp. 64–77, 1969.
- [155] M. Kamon, S. McCormick, and K. Sheperd, "Interconnect Parasitic Extraction in the Digital IC Design Methodology," in *ICCAD '99: Proc. of the 1999 IEEE/ACM Intl. Conf. on Computer-Aided Design*. Piscataway, NJ, USA: IEEE Press, 1999, pp. 223–231.
- [156] D. Abrahams, J. Siek, and T. Witt, "New Iterator Concepts," ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++, Tech. Rep. N1550=03-0133, April 2006.

- [157] D. Gregor, J. Willcock, and A. Lumsdaine, "Concepts for the C++0x Standard Library: Iterators," ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++, Tech. Rep. N2039=06-0109, June 2006.
- [158] M. Zalewski and S. Schupp, "Changing Iterators with Confidence. A Case Study of Change Impact Analysis Applied to Conceptual Specifications," in *Proc. of the Object-Oriented Programming Systems, Languages, and Applications Conf.*, San Diego, CA, USA, October 2005.
- [159] D. Gregor, J. Willcock, and A. Lumsdaine, "Concepts for the C++0x Standard Library: Iterators," ISO/IEC JTC 1, Information Technology, Subcommittee SC 22, Programming Language C++, Tech. Rep. N2039=06-0109, June 2006.

Own Publications

Book Editorships:

- [E1] W. Benger, R. Heinzl, W. Kapferer, W. Schoor, M. Tyagi, S. Venkataraman, G. Weber (Eds.): Proceedings of the 4th High-End Visualization Workshop. Tyrol, Austria, July 2007, ISBN: 978-3-86541-216-4

Contributions to Books:

- [B1] H. Ceric, R. Heinzl, Ch. Hollauer, T. Grasser, S. Selberherr. Microstructure and Stress Aspects of Electromigration Modeling. In *Stress-Induced Phenomena in Metallization*, American Institute of Physics, Melville, 2006, ISBN: 0-7354-03104, 262 - 268.

Papers in Journals:

- [J1] R. Heinzl, P. Schwaha, and T. Grasser. A High Performance Generic Scientific Simulation Environment. In *Lecture Notes in Computer Science*, Springer, accepted.
- [J2] M. Spevak, R. Heinzl, P. Schwaha, and T. Grasser. A Computational Framework for Topological Operations. In *Lecture Notes in Computer Science*, Springer, accepted.
- [J3] R. Heinzl, P. Schwaha, and S. Selberherr. A Generic Topology Library. In *Science of Computer Programming*, Elsevier, submitted.

Publications in Conference Proceedings:



- [P1] A. Sheikholeslami, E. Al-Ani, R. Heinzl, C. Heitzinger, F. Parhami, F. Badrieh, H. Puchner, T. Grasser, and S. Selberherr. Level Set Method Based General Topography Simulator and its Applications in Interconnect Processes. In *Intl. Conf. on Ultimate Integration of Silicon*, pages 139–142, Bologna, Italy, July 2005.
- [P2] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. A Novel Technique for Coupling Three-Dimensional Mesh Adaptation with an A Posteriori Error Estimator. In *Proc. 2005 PhD Research in Microel. and Elect.*, pages 175–178, Lausanne, Switzerland, July 2005.
- [P3] A. Sheikholeslami, F. Parhami, R. Heinzl, F. Badrieh, H. Puchner, T. Grasser, and S. Selberherr. Applications of Three-Dimensional Topography Simulation in the Design of Interconnect Lines. In *Proc. Conf. Sim. of Semiconductor Processes and Devices*, pages 187–190, Tokio, Japan, September 2005.

- [P4] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser. Coupling Three-Dimensional Mesh Adaptation with an A Posteriori Error Estimator. In *Proc. Conf. Sim. of Semiconductor Processes and Devices*, pages 235–238, Tokio, Japan, September 2005.
- [P5] R. Heinzl and T. Grasser. Generalized Comprehensive Approach for Robust Three-Dimensional Mesh Generation for TCAD. In *Proc. Conf. Sim. of Semiconductor Processes and Devices*, pages 211–214, Tokio, Japan, September 2005.
- [P6] P. Schwaha, R. Heinzl, W. Brezna, J. Smoliner, H. Enichlmair, R. Minixhofer, and T. Grasser. Fully Three-Dimensional Analysis of Leakage Current in Non-Planar Oxides. In *2005 Europ. Sim. and Modeling Conf.*, pages 469–473, Porto, Portugal, October 2005.
- [P7] E. Al-Ani, R. Heinzl, P. Schwaha, T. Grasser, and S. Selberherr. Three-Dimensional State of the Art Topography Simulation. In *2005 Europ. Sim. and Modeling Conf.*, pages 430–432, Porto, Portugal, October 2005.
- [P8] R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser. Adaptive Mesh Generation for TCAD with Guaranteed Error Bounds. In *Proc. 2005 Europ. Sim. and Modeling Conf.*, pages 425–429, Porto, Portugal, October 2005.
- [P9] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. Concepts for High Performance Generic Scientific Computing. In *Proc. of the MATHMOD Conf.*, volume 1, page 34 Vienna, Austria, February 2006.
- [P10] M. Spevak, R. Heinzl, P. Schwaha, and T. Grasser. Simulation of Microelectronic Structures using A Posteriori Error Estimation and Mesh Adaptation. In *Proc. of the MATHMOD Conf.*, volume 1, page 317, Vienna, Austria, February 2006.
- [P11] M. Spevak, R. Heinzl, P. Schwaha, and T. Grasser. Process and Device Simulation with a Generic Scientific Simulation Environment. In *Proc. of the Intl. Conf. on Microelectronics*, volume 2, pages 475–478, Belgrad, Serbia and Montenegro, April 2006.
- [P12] P. Schwaha, R. Heinzl, W. Brezna, J. Smoliner, H. Enichlmair, R. Minixhofer, and T. Grasser. Leakage Current Analysis of a Real World Silicon-Silicon Dioxide Capacitance. In *Proc. of the Intl. Caribbean Conf. on Device, Circuits and Systems*, pages 365–370, Playa del Carmen, Mexico, April 2006.
- [P14] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. Multidimensional and Multitopological TCAD with a Generic Simulation Environment. In *Proc. of the Intl. Caribbean Conf. on Device, Circuits and Systems*, pages 173–176, Playa del Carmen, Mexico, April 2006.
- [P15] R. Heinzl, M. Spevak, and P. Schwaha. A Novel High Performance Approach for Technology Computer Aided Design. In *Proc. of the Electrical Engineering, Information and Communication Technologies Conf.*, volume 4, pages 446–450, Brno, Czech Rep., April 2006.
- [P16] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. A Generic Scientific Simulation Environment for Multidimensional Simulation in the Area of TCAD. In *Proc. of the NSTI.*, volume 4, pages 526–529, Boston, USA, May 2006.
- [P17] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. High Performance Process and Device Simulation with a Generic Environment. In *Proc. of the 14th Iranian Conf. on El. Eng. ICEE 2006*, pages 446–450, Teheran, Iran, May 2006.

- [P18] A. Sheikholeslami, R. Heinzl, S. Hozler, C. Heitzinger, M. Spevak, M. Leicht, O. Häberlen, J. Fugger, F. Badrieh, F. Parhami, H. Puchner, T. Grasser, and S. Selberherr. Applications of Two- and Three-Dimensional General Topography Simulator in Semiconductor Manufacturing Processes. In *Proc. of the 14th Iranian Conf. on El. Eng. ICEE 2006*, pages 446–450, Teheran, Iran, May 2006.
- [P19] M. Spevak, R. Heinzl, P. Schwaha, and T. Grasser. A Computational Framework for Topological Operations. In *Proc. of the PARA Conf.*, page 57, Umea, Sweden, June 2006.
- [P20] R. Heinzl, M. Spevak, P. Schwaha, and T. Grasser. A High Performance Generic Scientific Simulation Environment. In *Proc. of the PARA Conf.*, page 61, Umea, Sweden, June 2006.
- [P21] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser. Advanced Equation Processing for TCAD. In *Proc. of the PARA Conf.*, page 64, Umea, Sweden, June 2006.
- [P22] R. Heinzl, P. Schwaha, M. Spevak, and T. Grasser. Performance Aspects of a DSEL for Scientific Computing with C++. In *Proc. of the Parallel/High Performance Object-Oriented Scientific Computing Conf.*, pages 37–41, Nantes, France, July 2006.
- [P23] P. Schwaha, R. Heinzl, M. Spevak, and T. Grasser. A Generic Approach to Scientific Computing. In *Proc. of the Intl. Congress on Computational and Applied Mathematics*, page 137, Leuven, Belgium, July 2006.
- [P24] M. Spevak, P. Schwaha, R. Heinzl, and T. Grasser. Automatic Linearization using Functional Programming for Scientific Computing. In *Proc. of the Intl. Congress on Computational and Applied Mathematics*, page 147, Leuven, Belgium, July 2006.
- [P25] M. Spevak, R. Heinzl, P. Schwaha, T. Grasser, and S. Selberherr. A Generic Discretization Library. In *Proc. of the Object-Oriented Programming Systems, Languages and Applications*, pages 95–100, Portland, USA, October 2006.
- [P26] R. Heinzl, M. Spevak, P. Schwaha, and S. Selberherr. A Generic Topology Library. In *Proc. of the Object-Oriented Programming Systems, Languages and Applications*, pages 85–93, Portland, USA, October 2006.
- [P27] R. Heinzl, M. Spevak, P. Schwaha, and S. Selberherr. Performance Analysis for High-Precision Interconnect Simulation. In *Proc. of the European Sim. and Modelling Conf.*, pages 113–116, Toulouse, France, October 2006.
- [P28] P. Schwaha, R. Heinzl, G. Mach, C. Pogoreutz, S. Fister, and S. Selberherr. A High Performance Webapplication for an Electro-Biological Problem. In *Proc. of the European Conf. on Modelling and Sim.*, pages 218–222, Prague, Czech Republic, June 2007.
- [P29] R. Heinzl, P. Schwaha, C. Giani, and S. Selberherr. Modeling of Non-Trivial Data Structures with a Generic Scientific Simulation Environment. In *Proc. of the 4th High-End Visualization Workshop*, pages 3–11, Tyrol, Austria, June 2007.
- [P30] P. Schwaha, C. Giani, R. Heinzl, and S. Selberherr. Visualization of Polynomials Used in Series Expansion. In *Proc. of the 4th High-End Visualization Workshop*, pages 137–146, Tyrol, Austria, June 2007.
- [P31] W. Benger, G. Ritter, and R. Heinzl. The Concepts of VISH. In *Proc. of the 4th High-End Visualization Workshop*, pages 26–39, Tyrol, Austria, June 2007.
- [P32] R. Heinzl, P. Schwaha, and S. Selberherr. Modern Concepts for High-Performance Scientific Computing. In *Proc. of the Intl. Conf. on Software and Data Technologies*, pages 100–107, Barcelona, Spain, July 2007.

- [P33] P. Schwaha, R. Heinzl, and S. Selberherr. Simulation Methodologies for Scientific Computing. In *Proc. of the Intl. Conf. on Software and Data Technologies*, pages 270–276, Barcelona, Spain, July 2007.
- [P34] R. Heinzl, G. Mach, P. Schwaha, and S. Selberherr. Labtool - A Managing Software for Computer Courses. In *Proc. of the European Sim. and Modelling Conf.*, accepted, St. Julian's, Malta, October 2007.
- [P35] R. Heinzl, G. Mach, P. Schwaha, and S. Selberherr. A Performance Test Platform. In *Proc. of the European Sim. and Modelling Conf.*, accepted, St. Julian's, Malta, October 2007.
- [P36] P. Schwaha, R. Heinzl, and S. Selberherr. Electro-Biological Simulation using a Web Front-End. In *Proc. of the European Sim. and Modelling Conf.*, accepted, St. Julian's, Malta, October 2007.
- [P37] F. Stimpfl, R. Heinzl, P. Schwaha, and S. Selberherr. A Multimode Mesh Generation Approach for Scientific Computing. In *Proc. of the European Sim. and Modelling Conf.*, accepted, St. Julian's, Malta, October 2007.

Curriculum Vitae

	
1997	High School Graduation at the HTL Donaustadt, Wien
1997-1998	Military service
October 1998	Enrolled in Electrical Engineering at the Technical University of Vienna
October 2003	Received degree of “Diplomingenieur” in Electrical Engineering with a specialization in Computer Technology from the Technical University of Vienna
November 2003	Entered doctoral program at the Institute for Microelectronics, TU Vienna
September 2007	Finished doctoral program at the Institute for Microelectronics, TU Vienna