The LIR Space Partitioning System applied to the Stokes Equations

Vom Fachbereich Informatik der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation
von

M. Sc. Sven Linden

Datum der wissenschaftlichen Aussprache: 06.11.2014

Dekan:	Prof. Dr. Klaus Schneider
1. Berichterstatter:	Prof. Dr. Hans Hagen
2. Berichterstatter:	Prof. Dr. David Adalsteinsson

D (386)

Danksagung

An dieser Stelle möchte ich meinem Doktorvater Prof. Hagen und meinem Betreuer Dr. Wiegmann für die Unterstützung und die zahlreichen Gespräche danken. Ohne sie wäre diese Arbeit nicht möglich gewesen. Sie ermöglichten mir sämtliche Ideen, egal wie ausgefallen sie waren, zu implementieren und zu analysieren. Dennoch haben sich mich einige Male wieder zurück auf einen sinnvollen Weg geführt, wenn ich zu besessen von einem Ansatz war. Für die zahlreichen Diskussionen und Denkanstöße während meines Auslandsaufenthalts möchte ich mich bei Prof. David Adalsteinsson bedanken.

Ebenfalls möchte ich mich bei dem Fraunhofer ITWM, der International Research Training Group 1131 und der Math2Market GmbH für die wissenschaftliche, finanzielle und organisatorische Unterstützung der letzten 3 Jahre bedanken.

Meiner Familie bin ich zutiefst dankbar für die nötige moralische Unterstützung. Sie hatten immer ein offenes Ohr für meine Sorgen und haben mir in jeder Lebenssituation Beistand geleistet.

Zu guter Letzt möchte ich meinen Freunden für die zahlreichen Ablenkungen, sozialen Ereignisse und tiefgründigen Gespräche während meiner kreativen Phasen danken.

Zusammenfassung

Diese Arbeit befasst sich mit zwei Themen: Räumliche Gebietszerlegung und die Simulation von kriechenden Strömungen in repräsentativen Volumina.

Im ersten Teil stellen wir ein neuartiges Verfahren zur multi-dimensionalen Gebietszerlegung [24] vor. Diese Baumstruktur vereinigt die Vorteile des Octrees [19] und des KD-trees [4] ohne dessen Nachteile. Sie erlaubt lokale Verfeinerung, Parallelisierung und Erhaltung mathematisch günstiger Eigenschaften. Die Baumstruktur wird über eine topologische Algebra, basierend auf der Menge der drei Symbole $A = \{L, I, R\}$, konstruiert und LIR-Baum genannt. Die Menge der Nachfolger wird beschränkt, so dass jeder innere Knoten die Eigenschaft der Partition der Einheit erfüllt. Unsere Methode erlaubt die Benutzung von Splines um wissenschaftliche Größen, wie z.B. Geschwindigkeit oder Druck, zu komprimieren oder zu rekonstruieren. Wir präsentieren einen Generator zur Konstruktion eines Baums der Voxel Geometrien repräsentiert. Die Gebietszerlegung wird dabei als Programmiergerüst für numerische Fragestellungen benutzt. Diese Arbeit ist motivert durch die mathematisch günstige Repräsentation von riesigen Voxel Geometrien. Diese Geometrien können aus mehreren Milliarden Voxeln bestehen und treten im Kontext großer repräsentativer Volumina (REV) [23] auf.

Im zweiten Teil stellen wir einen neuartigen Ansatz zur Anordnung von Druck- und Geschwindigkeitsvariablen für die Lösung der Stokes Gleichungen vor. Die Grundidee unserer Methode besteht darin, dass jede Zelle, unabhängig von ihren Nachbarzellen, ein gegebenes physikalisches Gesetz erfüllen kann. Dies wird erreicht durch Spaltung der Geschwindigkeitsvariablen zu einer links- und rechtsseitig konvergierenden Komponente. In jeder Zelle kann ein kleines lineares Gleichungssystem aufgestellt werden, dass die Impuls- und Massenerhaltungsgleichungen beschreiben. Diese Formulierung erlaubt die Benutzung des Gauß-Seidel Algorithmus zur Lösung des globalen Gleichungssystems. Der LIR-Baum wird für die Gebietszerlegung und Konstruktion einer initialen Diskretisierung benutzt. Zusätzlich stellen wir ein Verfahren vor, dass das aktuelle Geschwindigkeitsfeld benutzt um die Baumstruktur lokal zu verfeinern zur Erhöhung der numerischen Genauigkeit an benötigten Stellen. Wir benutzen keine bestehenden Methoden wie den SIMPLE Algorithmus [27], Lattice-Boltzmann Methoden [18] oder Explizite-Sprung Methoden [36], da sie für regelmäßige Gitter gedacht sind. Andere CFD Ansätze extrahieren Oberflächen und konstruieren ein Tetraeder-Netz. Daher können sie ebenfalls nicht direkt mit der vorgestellten Datenstruktur benutzt werden. Unsere Diskretisierung konvergiert gegen die analytische Lösung unter Gitterverfeinerung. Wir folgern eine hervorragende Perfomanz in Bezug auf Rechenzeit und Speicherbedarf in Geometrien mit einer hohen Porosität. In Geometrien mit einer niedrigen Porosität erhalten wir eine gute Perfomanz in Bezug auf Speicherbedarf.

Abstract

We consider two major topics in this thesis: spatial domain partitioning which serves as a framework to simulate creep flows in representative volume elements.

First, we introduce a novel multi-dimensional space partitioning method [24]. A new type of tree combines the advantages of the Octree [19] and the KD-tree [4] without having their disadvantages. We present a new data structure allowing local refinement, parallelization and proper restriction of transition ratios between nodes. Our technique has no dimensional restrictions at all. The tree's data structure is defined by a topological algebra based on the symbols $A = \{L, I, R\}$ that encode the partitioning steps. The set of successors is restricted such that each node has the partition of unity property to partition domains without overlap. With our method it is possible to construct a wide choice of spline spaces to compress or reconstruct scientific data such as pressure and velocity fields and multidimensional images. We present a generator function to build a tree that represents a voxel geometry. The space partitioning system is used as a framework to allow numerical computations. This work is triggered by the problem of representing, in a numerically appropriate way, huge three-dimensional voxel geometries that could have up to billions of voxels. These large datasets occure in situations where it is needed to deal with large representative volume elements (REV) [23].

Second, we introduce a novel approach of variable arrangement for pressure and velocity to solve the Stokes equations. The basic idea of our method is to arrange variables in a way such that each cell is able to satisfy a given physical law independently from its neighbor cells. This is done by splitting velocity values to a left and right converging component. For each cell we can set up a small linear system that describes the momentum and mass conservation equations. This formulation allows to use the Gauß-Seidel algorithm to solve the global linear system. Our tree structure is used for spatial partitioning of the geometry and provides a proper initial guess. In addition, we introduce a method that uses the actual velocity field to refine the tree and improve the numerical accuracy where it is needed. We developed a novel approach rather than using existing approaches such as the SIMPLE algorithm [27], Lattice-Boltzmann methods [18] or Exlicit jump methods [36] since they are suited for regular grid structures. Other standard CFD approaches extract surfaces and creates tetrahedral meshes to solve on unstructured grids thus can not be applied to our datastructure. The discretization converges to the analytical solution with respect to grid refinement. We conclude a high strength in computational time and memory for high porosity geometries and a high strength in memory requirement for low porosity geometries.

Contents

List of Figures

List of Tables

Nomenclature

η	Viscosity constant
$\mathcal{C}$	Cells
$\mathcal{E}$	Edges
$\mathcal{G}$	Tree
$\mathcal{X}$	Nodes
$\mathfrak{B}$	Set of intervals
$\mho$	Oracle
ι	Modifier function
Ω	Vector oracle
ϕ	Unity system
ψ	Input function
Υ	Ternary oracle
ξ	Interval partitioning system
A	Ternary alphabet
I	Identity symbol
L	Left symbol
P	Partitions of unity
p	Pressure variable
R	Right symbol
u	Velocity variable

Chapter 1

Introduction

Simulation of materials received great attention in the last decades. Expensive and time-consuming construction of prototypes used in real experiments can be replaced by virtual material design and simulation of physical equations. Numerical simulations are suited for material engineering. They can be accomplished in a short amount of time and are considerably cheaper than real experiments. The spatial size of simulated domains can range from nanometer- to meter-scale. Engineers are interessted in effective material properties (e.g. permeability) of virtual materials or computed tomographies. In addition, point-wise quantities such as velocity or pressure are interesting to gain a better understanding of physical behaviors. Automated parameter studies with respect to virtual material generation are possible to construct materials with desired properties. Therefore, engineers need simulation tools that fits into their workflow and supports them to increase the quality and properties of manufactured products.

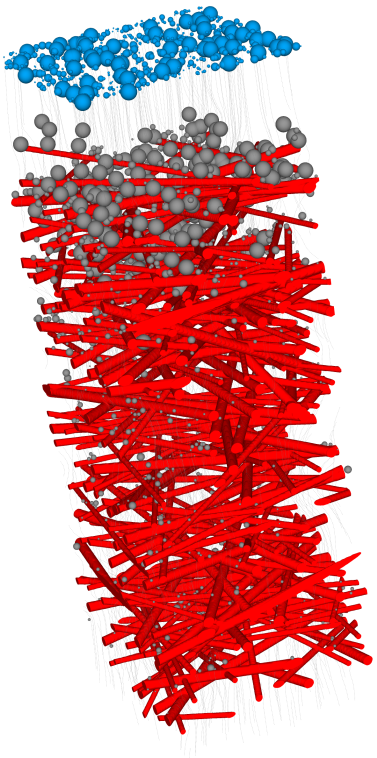


Figure 1.1: Filtration simulation with particles inside a fiber geometry

In this work, we focus on the stationary Stokes equations which is a simplification of the general Navier-Stokes equations where velocity and pressure are considered. Numerical computations allow to determine the effective permeability tensor for heterogeneous porous media. The Stokes equations are valid in slow flow regimes thus restrict to certain applications. An important research area can be found in geoscience.

The complex pore structure inside different kinds of ground guides physical behaviors and is a challenge for numerical solvers. Typically, porosity is low and geometric information is retrieved from computed tomographies. Another important application area are manufactured filter elements, e.g. for automobile industry. In that context, fiber geometries are from greater interest and engineers want to do parameter studies with respect to fiber distribution and orientations. In contrast to geoscience, filter elements tend to high porosity environments. Figure 1.1 shows simulated particles inside a fiber geometry.

Challenging for numerical flow solvers are huge geometries described by up to billions of voxel. Therefore, efficient memory approaches and fast algorithms are needed to satisfy these requirements. Numerical simulation can be split into two different aspects: space partitioning and solving of a linear system that describes the Stokes equations.

Space partitioning considers the way of partition a given domain such that it can be used to arrange variables in numerical applications. The efficient representation of billions of cells is a great challenge and requires enormous computational resources. Regular grid structures in a higher dimensional settings hit the wall very quickly. Therefore, adaptive space partitioning is needed to solve that problem. Sophisticated approaches can be used to provide higher accuracy at regions where it is needed. Moreover, the way of partitioning should reflect features in physical quantities of flow simulations. Memory efficiency is a very important aspect and parallelization is also needed to utilize huge computational resources.

Velocity and pressure variables have to be arranged by an adaptive data structure such that we can establish a linear system that describes the Stokes equations. The linear system should have properties that allows its solution without auxiliary variables due to the restrictive memory requirements. In addition, a single iteration step towards its solution should be very cheap to satisfy given time-constraints.

The described challenges imply the following objectives of this work:

First, the development and implementation of a memory efficient and high performance data structure that is able to represent huge voxel geometries in a numerically suited way. The data structure should serve as a framework to solve partial differential equations in computational fluid dynamics.

Second, the development and implementation of a very fast, stable and sufficiently accurate numerical solver for the Stokes equations that can be generalized to solve the (Navier)-Stokes-(Brinkman) equations.

We claim to solve these tasks by just using a ternary alphabet used in an elegant way.

Chapter 2

The LIR Space Partitioning System

Abstract

We introduce a novel multi-dimensional space partitioning method [24]. A new type of tree combines the advantages of the Octree [19] and the KD-tree [4] without having their disadvantages. The data structure allows local refinement, parallelization and proper restriction of transition ratios between leafs. Our technique has no dimensional restrictions at all. The tree's data structure is defined by a topological algebra based on the symbols $A = \{L, I, R\}$ that encode the partitioning steps. The set of successors is restricted such that each cell has the partition of unity property to partition domains without overlap. With our method it is possible to construct a wide choice of spline spaces to compress or reconstruct scientific data such as pressure and velocity fields and multidimensional images. We present a generator function to build a tree that represents a voxel geometry. The space partitioning system is used as a framework to allow numerical computations. This work is triggered by the problem of representing, in a numerically appropriate way, huge three-dimensional voxel geometries that could have up to billions of voxels.

2.1 Introduction

The goal of this work is to introduce and apply a novel mathematical model to partition n-dimensional domains. We give a detailed definition and description of the theoretical background and the algorithms. The space partitioning is done by a tree that can be seen as a hybrid of an Octree and a KD-tree. We combine the advantages while avoiding the disadvantages.

The basic idea of the tree is the definition of a ternary alphabet. This alphabet is applied recursively and dimensionally independently to a system of functions. We abstract from the geometrical properties and define an algebraic approach to efficiently partition domains and evaluate functions recursively. Evaluation and computation as well as proofs on these trees can be done by

structural induction.

The tree can be used to compress geometries as well as scalar and multi-dimensional fields. It is possible to access different kinds of information and operators that are influenced by the structure of the tree, e.g. interpolation schemes, differential operators, subsets of the given domain and neighborhoods of cells. In this work we focus on voxel geometries. But it is also possible to partition different kinds of sets. In many applications it is a disadvantage if the tree degenerates in a single direction. Therefore we introduce a non-degenerative input function that analyses voxel geometries.

2.2 Related Work

There are many different kinds of space partitioning methods. Regular grids are an easy commonly used way to discretize the two- and three-dimensional space. They allow an efficient alignment of the data in the memory and enables the user to easily formulate discretized differential operators. Data can be accessed in constant time, but the disadvantage of the regular grids is that the requirements for computational effort and memory grows at least with the power of the dimension. It is also not easy to describe smooth boundaries, e.g. the interfaces in two-phase flows and between different materials in solid mechanics.

A way to treat this problem is to use a body-fitted mesh of tetrahedra. But a mesh of tetrahedra requires a lot of overhead, e.g. you have to store positions, normal vectors and topological information. This overhead significantly decreases the number of cells that can be stored and processed.

Another way to treat this problem is to use an Octree [19] [20]. A disadvantage of the Octree is the limited choice of partitioning. You have just the choice to do no partitioning or to partition simultaneously in all dimensions. Therefore one is forced to increase the number of cells even in directions where it is not necessary. But the Octree is an important data structure in numerical mathematics. It received attention in isogeometric analysis [12] [13] within the last years.

The latter issue is addressed by the KD-tree [4]. It is able to increase the number of cells just in one direction. But a KD-tree is not well suited to use for numerical calculations as it is designed for partitioning point clouds. Another problem is the high number of interior nodes in higher dimensional settings.

A detailed description of the Octree and KD-tree can also be found in [28] and [29] covering the latest developments and applications. The model we describe in this chapter avoids the disadvantages of the KD-tree and Octree and combines their advantages.

2.3 Alphabet

For a one-dimensional finite interval there exist three choices: no partitioning, partition and take the left part of the interval, or partition and take the right part of the interval. Similar to partitioning there exist three choices for embedding intervals: no embedding, embed to the left or embed to the right. These choices are used to define the alphabet:

Definition 1. *A is called the* ***alphabet*** *defined by*

$$A := \{L, I, R\} \tag{2.1}$$

and contains three symbols that denote: L - left, I - identity and R - right. L and R are interpreted as complementary symbols and I as neutral symbol. We introduce a unary minus operator defined by

$$-L := R \quad -R := L \quad -I := I \tag{2.2}$$

that is also used vector-wise.

Definition 2. *The bold notation is used to see the symbols in A as sets by*

$$\mathbf{A} := \{\mathbf{L}, \mathbf{I}, \mathbf{R}\} \tag{2.3}$$

with the symbol sets

$$\mathbf{L} := \{L\} \quad \mathbf{I} := \{L, R\} \quad \mathbf{R} := \{R\} \tag{2.4}$$

and to introduce the conversion

$$v = (v_1, \cdots, v_n) \in A^n \Leftrightarrow \mathbf{v} = \mathbf{v_1} \times \cdots \times \mathbf{v_n} \in \mathbf{I}^n. \tag{2.5}$$

Definition 3. *The set of vectors of symbols defined by*

$$P := \{p \subseteq A^n : \bigcup_{v \in p} \mathbf{v} = \mathbf{I}^n \wedge \forall_{\substack{v,w \in p \\ v \neq w}} \mathbf{v} \cap \mathbf{w} = \emptyset\} \tag{2.6}$$

denotes all sets of vectors that are partitions of unity. The sets in P are the basis to construct the LIR-tree.

Definition 4. *We use S_n to denote the symmetric group and $\{1, -1\}^n$ as selective inversion. Let $p, q \in P$ be two partitions of unity. $p \sim q$ means they are equivalent with respect to rotation and inversion. That is*

$$p \sim q \Leftrightarrow \exists_{(s,h) \in S_n \times \{1,-1\}^n} \{h \cdot (v_{s_i})_{i=1}^n : v \in p\} = q \tag{2.7}$$

then $P/\sim$ describes the set of equivalence classes.

Table 2.1 shows the number of different unique and equivalent partitions of unity. Figure 2.1 and Fig. 2.2 illustrate a choice of partitions of unity for the two- and three-dimensional case. The number of different partitions grows very fast but is small until $n = 4$.

Table 2.1: Number of unique and equivalent partitions of unity for $n \in \mathbb{N}$.

n	1	2	3	4	5
$\lvert P \rvert$	3	8	154	89512	71319425714
$\lvert P/\sim \rvert$	2	4	15	434	> 100000

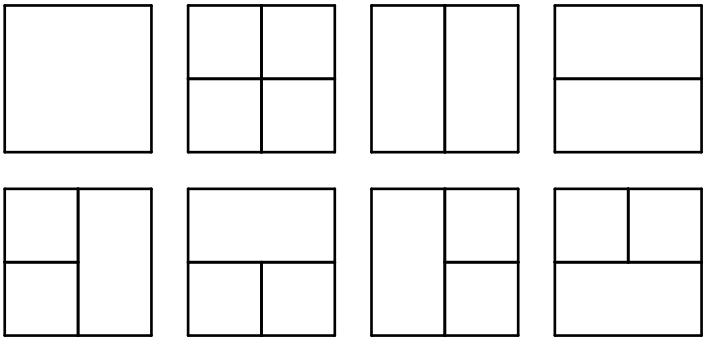


Figure 2.1: 8 unique and 4 equivalent partitions of unity exist for the two-dimensional case. The last two partitions with two elements in the first row belong to the same equivalence class and the partitions with three elements in the second row belong to the same equivalence class.

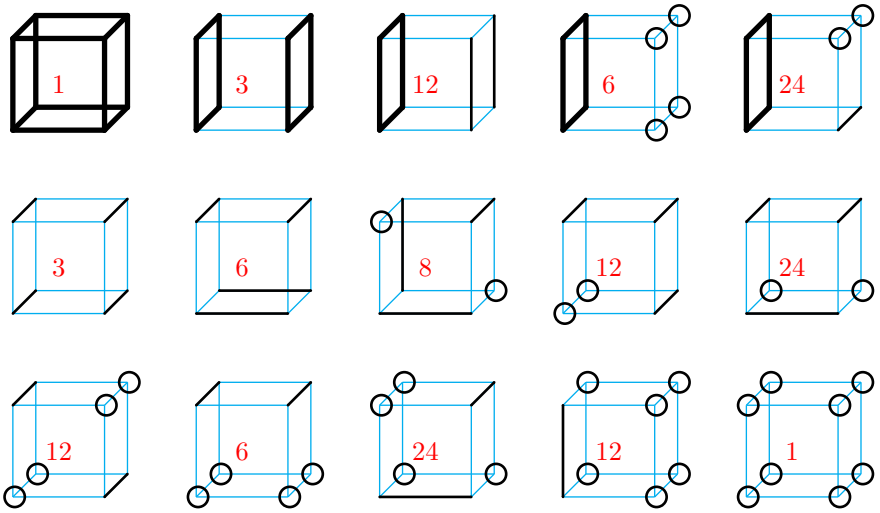


Figure 2.2: 154 unique and 15 equivalent partitions of unity exist for the three-dimensional case. For a better visibility we use bold entities representing the different parts. The bold squares represent a vector of symbols with two identity symbols, a bold line represents a vector with one identity and a bold circle represents a vector with zero identity symbols. The numbers inside the cubes denote the cardinality of the equivalence classes.

2.4 Algebraic Structures

In this section we introduce binary relations and operators. These are used to formulate algebraic structures. We introduce finite abelian groups and fields that allow to use basic calculation rules. The proofs of the following theorems can be done by checking the small finite number of elements, e.g. commutativity is implied by symmetric evaluation tables.

Definition 5. *We define a binary plus and multiplication operator*

$$+ : A \times A \to A \qquad \cdot : A \times A \to A \tag{2.8}$$

for symbols in A by the evaluation tables

$+$	L	I	R
L	R	L	I
I	L	I	R
R	I	R	L

$\cdot$	L	I	R
L	R	I	L
I	I	I	I
R	L	I	R

Let $a, b \in A$ then $a + b$ yields the neutral symbol for different complementary symbols and inverts for same complementary symbols. $a \cdot b$ yields L for different complementary symbols and R for same complementary symbols.

Theorem 1. *$(A, +, \cdot)$ is a finite field.*

- *$(A, +)$ is an abelian group with neutral element I*
- *$(\mathbf{I}, \cdot)$ is an abelian group with involution and multiplicative identity R*
- *The distributive property applies for addition and multiplication*

Definition 6. *The bold notation allows set-wise relations. Let $a, b \in A$ then*

$$a \subseteq b :\Leftrightarrow \mathbf{a} \subseteq \mathbf{b} \tag{2.9}$$
$$a \cap b :\Leftrightarrow \mathbf{a} \cap \mathbf{b} \neq \emptyset \tag{2.10}$$

In addition, we introduce the parallel and orthogonal relation defined by

$$a \parallel b :\Leftrightarrow (a = I \Leftrightarrow b = I) \tag{2.11}$$
$$a \perp b :\Leftrightarrow (a = I \Leftrightarrow b \in \mathbf{I}) \tag{2.12}$$

Vectors $v, w \in A^n$ are parallel iff identity symbols are at the same position. They are orthogonal iff identity and complementary symbols are at different positions.

Definition 7. *We use the parallel relation to define the equivalence classes*

$$\mathsf{A} := A/\parallel = \{\{I\}, \mathbf{I}\} \qquad \mathsf{A}^n := \prod_{i=1}^{n} \mathsf{A} \tag{2.13}$$

These axis aligned sets of vectors are a subset of partitions of unity, i.e. $\mathsf{A}^n \subset P$.

Corollary 1. *Let $a \in A$ and $\mathsf{a} \in \mathsf{A}$ then*

$$a/\perp = \{b \in A : a \perp b\} \in \mathsf{A} \tag{2.14}$$
$$\mathsf{a}/\perp = \{b \in A : \forall_{a \in \mathsf{a}} a \perp b\} \in \mathsf{A} \tag{2.15}$$

Definition 8. *For* A *we define the binary plus operator*

$$+ : \mathsf{A} \times \mathsf{A} \to \mathsf{A} \tag{2.16}$$

by the evaluation table

$+$	$\{I\}$	$\mathbf{I}$
$\{I\}$	$\{I\}$	$\mathbf{I}$
$\mathbf{I}$	$\mathbf{I}$	$\{I\}$

where we split vectors at unequal splits and merge at equal splits. Let $\mathsf{v}, \mathsf{d} \in \mathsf{A}^n$ *then vector-wise addition is defined by*

$$\mathsf{v} + \mathsf{d} = \prod_{i=1}^{n} \mathsf{v}_i + \mathsf{d}_i \tag{2.17}$$

where we add individual equivalence classes.

Theorem 2. $(\mathsf{A}, +)$ *is an abelian group with involution and neutral element* $\{I\}$.

Corollary 2. *Let* $\mathsf{a}, \mathsf{b} \in \mathsf{A}$ *then* $\mathsf{a}/\!\perp + \mathsf{b}/\!\perp = (\mathsf{a} + \mathsf{b})/\!\perp$.

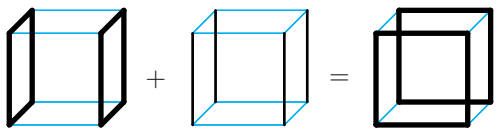
Example 1. *The two equivalence classes*

$$\mathsf{v} = \{(L, I, I), (R, I, I)\} \tag{2.18}$$
$$\mathsf{d} = \{(L, I, L), (L, I, R), (R, I, L), (R, I, R)\} \tag{2.19}$$

yield the sum

$$\mathsf{v} + \mathsf{d} = \{(I, I, L), (I, I, R)\} \tag{2.20}$$

with the graphical representation



where we merge in x*-direction and split in* z*-direction.*

2.5 Space Partitioning System

Partitioning (or embedding) of sets can be described in a recursive way. Sets of functions that are indexed by the alphabet describe how the partitioning is done. A symbol or a vector of symbols represents the choice of partitioning function. Composition of such functions can be described by words of symbols or vectors of words, respectively.

Definition 9. *The set of* ***words*** *of complementary symbols is given by the word monoid* $(\mathbf{I}^*, \cdot, I)$. *The identity symbol is the neutral element, i.e. the empty word and* $\cdot$ *is the concatenation of words. The* $*$ *operator constructs the set of all finite words. The definition is used to induce a unique representation for partitioned domains. Let* $w \in \mathbf{I}^k$ *be a word, then a minus operator for words is defined by* $-w := (-w_{n-i+1})_{i=1}^{n}$ *such that complementary symbols are inverted and the order is reversed.*

Definition 10. *Let $\mathfrak{D}$ be a set and $f_A = \{f_L, f_R, f_I\}$ be a set of functions such that $f_{a\in A} : \mathfrak{D} \to \mathfrak{D}$. Then we introduce the notation*

$$f_{w\in\mathbf{I}^k} := f_{w_k} \circ \cdots \circ f_{w_1} \tag{2.21}$$

*for the lower and the upper index to denote recursive applications, i.e. composition of functions where w is a word of symbols. A set of functions described in that way is called a **system**. A vector-wise system of functions arises from*

$$f_{v\in A^n} : \mathfrak{D}^n \to \mathfrak{D}^n \qquad \mathbf{x} \mapsto f_{v_1}(\mathbf{x}_1) \times \cdots \times f_{v_n}(\mathbf{x}_n) \tag{2.22}$$

to describe the set of functions f_{A^n}.

2.5.1 Interval Partitioning

In this work we focus on interval partitioning and use it for voxel geometries. Therefore, we define intervals and a corresponding interval partitioning system. Partitioning and embedding of intervals can be merged into a group. Figure 2.3 illustrates multi-dimensional and recursive application of symbols to a two-dimensional interval.

Definition 11. *We use the set of all intervals given by*

$$\mathfrak{B} := \{\mathfrak{b} = [\mathfrak{b}_L, \mathfrak{b}_L] : (\mathfrak{b}_L, \mathfrak{b}_R) \in \mathbb{R}^2\} \tag{2.23}$$

to define the combined system

$$\xi_A^A := \{\xi_L^L, \xi_I^L, \xi_R^L, \xi_L^I, \xi_I^I, \xi_R^I, \xi_L^R, \xi_I^R, \xi_R^R\} \tag{2.24}$$

such that $\xi_{b\in A}^{a\in A} : \mathfrak{B} \to \mathfrak{B}$ where $\xi_A^L = \{\xi_L^L, \xi_I^L, \xi_R^L\}$ denotes the system of partitioning

$$\xi_L^L(\mathfrak{b}) = \left[\mathfrak{b}_L, \frac{\mathfrak{b}_L + \mathfrak{b}_R}{2}\right] \qquad \xi_R^L(\mathfrak{b}) = \left[\frac{\mathfrak{b}_L + \mathfrak{b}_R}{2}, \mathfrak{b}_R\right] \tag{2.25}$$

with the neutral system $\xi_A^I = \{\xi_L^I, \xi_I^I, \xi_R^I\}$ which is defined by

$$\xi_I^L(\mathfrak{b}) = \xi_L^I(\mathfrak{b}) = \xi_I^I(\mathfrak{b}) = \xi_R^I(\mathfrak{b}) = \xi_I^R(\mathfrak{b}) = \mathfrak{b} \tag{2.26}$$

and $\xi_A^R = \{\xi_L^R, \xi_I^R, \xi_R^R\}$ denotes the system of embedding intervals defined by

$$\xi_L^R(\mathfrak{b}) = [2\mathfrak{b}_L - \mathfrak{b}_R, \mathfrak{b}_R] \qquad \xi_R^R(\mathfrak{b}) = [\mathfrak{b}_L, 2\mathfrak{b}_R - \mathfrak{b}_L] \tag{2.27}$$

The composition of multi-dimensional interval partitioning functions is defined by

$$\Xi := \{\xi_w^q : q \in \mathbf{I}^* \wedge w \in \mathbf{I}^{*\times n}\} \tag{2.28}$$

Theorem 3. *$(\Xi, \circ)$ is a group.*

Proof. It is sufficient to show the existence of the neutral and inverse elements. $\xi_I^{a\in\mathbf{I}} = \xi_{a\in A}^I$ is the neutral function with different notations. Let $\xi_w^q \in \Xi$ then $\xi_{-w}^{-q} \in \Xi$ is the inverse function. □

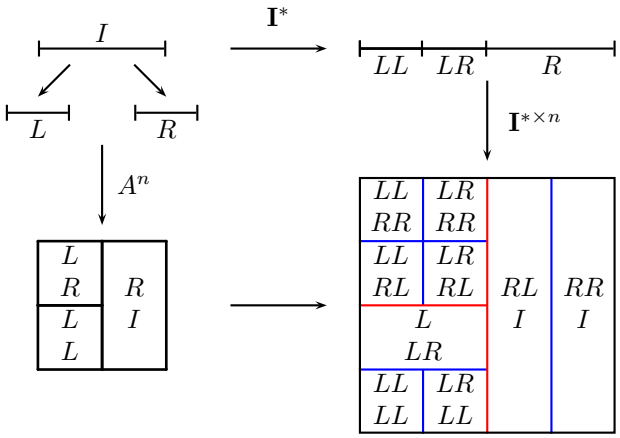


Figure 2.3: Intent of the alphabet A and its generalization to an n-dimensional domain and recursive application. L denotes left in x-direction and bottom in y-direction while R denotes right in x-direction and top in y-direction. The red lines indicate the partitioning in the first level and the blue lines indicate the partitioning in the second level.

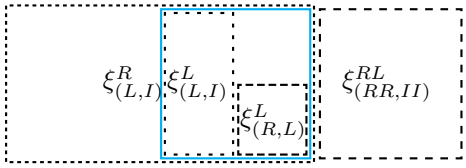


Figure 2.4: Effects of Ξ on the cyan colored domain: $\xi^R_{(L,I)}$ embeds to the left, $\xi^L_{(L,I)}$ restricts to the left, $\xi^L_{(R,L)}$ restricts to the bottom right corner and $\xi^{RL}_{(RR,II)}$ defines the right neighbor domain.

Example 2. *The vectors* $(L,L),(I,R) \in A^2$ *applied to* $\mathfrak{b} \in \mathfrak{B}^2$ *yield*

$$\xi^{LL}_{(L,LR)}(\mathfrak{b}) = \begin{pmatrix} \xi^L_L([\mathfrak{b}_{L,L}, \mathfrak{b}_{L,R}]) \\ \times \\ \xi^{LL}_{LR}([\mathfrak{b}_{R,L}, \mathfrak{b}_{R,R}]) \end{pmatrix} = \begin{pmatrix} [\mathfrak{b}_{L,L}, \frac{1}{2}\mathfrak{b}_{L,L} + \frac{1}{2}\mathfrak{b}_{L,R}] \\ \times \\ \xi^L_R([\mathfrak{b}_{R,L}, \frac{1}{2}\mathfrak{b}_{R,L} + \frac{1}{2}\mathfrak{b}_{R,R}]) \end{pmatrix} \tag{2.29}$$

$$= \begin{pmatrix} [\mathfrak{b}_{L,L}, \frac{1}{2}\mathfrak{b}_{L,L} + \frac{1}{2}\mathfrak{b}_{L,R}] \\ \times \\ [\frac{3}{4}\mathfrak{b}_{R,L} + \frac{1}{4}\mathfrak{b}_{R,R}, \frac{1}{2}\mathfrak{b}_{R,L} + \frac{1}{2}\mathfrak{b}_{R,R}] \end{pmatrix} \tag{2.30}$$

Vectors of words describe how interval partitioning is done. They can be understood as a set of instructions that point to a sub-interval with respect to an initial interval. The combination of embedding and partitioning ξ^{RL}_{vv} with $v \in A^n$ corresponds to the v-neighbored domain. Figure 2.4 illustrates the application of different vectors of words to the same initial domain.

2.5.2 Unity System

In addition to the interval partitioning system we present a minimal system of functions with domain and codomain in $\mathbf{A}$.

Definition 12. *The set of functions* $\phi_A := \{\phi_L, \phi_R, \phi_I\}$ *with*

$$\phi_{a\in A} : \mathbf{A} \to \mathbf{A} \tag{2.31}$$

defined by the table

$\mathbf{A}$	ϕ_L	ϕ_I	ϕ_R
L	**I**	**L**	**L**
I	**L**	**I**	**R**
R	**R**	**R**	**I**

is called ***unity*** *system. The composition of unity functions is defined by*

$$\Phi := \{\phi_w : w \in \mathbf{I}^{*\times n}\}.$$

Theorem 4. $(\Phi, \circ)$ *is a group with neutral function* ϕ_I*. Inverse element can be obtained by* $\neg$*.*

2.6 The Oracle Tree

In this section we introduce a formalism to define tree structures. The approach is used in a recursive way to generate trees of higher order. Different kinds of trees are described that can be used for diverse applications.

2.6.1 Oracles

We define tree structures by structural induction with a function that maps the current location, i.e. the node to a set of succeeding edges. These function are called oracles and are motivated by the memory layout of trees inside the computational memory.

Definition 13. *Let* $(X, +)$ *be a monoid with the generator set* $0 \notin T \subseteq X$*. The generator set is used to define the generator function*

$$\mho : X \to \mathcal{P}(T) = \{U : U \subseteq T\} \tag{2.32}$$

where the elements of X *are mapped to a subset of the generator set (*$\mathcal{P}$ *denotes the power set). Then we define the tree that is constructed with respect to* $\mho$ *by*

$$\mathcal{G} = (\mathcal{X}, \mathcal{E}) \qquad \mathcal{X} \subseteq X \qquad \mathcal{E} \subseteq \mathcal{X} \times \mathcal{X} \tag{2.33}$$

such that there exists a root node $0 \in \mathcal{X}$*. We construct a tree by structural induction such that*

$$x \in \mathcal{X} \Rightarrow \forall_{t\in\mho(x)} x + t \in \mathcal{X} \wedge (x, x + t) \in \mathcal{E}. \tag{2.34}$$

We also introduce the notation $\mathcal{X}(\mho)$ *and* $\mathcal{E}(\mho)$ *to denote nodes and edges, respectively. The generator function* $\mho$ *generates a tree and is called an* ***oracle*** *if and only if every node has exactly one predecessor except the root node, i.e.*

$$(y, x) \in \mathcal{E}(\mho) \Rightarrow 0 \neq x \tag{2.35}$$

$$0 \neq x \in \mathcal{X}(\mho) \Rightarrow \exists!(y, x) \in \mathcal{E}(\mho). \tag{2.36}$$

The leafs of the tree are denoted by $\mathcal{C}$ *or* $\mathcal{C}(\mho)$*, respectively.*

We use the definition of oracles to construct trees using the symbols in A or vectors of symbols, respectively. An example for the general case is shown in Fig. 2.5.

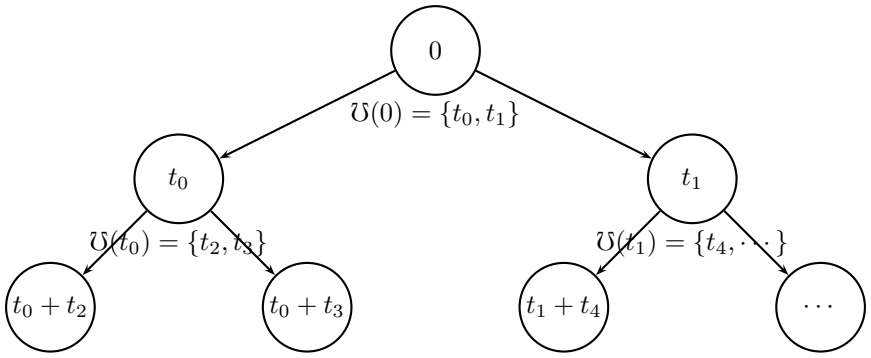


Figure 2.5: Construction of an oracle-tree. The oracle function returns the edges to succeeding nodes. The nodes are defined by adding preceding edges.

2.6.2 Ternary Oracle

In many situations we want to be able to deal with arbitrary sets of vectors, e.g. partitions of unity or overlapping sets of vectors to store data. The number of different vectors and the number of sets of vectors is determined by the dimension. We introduce the ternary oracle and the range oracle that are suited to efficiently represent and access arbitrary subsets of A^n.

Definition 14. *A **ternary oracle** Υ is defined by the word monoid $(A^*, \cdot, \epsilon)$ such that*

$$\Upsilon : A^* \to \mathcal{P}(A) \tag{2.37}$$

where all leafs are at level n, i.e.

$$\forall v \in \mathcal{X}(\Upsilon) \quad v \in A^n \Rightarrow \Upsilon(v) = \emptyset \tag{2.38}$$

The set of all ternary oracles is denoted by $\boldsymbol{\Upsilon}$.

The ternary oracle is used to describe subsets of A^n efficiently. Since the number of different partitions of unity is small until $n = 4$ (see Table 2.1) we use a look-up table in the implementation to eliminate any overhead in computational time and memory. An example of a ternary tree with three levels is shown in Fig. 2.6.

Definition 15. *Let $j, k \in \mathbb{N}$ with $j \leq k \leq n$ be the minimum and maximum number of identities then we define the **range-oracle** $\Upsilon_j^k \in \boldsymbol{\Upsilon}$ by*

$$\Upsilon_j^k : A^* \to \mathcal{P}(A) \tag{2.39}$$

$$v \in A^m \mapsto \{L, R : n - m > j - |v|\} \cup \{I : |v| < k\} \tag{2.40}$$

where we use the count operator for identity symbols

$$|v| := |\{i \in \mathbb{N} : v_i = I\}| \tag{2.41}$$

The range-oracles are a subset of the ternary oracle and the vectors denoted by

$$V_j^k := \{v \in A^n : j \leq |v| \leq k\} \subset \mathcal{X}(\Upsilon_j^k)$$

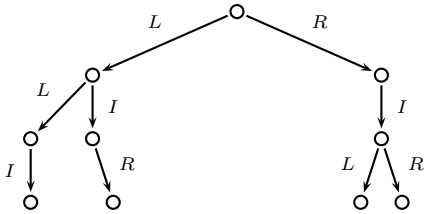


Figure 2.6: Example of a ternary-tree encoding vectors in A^3.

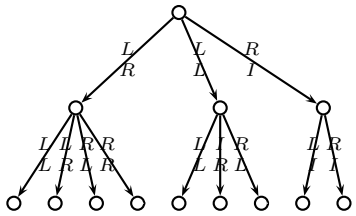


Figure 2.7: Example of a vector-tree encoding vectors of words in $\mathbf{I}^{*\times 2}$.

are the corresponding leaf nodes. The tree constructed by Υ_j^k *is called range-tree and allows the representation of vertices, edges, faces or other higher dimensional entities in a cuboid. We use the abbreviation* $V_j = V_j^j$ *for sets of vectors with the same number of identities.*

Since the structure of a range tree is fixed and known at compilation time we can eliminate any overhead similar to the ternary tree. An important application of the range-tree is store neighborhoods of nodes such that we can access a neighbor by a vector $v \in A^n$.

2.6.3 LIR-Oracle

In this section we describe a tree structure called vector-tree that uses vectors of symbols for construction of a tree that represents a set of vectors of words. By restriction to partitions of unity we finally get the LIR-tree.

Definition 16. *Let* $n \in \mathbb{N}$ *then a* ***vector-oracle*** Ω *is defined by the vector of words monoid* $(\mathbf{I}^{*\times n}, \cdot, I)$ *where* $\cdot$ *is the vector-wise concatenation of words and* I *the vector of empty words such that*

$$\Omega : \mathbf{I}^{*\times n} \to \mathcal{P}(A^n).$$

The set of all vector-oracles is denoted by $\mathbf{\Omega}$.

Definition 17. *By restricting the set of successors it is possible to introduce the partition of unity property. Therefore, the* ***LIR-oracle*** *is defined by*

$$\Omega_P : \mathbf{I}^{*\times n} \to P$$

such that Ω_P *constructs a vector tree and satisfies the partition of unity property in each node. The set of all LIR-oracles is denoted by* $\mathbf{\Omega}_P$. *The trees that can be constructed by LIR-oracles are called* ***LIR-trees****.*

An example of a small vector-tree with two levels is shown in Fig. 2.7. A larger example of a LIR-tree with a corresponding domain can be seen in Fig. 2.8. This domain is partitioned using the interval partitioning system.

In our basic implementation we use the memory layout shown in Fig. 2.9 for the nodes in a LIR-tree. A node is split into a type and an index. The type can either point into a look-up table for partitions of unity or indicates a material. If the type indicates a material then the cell is a leaf and the index points to the corresponding data entity. In the other case the index points to the first child cell. The number of remaining child nodes can be determined by the type.

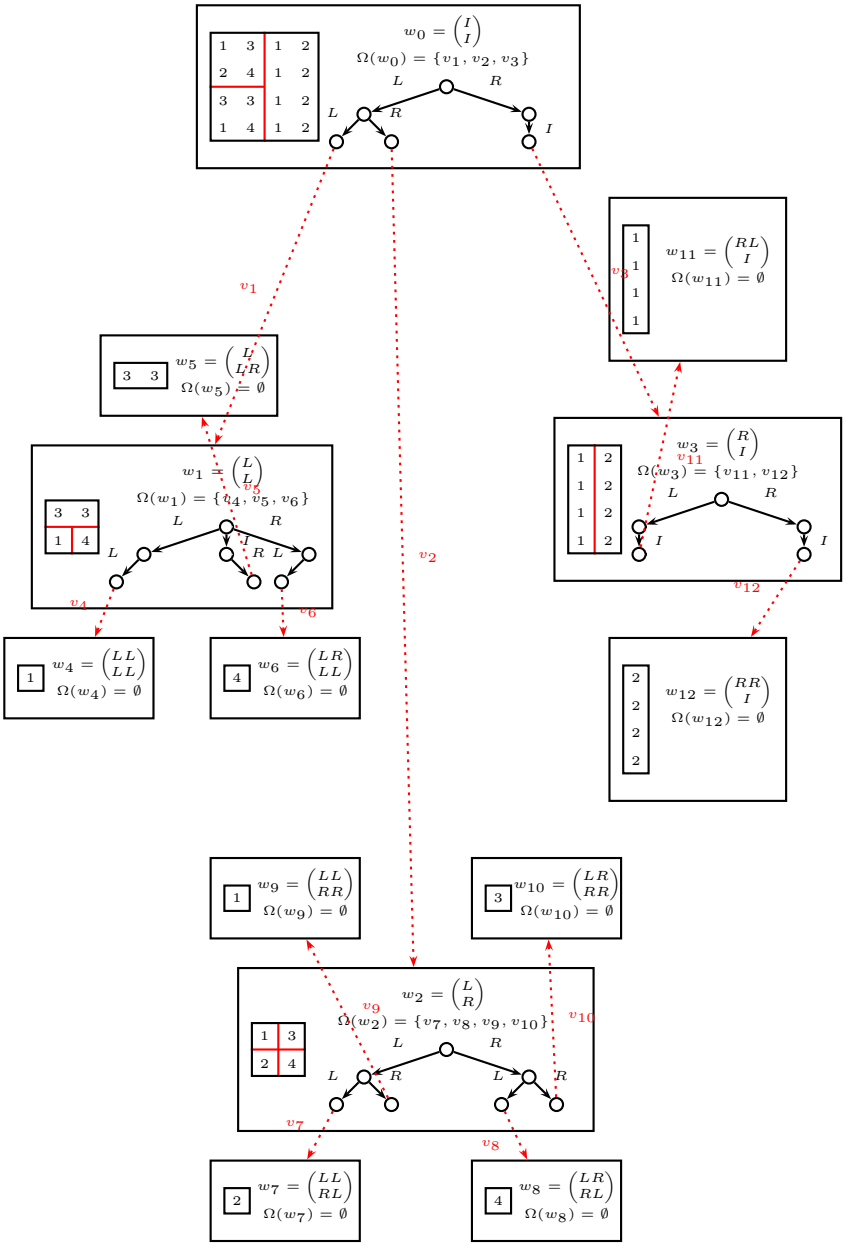


Figure 2.8: Example of a LIR-tree that is defined by the set of nodes $\mathcal{X}(\Omega_P) = \{w_0, \cdots, w_{12}\}$ and the set of links $\mathcal{E}(\Omega_P) = \{(w_0, w_1), \cdots, (w_3, w_{12})\}$. Note that the ternary trees are realized by a look-up table and do not occupy memory. The numbers inside the rectangles illustrate the domain and the corresponding data that is partitioned. The red lines show how the domains are partitioned and the red dotted lines show the links between nodes.

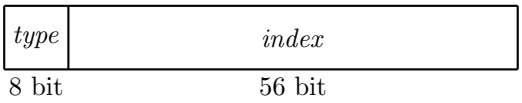


Figure 2.9: Memory layout of a cell in 64 bit. The type indicates either a partition of unity or a leaf. The index either points to the start of the child nodes or a data entity.

2.7 Partition Determination

Definition 18. *The one-dimensional edges can be represented by the range tree* $\Upsilon_1^1 \in \boldsymbol{\Upsilon}$ *with the set of edge vectors* $E := V_1$*. We assign a binary value to each edge, i.e.* $S := \mathbb{B}^{n \cdot 2^{n-1}}$ *with* $\mathbb{B} = \{0, 1\}$*. We use* S *to determine and modify appropriate partitions to build a tree. A one represents that the corresponding edge has to be split. A zero represents that an edge may or may not be split.*

Let $p \in P$ be a partition of unity and $s \in S$. If no vector of p contains a split then p is conform to s. We define a function ψ that takes a partition and return a zero where ever an edge is a subset of a vector of p:

$$\psi : P \to S \qquad p \mapsto \left(\begin{cases} 0 & \text{if } \exists_{v \in p} \mathbf{e} \subseteq \mathbf{v} \\ 1 & \text{else} \end{cases} \right)_{e \in E}. \tag{2.42}$$

with $E := V_{n-1}^{n-1}$. The inverse function of ψ is defined by the ψ^{-1} function that takes a $s \in S$ and returns a partition $p \in P$ that is conform to s and has a minimum number of vectors.

$$\psi^{-1} : S \to P \qquad s \mapsto \arg\min\{|p| : p \in P \wedge \neg\psi(p) \wedge s = 0\} \tag{2.43}$$

where $\wedge$ and $\neg$ denote bit-wise operators. Since the minimum partition is not unique, ψ^{-1} returns an arbitrary conforming partition that is minimal.

Similar to the ternary- and range-tree we use a look-up table for the evaluation of ψ and ψ^{-1} in our implementation. That way we get a $\mathcal{O}(1)$ runtime-complexity.

2.8 Neighborhood Retrieval

Most tree processing algorithms want to have access to its neighbor nodes. Therefore we provide an efficient way of neighborhood retrieval such that the time complexity is independent with respect to the depth of the tree. In this section, we use the range tree and introduce a traversal operation to determine successive neighborhoods of nodes.

A neighbor of an interval decomposition function in the direction given by $v \in A^n$ can be defined by the composition ξ_{vv}^{RL}. Let $\Omega \in \boldsymbol{\Omega}_P$ be a LIR-tree, $k \in \mathbb{N}$ be a number of steps and $(w^i \in A^n)_{i=1}^k$ an ordered set of vectors such that $\forall_{j=1}^k \sum_{i=1}^j w^i \in \mathcal{X}(\Omega)$. We assume that an algorithm travelled k steps from I to $\sum_{i=1}^k w^i$ through nodes of Ω. Let $0 < r \in \mathbb{N}$ then the algorithm stores the vector of words to the r to n dimensional neighbors of $\sum_{i=1}^j w^i$ in a set of range trees. These neighbors are defined by

$$w_v^j = \max\{w \in \mathcal{X}(\Omega) : \xi_{vv}^{RL} \circ \xi_{w_I^j}^L \subseteq \xi_w^L\}$$

for each $v \in V_r^n$. Let $w^{k+1} \in \Omega(w_I^k)$ be the next step of the algorithm then $w_I^{k+1} = w_I^k w^{k+1}$ denotes the next location in the tree. The succeeding neighbors of w_I^{k+1} are defined by

$$w_v^{k+1} = \max\{w \in \mathcal{X}(\Omega) : \xi_{vv}^{RL} \circ \xi_{w_I^{k+1}}^{L} \subseteq \xi_w^L\} \tag{2.44}$$

Since $w_d^k \in \mathcal{X}(\Omega)$ with the corresponding neighbor vector $d = -v^2(v + w^{k+1})$ we can precise the upper search bound by

$$w_v^{k+1} = \max\{w \in \mathcal{X}(\Omega) : \xi_{vv}^{RL} \circ \xi_{w_I^{k+1}}^{L} \subseteq \xi_w^L \subseteq \xi_{w_d^k}^{L}\} \tag{2.45}$$

Thus, we can determine w_v^{k+1} by travelling through the tree from w_d^k.

This way we can retrieve neighbors by using the neighbors of the parent cell and are independent with respect to the global depth of the tree, i.e. we do not have to climb up and down the tree. If the neighborhood retrieval consumes a considerable amount of computational time it is possible to store pre-determined neighborhoods to improve the performance.

2.9 Input Function

In this section we describe an oracle that constructs a (non-degenerative) tree until a given threshold for partitioning is reached. It is also possible to use overlapping sets to increase the number of cells at interfaces.

Let $t \in \mathbb{R}^n$ be a vector of real numbers that denote the limit for partitioning. Then we define the function

$$\vartheta : \mathfrak{B} \to S \qquad \mathfrak{b} \mapsto (e_i = I \Rightarrow |\mathfrak{b}_i| > t_i)_{e \in E} \tag{2.46}$$

that uses an interval and determines possible edges which are candidates to be split with respect to t.

The basic idea of the actual input function is to split edges where the corresponding segments of voxels contain different values. A segment of voxels is a finite straight chain of voxels that is used for analysis, see Fig. 2.10. We assume that $\omega : \mathbb{R}^n \to \mathfrak{D}$ is a given function that returns the type of material at each $x \in \mathbb{R}^n$ where $\mathfrak{D}$ denotes the domain of material. Then the next part of the input function is defined by

$$\theta : \mathfrak{B} \to S \qquad \mathfrak{b} \mapsto \left(\exists x, y \in \mathfrak{b} \begin{cases} \omega(x) \neq \omega(y) \\ x_i \neq y_i \Rightarrow e_i = I \end{cases} \right)_{e \in E} \tag{2.47}$$

Let $k \in \mathbb{R}$ be a range that describes the overlap of analysis. The final input function is a composition of ϑ and θ defined by

$$\zeta^k : \mathfrak{B} \to S \qquad \mathfrak{b} \mapsto \vartheta(\mathfrak{b}) \wedge \theta^k([\mathfrak{b}_L - k, \mathfrak{b}_R + k]) \tag{2.48}$$

We also suggest multiplication of the parameter k. The corresponding tree is constructed by the oracle

$$\Omega_P(w) := \psi^{-1}(\zeta^k(\xi_w^L(\mathfrak{b})))$$

where we assume that $\mathfrak{b} \in \mathfrak{B}$ is the initial domain we want to partition.

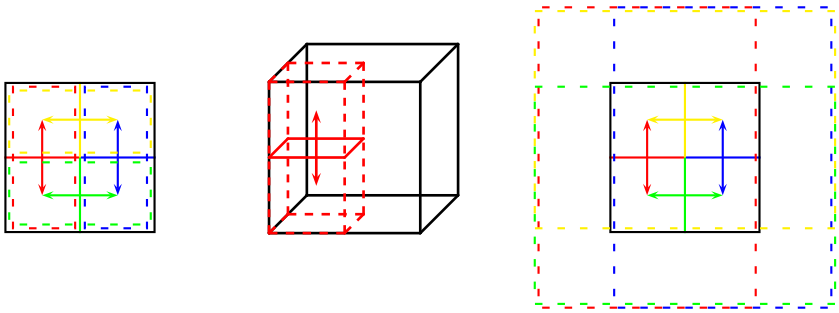


Figure 2.10: The colored regions illustrate the domain and direction of analysis for the edges. In the left and center case we use ζ^0 and in the right case we use ζ^1. A color component shows the edges and its domains of analysis. The direction of the arrow corresponds to i in Eqn. 2.46.

2.10 Iterative Refinement

Iterative tree refinement can be described as a sequence of oracles $(\Omega_i \in \mathbf{\Omega}_P)_{i\in\mathbb{N}}$. Assume we are given a sequence of modifier functions $\iota_i : \mathcal{C}_i \to S$ that return splits for each leaf of the LIR-tree described by Ω_i. There are two ways of iterative tree refinement: degenerative and non-degenerative. Therefore, we have to describe both ways that can be used for different requirements.

Definition 19 (Degenerative Refinement)**.** *We define the degenerative sequence of oracles*

$$\Omega_{i+1}(w) = \begin{cases} \psi^{-1}(\iota_i(w)) & \text{if } w \in \mathcal{C} \wedge \iota_i(w) \neq 0 \\ \Omega_i(w) & \text{else} \end{cases} \tag{2.49}$$

such that $\Omega_i \in \mathbf{\Omega}_P$. We allow modifications only at the leafs of trees.

Degenerative refinement is recommended for geometry analysis and subdivision splines where numerical properties are non-critical. Non-degenerative refinement is important for increasing numerical stability but requires a more complex set-up. First, we describe how splits are merge within parent nodes.

Definition 20. *Let $v \in A^n$ then we define merge of splits by*

$$\sigma_v : S \to S \qquad s \mapsto \left(\bigvee_{d \| e \wedge v \supseteq e} s_d \right)_{e\in E} \tag{2.50}$$

such that we merge parallel edges that are subset of v.

We use merging of splits to generalize arbitrary modifier functions by their child sets of vectors.

Definition 21 (Generalized Modifier Function)**.** *Let $\iota : \mathcal{C} \to S$ be a modifier function then we use recursion to define the generalized modifier function*

$$\widehat{\iota} : \mathcal{X} \to S \qquad w \mapsto \begin{cases} \bigvee_{v\in\Omega(w)} \sigma_v(\iota(wv)) & \text{if } \Omega(w) \neq \emptyset \\ \iota(w) & \text{else} \end{cases} \tag{2.51}$$

The domain is extended from leafs to nodes by merging embedded splits of children.

Finally, we are able to describe non-degenerative refinement by using generalized modifier functions.

Definition 22 (Non-Degenerative Refinement)**.** *Let $\widehat{\iota}_i : \mathcal{C}_i \to S$ be a sequence of generalized modifier functions then we define the sequence of non-degenerative oracles by*

$$\Omega_{i+1}(w) = \begin{cases} \Omega_i(w) & \text{if } w \in \mathcal{C}_i \wedge \widehat{\iota}_i(w) = 0 \\ \psi^{-1}(\widehat{\iota}_i(w)) & \text{if } w \in \mathcal{C}_i \wedge \widehat{\iota}_i(w) \neq 0 \\ \Omega_i(ldr) & \text{else} \end{cases} \tag{2.52}$$

The second cases occurs at the leafs describes by Ω_i where $\widehat{\iota}_i$ returns a split. The following sub-trees are considered by the third case. If $w \notin \mathcal{C}_i$ then we can describe the word by $w = lvr$ with $l \in \mathcal{C}_i$, $v \in A^n$ and $r \in \mathbf{I}^{*\times n}$ such that $\widehat{\iota}_i(l) \neq 0$ and $v \in \psi^{-1}(\widehat{\iota}_i(l))$. The corresponding location in the predecessor tree can be described by $ldr \in \mathcal{C}_i$ with $d \in \Omega_i(l)$ such that $v \subseteq d$.

Let $\Omega_P(w) = \psi^{-1}(\zeta^k(\xi_w^L(\mathfrak{b}))) \in \mathbf{\Omega}_P$ be an oracle defined by the geometry analysis described in Sec. 2.9 then we can build that tree by degenerative refinement with the sequence of modifier functions $\iota_i(w) = \zeta^k(\xi_w^L(\mathfrak{b}))$. There exists a final iteration $m \in \mathbb{N}$ such that $\Omega_k = \Omega_P$ for each $m < k \in \mathbb{N}$.

2.11 Results

In the last section we show the results of experiments where we applied the LIR-tree to a complex dataset and compare it to the Octree and KD-tree. In our research we focus on voxel geometries, especially on fiber geometries, see Fig. 2.11. Therefore we chose a complex generated fiberglass dataset for our experiments. We restrict to a three-dimensional context since the LIR-tree is designed for higher dimensional partitioning.

2.11.1 Generated Fiberglass

For our experiments we generated a very complex fiberglass dataset with the GeoDict software suite [35]. The dataset is an approximation to a High-Efficiency Particulate Air (HEPA) filter medium. HEPA filters are composed of randomly arranged fibers on the micrometer scale.

In the highest resolution the dataset is represented by $400 \times 400 \times 4000$ voxels that can either be empty or solid. The fibers have a solid-volume-fraction of 8%. In general they have an anisotropic spatial orientation that is isotropic in (x, y)-plane and layered in z-direction, see Fig. 2.12. A two-dimensional cut through the dataset is shown in Fig. 2.13. We use the input function with ζ^0, see Fig. 2.14. An overlapping input function is used in Fig. 2.15 to illustrate the behavior. A three-dimensional view of the tree structure is shown in Fig. 2.16. The partitioning is done by ξ_v^L with $v \in A^n$ where we modified the center to meet the voxel boundaries. In addition, we degenerate the tree first until a cubic spatial domain is given.

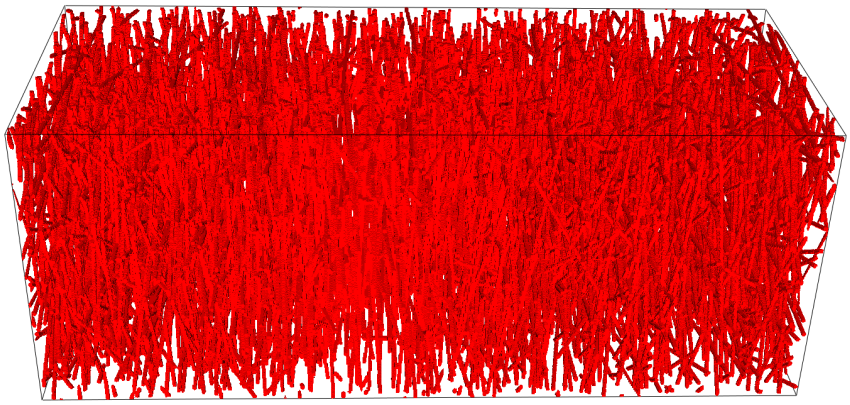


Figure 2.11: 3d fibrous material used as porous media model and as composite material models proposed in [31] and implemented in the GeoDict software [35].

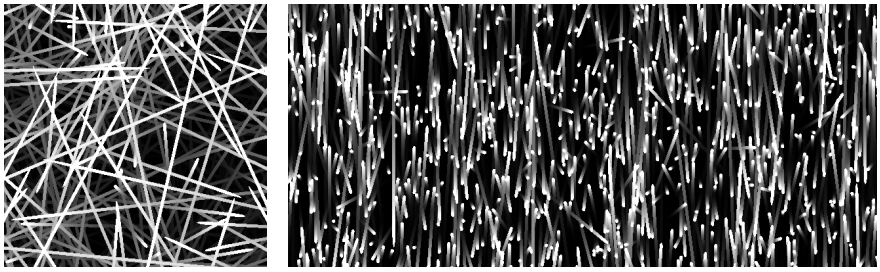


Figure 2.12: 2D views of a fiberglass medium from [35]: (x,y)-view, (y,z)-view.

2.11.2 Tree Comparison

The Octree and KD-tree are special LIR-trees generated by restriction of P by

$$P_{oct} = \{\emptyset, \mathbf{I}^3\} \qquad P^i_{kd} = \{\emptyset, \{v \in A^3 : v_i \in \mathbf{I} \wedge v_{j\neq i} = I\}\} \tag{2.53}$$

Figure 2.17 shows that the general LIR-tree has fewer nodes for each voxellength compared to the Octree and KD-tree. In fact, the LIR-tree has the interior cell complexity of the Octree and the leaf cell complexity of the KD-tree. Numerous experiments showed that on average the LIR-tree has at least two times fewer nodes compared to other trees in the three-dimensional case. We are interested in efficient numerically suited representations rather then pure binary data compression. Hence, the LIR-tree is an efficient method for that purpose.

2.11.3 Number of Children

In the next experiment we investigate the distribution of the number of children. Figure 2.18 shows the distribution for the fiberglass example. It turns out that in most cases a partitioned cell has two, three or four children. The cases of two and three children are changing their places with respect to the voxel length due

Figure 2.13: Cut through fiberglass.

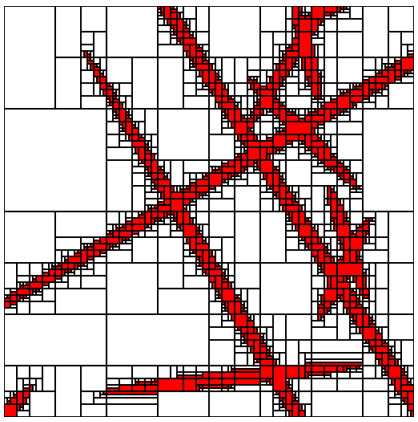


Figure 2.14: LIR-tree with ζ^0.

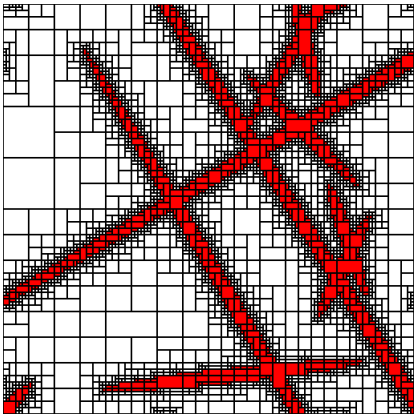


Figure 2.15: LIR-tree with ζ^1.

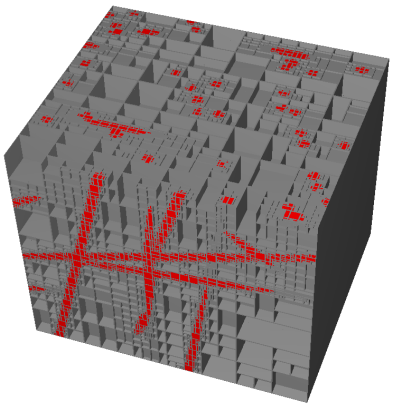


Figure 2.16: 3D-View of the LIR-tree.

to the non-cubic dataset. On average they make up 25% of the partitions. A five children partition occurs in 10% of the cases while the six, seven and eight children partitions occur in up to 4% of the cases.

2.11.4 Error Analysis

In our last experiment we used the highest resolution of the fiberglass as reference and compared a lower resolved LIR-tree with respect to the number of incorrect voxels and volume defect. Figure 2.18 shows that the relative error has linear correlation to the voxellength. But it also shows that the relative error has a quadratic correlation to the number of leafs. Hence, it makes sense to use the number of leafs to investigate convergence orders instead of the voxellength. The error of the volume is lower compared to the total error. That is an important property in situations where the same mass leads to same behaviors.

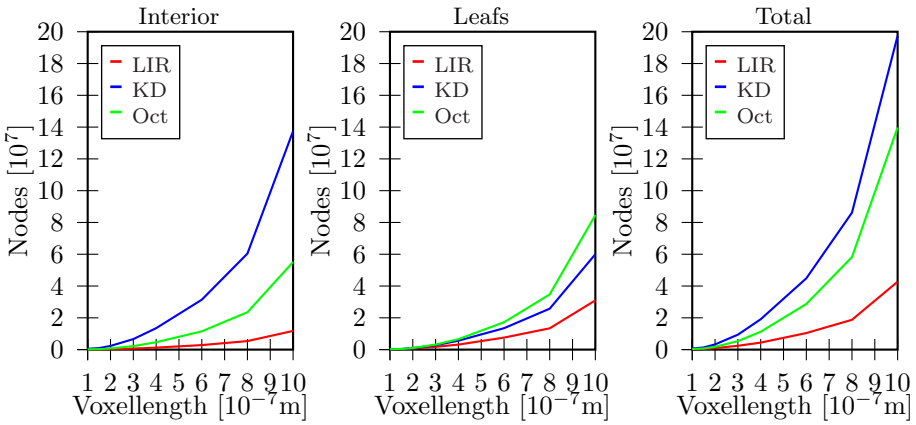


Figure 2.17: Number of nodes used by the LIR-tree, KD-tree and Octree. The interior, leaf and total number of nodes are considered.

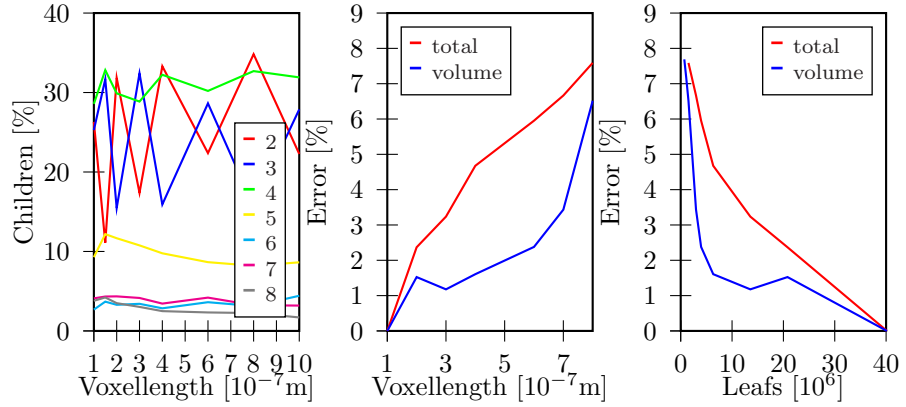


Figure 2.18: Left: Number of children distribution. Center and right: Incorrect voxel values and volume defect with respect to voxellength and number of leafs.

Chapter 3

Solving the Stokes Equation

Abstract

We introduce a novel approach of variable arrangement for pressure and velocity to solve the Stokes equations. The basic idea of our method is to arrange variables in a way such that each cell is able to satisfy a given physical law independently from its neighbor cells. This is done by splitting velocity values to a left and right converging component. For each cell we can set up a small linear system that describes the momentum and mass conservation equations. Theses formulation allows to use the Gauß-Seidel algorithm to solve the global linear system. The LIR-tree is used for spatial partitioning of the geometry and provides a proper initial guess. In addition, we introduce a method that uses the actual velocity field to refine the tree and improve the numerical accuracy where it is needed. The discretization converges to the analytical solution with respect to grid refinement. Our method is compared to other state of the art solvers. We conclude a high strength in computational time and memory for high porosity geometries and a high strength in memory requirement for low porosity geometries.

3.1 Governing Equations

The Navier-Stokes equations describe the motion of newtonian fluids. They are named after Claude Louis Henri Navier and George Gabriel Stokes. More specifically, the Navier Stokes equations describe the momentum conservation. Together with the mass conservation, energy conservation and state equation we get a system of non-linear partial differential equations of second order. They are the fundamental base of computational fluid dynamics.

3.1.1 Stokes Equations

The Stokes equations are a simplification of the general Navier-Stokes equations. They are used when the fluid velocity is very slow, i.e. the Reynolds number is low ($Re \ll 1$).

The influence of temperature is neglected and a constant density is assumed. Therefore, we consider only the pressure and velocity of a fluid. We also restrict to the steady state case where no unsteady acceleration is present. The convective acceleration is also omitted in the formulation.

A typical application area of the Stokes equations is geophysics. It is also used in the oil industry to simulate the motion of oil in porous media. Another important application area is research of filter media. In that area people are interested in homogenized material constants, e.g. permeability. The permeability tensor can be determined by applying axis aligned pressure drops in all spatial directions.

Definition 23 (Stokes Equation)**.** *Let $\Omega \subset \mathbb{R}^n$ be a domain such that all corresponding leaf nodes are either subset of Ω (computational domain) or subset of $-\Omega$ (solid domain), i.e.*

$$\forall_{w \in \mathcal{X}(\Omega)} \xi_w(\mathfrak{D}) \subseteq \Omega \vee \xi_w(\mathfrak{D}) \subseteq -\Omega \tag{3.1}$$

where $\Omega \subseteq \mathfrak{b} \in \mathfrak{B}$ denotes the embedding box. Then we define the two variables

$$u : \Omega \to \mathbb{R}^n \quad \textit{velocity} \tag{3.2}$$
$$p : \Omega \to \mathbb{R} \quad \textit{pressure} \tag{3.3}$$

The Stokes equation is then defined by

$$\eta \nabla^2 u - \nabla p + f = 0 \quad \textit{momentum conservation} \tag{3.4}$$
$$\nabla \cdot u = 0 \quad \textit{mass conservation} \tag{3.5}$$
$$u_{|\partial\Omega} = 0 \quad \textit{no slip condition} \tag{3.6}$$

where $\eta \in \mathbb{R}_+$ describes the viscosity constant. In the following sections we use the abbreviation $\xi_w := \xi_w(\mathfrak{D})$.

3.1.2 Effective Permeability

Engineers and researcher are interested in homogenized material constants and laws. The linear relation between velocity and pressure through porous medium is described by Darcy's Law.

Definition 24 (Darcy's Law)**.** *Darcy's law describes the flow of a fluid substance through a porous medium. It is defined by*

$$\eta \cdot u = -K \cdot \nabla p \tag{3.7}$$

where K is a tensor. The law was introduced by Henry Darcy based on experiments. There are three possible configurations:

- *Isotropic - the tensor is diagonal with equal entries*
- *Orthotropic - the tensor is diagonal but with different entries*
- *Anisotropic - the tensor is symmetric such that spatial directions are coupled*

The effective permeability of a representative volume element in the three-dimensional case is described by the tensor

$$K = \begin{pmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{pmatrix} \in \mathbb{R}^{3\times 3} \tag{3.8}$$

The tensor K can be determined by the application of three axis aligned pressure drops and evaluation of the corresponding mean velocity, e.g. the coefficients in x-direction are given by

$$\eta \cdot u = \begin{pmatrix} \mathbf{K_{11}} & K_{12} & K_{13} \\ \mathbf{K_{21}} & K_{22} & K_{23} \\ \mathbf{K_{31}} & K_{32} & K_{33} \end{pmatrix} \cdot \begin{pmatrix} f \\ 0 \\ 0 \end{pmatrix} \tag{3.9}$$

There are two ways to determine the mean velocity for a given pressure drop. Let $k \in \mathbb{N}$ be the number of voxels and v be the voxellength in x-direction. Then we can apply a volume force defined by $f = \frac{p_{\text{drop}}}{k \cdot v}$. The second way is to add or subtract the pressure drop p_{drop} whenever we access a periodic neighbor in left or right direction, respectively.

3.2 Related Work

The Stokes equations can be solved in numerous ways. There exist analytical solutions for slow stokes flows around periodic arrays of spheres with different alignments. These can be approximated by a series described in [30] for the two- and three-dimensional case. We use the tree-dimensional solution for simple cubic arrays of spheres to verify our algorithm.

For arbitrary complex geometries we have to use numerical method. We refer to three numerical methods to solve the Stokes equations. That are

- Semi implicit methods for pressure linked equations
- Explicit-Jump immersed interface method
- Lattice-Boltzmann methods

The Semi implicit methods for pressure linked equations (SIMPLE) are used in computational fluid dynamics to solve the Navier-Stokes equations. For the Stokes equation the method is decomposed into the following steps: guess a pressure field, solve the momentum conservation equation, correct the pressure field with respect to the mass conservation equation and correct the velocity field. These steps are iterated until a specified termination condition is reached. Relaxation parameters for velocity and pressure have to be given that can either be used to accelerate the convergence or to stabilize divergent behavior. The method can used for complex geometries due to the low memory requirement. There exist several enhancement of the original algorithm, e.g. SIMPLE-Corrected (SIMPLEC), SIMPLE-Revised (SIMPLER), or SIMPLE-FFT.

Explicit-Jump immersed interface method (EJ) is a very fast finite difference method to solve partial differential equations [37]. It can be used to solve the Stokes equations on binarized microstructures. Velocity and pressure values are considered in all cells of a regular grid even in the solid phase. Additional

fictitous forces are applied on the interfaces to satisfy the no-slip boundary condition. The fast convergence rate is achieved by using the Fast Fourier Transformation (FFT) on four Poisson problems. The memory requirements are very low and also depend on the surface area of the interface. The method is designed for geometries with high porosity. An implementation of the Explicit-Jump method with simplified boundary conditions is shown in [36].

The Lattice Boltzmann methods (LB) can also be used to solve the Navier-Stokes equations. It is based on molecular dynamics and solves the Lattice-Boltzmann equation rather than solving Navier-Stokes equations itself. The Lattice-Boltzmann equation describes interactions between particles. The flow is simulated with collision models such as Bhatnagar-Gross-Krook. The method allows to include different models to simulate different kinds of physical behaviors, e.g. thermal properties in fluids. The method has low memory and computational requirement per cell.

We do not use the referred methods on the LIR-tree since they are designed for regular grids and not suited for adaptive discretizations. Another reason for that decision is the expensive neighborhood retrieval that has to be done multiple times per iteration. Thus, we introduce a novel approach of solving the Stokes equations with block linear systems such that the number of neighborhood retrivals is minimized.

3.3 Conservation Law based Cellular Structure

In this section we describe a novel approach of variable arrangement to solve partial differential equations. Since we restrict our work to flow simulations we depict the velocity and pressure variable arrangement for LIR-trees. We also provide a discretization for the Laplacian, gradient and divergence operator term as well.

The fundamental idea of variable arrangement in a LIR-tree is that each cell has to be able to satisfy a given (physical) law, e.g. the Stokes equations. For that equations a cell has to be able to satisfy the conserve momentum and mass. The crucial part of that given equations is the mass conservation. The staggered and the collocated variable arrangement is not able to satisfy the mass conservation for each cell independently. Therefore, a single cell in a tree stores two velocity values for each direction at the face and a pressure value in the center. As a consequence a regular grid would have two velocity values for each location such that we have a left converging and a right converging value. In fact, we use the equivalence classes in A^n to describe variable arrangement.

3.3.1 Cells

The leafs of a LIR-tree are called cells and they are the places of computation. Therefore, we introduce a notation for the set of cells and its corresponding sets of neighbor cells.

Definition 25 (Cells)**.** *Data is stored in the cells of a LIR-tree. Let* $\Omega \in \mathbf{\Omega}$ *then* ***cells*** *are nodes* $w \in \mathcal{X}(\Omega)$ *such that* $\Omega(w) = \emptyset$*, i.e. the set of children is empty. In addition, the domain of the cell has to be inside non-solid. The set*

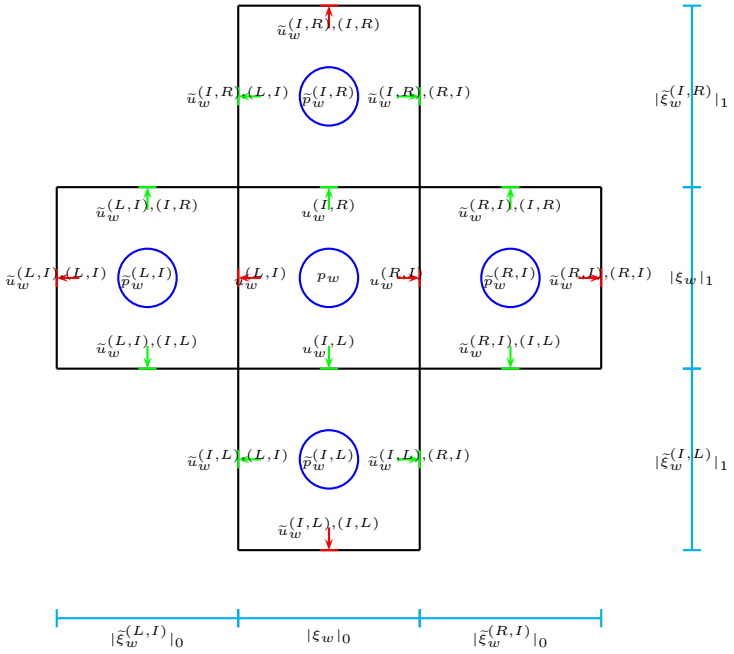


Figure 3.1: Variable arrangement for pressure and velocity.

of cells for Ω *is denoted by*

$$\mathcal{C}(\Omega) := \{w \in \mathcal{X}(\Omega) : \Omega(w) = \emptyset \wedge \xi_w \subseteq \Omega\} \tag{3.10}$$

In the following we also use the abbreviation $\mathcal{C} := \mathcal{C}(\Omega)$.

Definition 26 (Neighborhood Cells)**.** *Let* $\Omega \in \mathbf{\Omega}$ *and* $w \in \mathcal{C}$ *a cell then the corresponding set of neighbor cells in* $v \in A^n$ *direction is defined by*

$$\mathcal{C}_w^v := \{q \in \mathcal{C} : \forall_{i \in \mathbb{N}} \xi_{(-v)^i}^{L^i} \circ \xi_{vv}^{RL} \circ \xi_w \cap \xi_q \neq \emptyset\} \tag{3.11}$$

In the following sections we also use the abbreviation $D := V_{n-1}$ *and* $\mathsf{D} := V_{n-1}/\!\parallel$ *for the set of parallel face vectors. In addition, we define the sets of left and right neighbors by*

$$\mathcal{C}_w^L := \bigcup_{v \in D^L} \mathcal{C}_w^v \qquad \mathcal{C}_w := \bigcup_{v \in D} \mathcal{C}_w^v \qquad \mathcal{C}_w^R := \bigcup_{v \in D^R} \mathcal{C}_w^v \tag{3.12}$$

where we consider the left neighbor vectors $D^L = D \cap \{L, I\}^n$ *and right neighbor vector* $D^R = D \cap \{I, R\}^n$.

3.3.2 Variable Arrangement

The main idea of variable arrangement in a LIR-tree is that vector variables have a left and a right converging value for direction in a cell.

Definition 27 (Variables)**.** *Let $w \in \mathcal{C}$ be a cell then its variables consist of $2n$ velocity value located on the faces and one pressure value located in the center:*

$$u_w \in \mathbb{R}^n \times \mathbb{R}^n \quad (u_w)_{w\in\mathcal{C}} = u \in \mathbb{R}^{|\mathcal{C}|\cdot 2n} \qquad \textit{velocity} \tag{3.13}$$

$$p_w \in \mathbb{R} \qquad (p_w)_{w\in\mathcal{C}} = p \in \mathbb{R}^{|\mathcal{C}|} \qquad \textit{pressure} \tag{3.14}$$

$$(u_w, p_w) = c_w \in \mathbb{R}^{2n+1} \qquad (c_w)_{w\in\mathcal{C}} = c \in \mathbb{R}^{|\mathcal{C}|\cdot(2n+1)} \qquad \textit{cell values} \tag{3.15}$$

For velocity these left and right values can be seen as flow entering and leaving the cell, respectively.

Definition 28 (Location)**.** *Let $\mathfrak{b} \in \mathfrak{B}$ be an interval and $v \in A^n$ be a vector then we define two projection operator $\downarrow_v$ and $\uparrow_v$. The first operator is defined by*

$$[\mathfrak{b}_L, \mathfrak{b}_R]\downarrow_L = \mathfrak{b}_L \qquad [\mathfrak{b}_L, \mathfrak{b}_R]\downarrow_I = \frac{1}{2}(\mathfrak{b}_L + \mathfrak{b}_R) \qquad [\mathfrak{b}_L, \mathfrak{b}_R]\downarrow_R = \mathfrak{b}_R \tag{3.16}$$

and maps the interval to an interior location pointed by v. The second operator maps to the opposing location and is defined by

$$[\mathfrak{b}_L, \mathfrak{b}_R]\uparrow_L = \mathfrak{b}_R \qquad [\mathfrak{b}_L, \mathfrak{b}_R]\uparrow_I = \frac{1}{2}(\mathfrak{b}_L + \mathfrak{b}_R) \qquad [\mathfrak{b}_L, \mathfrak{b}_R]\uparrow_R = \mathfrak{b}_L \tag{3.17}$$

These operators are used to point at the center of edges, faces and the interval itself. They can describe the location of velocity and pressure variables inside a cell.

3.3.3 Neighborhood Evaluation

Computations inside a cell often requires values also at the neighbors. In numerous experiments we found out that more complex methods of interpolating values for the neighbors (e.g. B-Splines) do not significantly increase the quality of the result. But they increase the computational time significantly. Therefore, we just average neighbor values and use linear interpolation.

Definition 29 (Size Determination)**.** *Let $w \in \mathcal{C}$ be a cell and $v \in A^n$ be a neighbor direction then we average the sizes of neighboring cells by*

$$\widetilde{\xi}^v_w := \frac{1}{|\mathcal{C}^v_w|} \sum_{q\in\mathcal{C}^v_w} \xi_q \tag{3.18}$$

The addition and multiplication of space partitioning functions is done by adding or multiply the left and right boundary vectors, respectively. In the case of solid neighbor nodes, i.e. $\mathcal{C}^v_w = \emptyset$ we can also use $\widetilde{\xi}^v_w = \xi^{RL}_{vv} \circ \xi_w$ for that neighbor. Let $\mathsf{v} \in \mathsf{A}^n$ then we define the volume evaluation operator by

$$|\xi|_{\mathsf{v}} := \prod_{\substack{i=1 \\ \mathsf{v}_i=\mathbf{I}}}^{n} |\xi|_i \tag{3.19}$$

where we consider directions that are split. These sizes are used to discretize differential operators.

Definition 30 (Value Determination)**.** *In the same way, let $w \in \mathcal{C}$ be a cell and $v \in A^n$ be a neighbor direction then we average the values of neighboring cells by*

$$\widehat{x}_w^v = \frac{1}{|\mathcal{C}_w^v|} \sum_{q \in \mathcal{C}_w^v} x_q \tag{3.20}$$

to define a representative value $\widehat{x}_w^v$. Averaging of neighbor values is denoted by the $\widehat{\cdot}$ operator. We increase the numerical accuracy by linear interpolation to the desired planes of the originating cell size. Let $w \in \mathcal{C}$ be a cell, $v \in A^n$ be a neighbor direction and $d \in \mathsf{d} \in \mathsf{D}$ a variable location of the variable x. Then all vectors $t \in \mathsf{d}$ that are parallel to d are used for linear interpolation defined by

$$\widetilde{x}_w^{v,d} = \sum_{t \parallel d} \frac{|[\xi_w \downarrow_d, \widetilde{\xi}_w^v \uparrow_t]|_{\mathsf{d}}}{|\widetilde{\xi}_w^v|_{\mathsf{d}}} \cdot \widehat{x}_w^{v,t} \tag{3.21}$$

The $\widetilde{\cdot}$ operator applied to variables is used to determine appropriate representative neighbor values used in our discretization.

A flow can be induced by a pressure drop at periodic neighbor cells in flow direction. In that case value determination $\widetilde{x}$ has to add a constant pressure drop accordingly.

The advantage of averaging sizes and values is alignment in the memory. That allows the usage of vectorized instructions in modern CPUs. As a consequence we sacrifice accuracy to gain more computational speed. If the accuracy is not sufficient in a certain region of the domain then we rely on refinement of the tree.

3.3.4 Discretized Differential Operators

At this point we know how to evaluate neighbor values and sizes for a given cell inside the LIR-tree. These values are used to discretize partial differential operators. For the Stokes equation we have to discretize the Laplacian and divergence for velocities and the gradient for the pressure. We present a general derivation and interpolation operator that can be used to discretize more complex differential operators.

Definition 31 (Derivation operator)**.** *We introduce a general derivation operator for k-dimensional variables embedded to a n-dimensional domain along equivalence classes. Let $\mathsf{v} \in \mathsf{A}^n$ and $\{l, r\} = \mathsf{d} \in \mathsf{D}$ where v denotes the variable and d a specified direction. Without loss of generality, we assume that $l \in D^L$ and $r \in D^R$. Then we define the derivation operator*

$$\partial_{\mathsf{d}} : \mathbb{R}^{|\mathcal{C}| \cdot |\mathsf{v}|} \to \mathbb{R}^{|\mathcal{C}| \cdot |\mathsf{v}+\mathsf{d}|} \tag{3.22}$$

$$x \mapsto \left(\begin{cases} \frac{x_w^{v+r} - x_w^{v+l}}{|\xi_w|_{\mathsf{d}}} & \text{if } v+l \in \mathsf{v} \wedge v+r \in \mathsf{v} \\ 2\frac{\widetilde{x}_w^{r,v+l} - x_w^{v+l}}{|\xi_w + \widetilde{\xi}_w^r|_{\mathsf{d}}} & \text{if } v+l \in \mathsf{v} \wedge v+r \notin \mathsf{v} \\ 2\frac{x_w^{v+r} - \widetilde{x}_w^{l,v+r}}{|\widetilde{\xi}_w^l + \xi_w|_{\mathsf{d}}} & \text{if } v+l \notin \mathsf{v} \wedge v+r \in \mathsf{v} \end{cases} \right)_{(w,v) \in \mathcal{C} \times (\mathsf{v}+\mathsf{d})} \tag{3.23}$$

The generalization to $\mathsf{d} \in \mathsf{A}^n$ is done by consecutive composition. The gradient operator arises from D defined by

$$\nabla = (\partial_{\mathsf{d}})_{\mathsf{d} \in \mathsf{D}} \tag{3.24}$$

This operator also used to discretize the Laplace and divergence operators.

Definition 32 (Interpolation operator)**.** *In order to move variable positions to other points inside a cell we define the interpolation operator*

$$\square_{\mathsf{d}} : \mathbb{R}^{|\mathcal{C}|\cdot|\mathsf{v}|} \to \mathbb{R}^{|\mathcal{C}|\cdot|\mathsf{v}+\mathsf{d}|} \tag{3.25}$$

$$x \mapsto \left(\begin{cases} \frac{x_w^{v+r}+x_w^{v+l}}{2} & \text{if } v+l \in \mathsf{v} \wedge v+r \in \mathsf{v} \\ \frac{|\xi_w|_{\mathsf{d}}\widetilde{x}_w^{r,v+l}+|\widetilde{\xi}_w^r|_{\mathsf{d}}x_w^{v+l}}{|\xi_w+\widetilde{\xi}_w^r|_{\mathsf{d}}} & \text{if } v+l \in \mathsf{v} \wedge v+r \notin \mathsf{v} \\ \frac{|\widetilde{\xi}_w^l|_{\mathsf{d}}x_w^{v+r}+|\xi_w|_{\mathsf{d}}\widetilde{x}_w^{l,v+r}}{|\widetilde{\xi}_w^l+\xi_w|_{\mathsf{d}}} & \text{if } v+l \notin \mathsf{v} \wedge v+r \in \mathsf{v} \end{cases} \right)_{(w,v)\in\mathcal{C}\times(\mathsf{v}+\mathsf{d})} \tag{3.26}$$

Similar to the derivation operator, we introduce a gradient-like vector operator along spatial directions defined by

$$\square = (\square_{\mathsf{d}})_{\mathsf{d}\in\mathsf{D}} \tag{3.27}$$

These operators can be composed with derivation operators, e.g. $\square\nabla$.

Definition 33 (Gradient)**.** *The gradient of pressure is discretized by the general gradient operators and reduces to*

$$\nabla p = (\nabla_w p)_{w\in\mathcal{C}} \in \mathbb{R}^{|\mathcal{C}||D|} \tag{3.28}$$

such that

$$\nabla_w p = \left(\begin{cases} 2\frac{\widetilde{p}_w^r - p_w}{|\xi_w+\widetilde{\xi}_w^r|_{\mathsf{d}}} & \text{if } v = r \\ 2\frac{p_w - \widetilde{p}_w^l}{|\widetilde{\xi}_w^l+\xi_w|_{\mathsf{d}}} & \text{else} \end{cases} \right)_{v\in\{l,r\}=\mathsf{d}\in\mathsf{D}} \tag{3.29}$$

for a cell $w \in \mathcal{C}$*. The pressure value in the center yields a pressure gradient value for each face center.*

Definition 34 (Laplacian)**.** *Let* $w \in \mathcal{C}$ *be a cell inside a LIR-tree then and* $v \in A^n$ *be a vector that identifies a position on the face of a cell we discretize the Laplacian for velocities by*

$$\nabla^2 u = \left(\nabla_w^2 u\right)_{w\in\mathcal{C}} \in \mathbb{R}^{|\mathcal{C}||D|} \tag{3.30}$$

such that

$$\nabla_w^2 u = \left(\sum_{\{l,r\}=\mathsf{d}\in\mathsf{D}} \begin{cases} \frac{2}{|\widetilde{\xi}_w^l+\xi_w|_{\mathsf{d}}} \left(\frac{u_w^r-u_w^l}{|\xi_w|_{\mathsf{d}}} - \frac{\widetilde{u}_w^{l,r}-\widetilde{u}_w^{l,l}}{|\widetilde{\xi}_w^l|_{\mathsf{d}}} \right) & \text{if } v = l \\ \frac{2}{|\xi_w+\widetilde{\xi}_w^r|_{\mathsf{d}}} \left(\frac{\widetilde{u}_w^{r,r}-\widetilde{u}_w^{r,l}}{|\widetilde{\xi}_w^r|_{\mathsf{d}}} - \frac{u_w^r-u_w^l}{|\xi_w|_{\mathsf{d}}} \right) & \text{if } v = r \\ \frac{2}{|\xi_w|_{\mathsf{d}}} \left(\frac{\widetilde{u}_w^{r,v}-u_w^v}{|\xi_w+\widetilde{\xi}_w^r|_{\mathsf{d}}} - \frac{u_w^v-\widetilde{u}_w^{l,v}}{|\widetilde{\xi}_w^l+\xi_w|_{\mathsf{d}}} \right) & \text{else} \end{cases} \right)_{v\in D} \tag{3.31}$$

The first two cases consider parallel neighbors where the last case considers orthogonal neighbor values.

Definition 35 (Divergence)**.** *Let* $w \in \mathcal{C}$ *be a cell then we discretize the divergence for velocities by*

$$\nabla \cdot u = ((\nabla\cdot)_w u)_{w\in\mathcal{C}} \in \mathbb{R}^{|\mathcal{C}|} \tag{3.32}$$

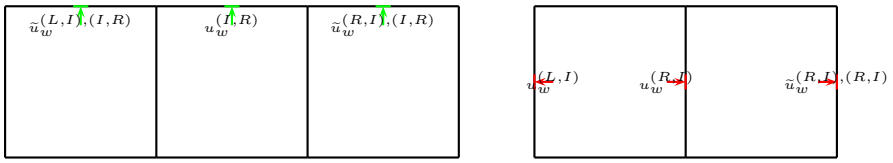


Figure 3.2: Two cases of the Laplacian with continuity condition.

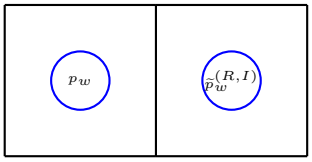


Figure 3.3: Pressure gradient.

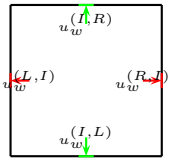


Figure 3.4: Divergence.

such that

$$(\nabla\cdot)_w u = \sum_{\{l,r\}=\mathsf{d}\in\mathsf{D}} \frac{u_w^r - u_w^l}{|\xi_w|_\mathsf{d}} \tag{3.33}$$

where we sum up the discretized derivations. Note that no neighbor values are used to satisfy the mass conservation.

Definition 36 (Continuity)**.** *In order to ensure a continuous velocity field we introduce the continuity condition defined by*

$$M_{w,q}^{v,d} \in \mathbb{R}|_{\leftrightarrow} := \begin{cases} M_{w,w}^{v,v} - \sum_{p\in\mathcal{C}_w^v} M_{w,p}^{v,-v} & \text{if } w = q \wedge d = v \in D \\ 0 & \text{if } q \in \mathcal{C}_w^v \wedge -d = v \in D \\ M_{w,q}^{v,d} & \text{else} \end{cases} \tag{3.34}$$

The coefficients of opposing neighbor velocity values are moved to the cell w. The continuity operators enforces left and right converging components to converge to (approximately) the same values.

Definition 37 (No Slip)**.** *The no-slip condition is enforced by replacing certain momentum equations by equality equations. If a velocity values lives on the empty-solid interface then its corresponding momentum equation is replaced by an equation that sets its value to zero, i.e.*

$$M_{w,q}^{v,d} \in \mathbb{R}|^{\partial\Omega=0} := \begin{cases} M_{w,q}^{v,d} & \text{if } \mathcal{C}_w^v \neq \emptyset \\ 1 & \text{if } v = d \\ 0 & \text{else} \end{cases} \tag{3.35}$$

and for vectors, i.e.

$$f_w^v|^{\partial\Omega=0} := \begin{cases} f_w^v & \text{if } \mathcal{C}_w^v \neq \emptyset \\ 0 & \text{else} \end{cases} \tag{3.36}$$

The notation $|^{\partial\Omega=0}$ can be applied to a matrix and a vector.

3.4 Solving Single Cell Systems

The main idea of solving a partial differential equation with a LIR-tree is defining a local linear system for each cell. That linear system has to be discretized such that a cell can satisfy the PDE itself without changing neighbor values. Hence the global linear system can be seen as block linear system. These blocks have to be regular at the diagonal such that the linear system is block-diagonally dominant. That kind of describing a PDE with a block-wise matrix is called single cell system.

3.4.1 Stokes Block System

Definition 38 (Block System Matrix)**.** *Let $w, q \in \mathcal{C}$ then a block is defined as $M_{w,q} \in \mathbb{R}^{(2n+1)^2}$ that can be gathered to a matrix of blocks, i.e.*

$$M = \left(M_{w,q} \in \mathbb{R}^{(2n+1)^2}\right)_{(w,q)\in\mathcal{C}^2} \in \mathbb{R}^{(|\mathcal{C}|\cdot(2n+1))^2} \tag{3.37}$$

We use the notation M_w to denote the row block vector for the cell w. M is called the block system matrix.

Definition 39 (Block Linear System)**.** *The Stokes equation is discretized by gathering the discretizations for each differential operator to blocks, i.e.*

$$M \cdot c = \begin{pmatrix} \nabla^2 u - \nabla p \\ \nabla \cdot u \end{pmatrix}\Big|_{\leftrightarrow}^{\partial\Omega=0} = \begin{pmatrix} \nabla_w^2 u - \nabla_w p \\ (\nabla\cdot)_w u \end{pmatrix}\Big|_{\leftrightarrow}^{\partial\Omega=0}{}_{w\in\mathcal{C}} = \tag{3.38}$$

$$= f = \left(f_w|^{\partial\Omega=0}\right)_{w\in\mathcal{C}} \tag{3.39}$$

where $f \in \mathbb{R}^{|\mathcal{C}|\cdot(2n+1)}$ is a given right hand side.

Theorem 5. *Let $w \in \mathcal{C}$ a cell then its block $M_{w,w}$ is regular, i.e. the inverse block $M_{w,w}^{-1}$ such that $M_{w,w} \cdot M_{w,w}^{-1} = \mathbf{1}$ exists.*

Example 3. *Let $n = 2$ then we get the two-dimensional Stokes equation. For simplicity we set all cell sizes and the dynamic viscosity constant to one. We also assume periodic boundary conditions for the cells at the border of the domain. The variable arrangement for the two-dimensional case and a regular grid like LIR-tree is illustrated in figure 3.5. The top right region is solid and introduces no-slip boundary conditions for the bottom left and top left cells.*

The block equation for the bottom left cell $(L, L) \in \mathcal{C}$ is defined by

$$\begin{pmatrix} -4 & 1 & & & -1 \\ 1 & -4 & & & 1 \\ & & -4 & 1 & -1 \\ & & 1 & -4 & 1 \\ -1 & 1 & -1 & 1 & \end{pmatrix} \cdot \begin{pmatrix} u_{(L,L)}^{(L,I)} \\ u_{(L,L)}^{(R,I)} \\ u_{(L,L)}^{(I,L)} \\ u_{(L,L)}^{(I,R)} \\ p_{(L,L)} \end{pmatrix} = \begin{pmatrix} b_{(L,L)}^{(L,I)} \\ b_{(L,L)}^{(R,I)} \\ b_{(L,L)}^{(I,L)} \\ b_{(L,L)}^{(I,R)} \\ b_{(L,L)}^{(I,I)} \end{pmatrix} \tag{3.40}$$

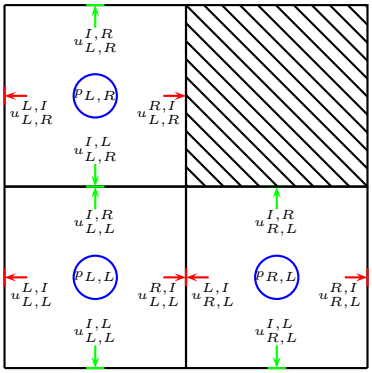


Figure 3.5: Variable arrangement for pressure and velocity values in the two-dimensional case.

with the right hand side

$$\begin{pmatrix} b^{(L,I)}_{(L,L)} \\ b^{(R,I)}_{(L,L)} \\ b^{(I,L)}_{(L,L)} \\ b^{(I,R)}_{(L,L)} \\ b^{(I,I)}_{(L,L)} \end{pmatrix} = \begin{pmatrix} -u^{(L,I)}_{(R,L)} - u^{(L,I)}_{(L,R)} - u^{(L,I)}_{(L,R)} - p_{(R,L)} - f^{(L,I)}_{(L,L)} \\ -u^{(R,I)}_{(R,L)} - u^{(R,I)}_{(L,R)} - u^{(R,I)}_{(L,R)} + p_{(R,L)} - f^{(R,I)}_{(L,L)} \\ -u^{(I,L)}_{(R,L)} - u^{(I,L)}_{(R,L)} - u^{(I,L)}_{(L,R)} - p_{(L,R)} - f^{(I,L)}_{(L,L)} \\ -u^{(I,R)}_{(R,L)} - u^{(I,R)}_{(R,L)} - u^{(I,R)}_{(L,R)} + p_{(L,R)} - f^{(I,R)}_{(L,L)} \\ 0 \end{pmatrix} \tag{3.41}$$

The block matrix has the inverse block matrix

$$M^{-1}_{(L,L),(L,L)} = \frac{1}{60} \begin{pmatrix} -13 & -7 & 3 & -3 & -15 \\ -7 & -13 & -3 & 3 & 15 \\ 3 & -3 & -13 & -7 & -15 \\ -3 & 3 & -7 & -13 & 15 \\ -15 & 15 & -15 & 15 & 75 \end{pmatrix} \tag{3.42}$$

The block system matrix has the following appearance

$$\left(\begin{array}{ccccc|ccccc|ccccc} -4 & 1 & & & -1 & 1 & & & & 1 & 2 & & & & \\ 1 & -4 & & & 1 & & 1 & & & -1 & & 2 & & & \\ & & -4 & 1 & -1 & & & 2 & & & & & 1 & & 1 \\ & & 1 & -4 & 1 & & & & 2 & & & & & 1 & -1 \\ -1 & 1 & -1 & 1 & & & & & & & & & & & \\ \hline 1 & & & & -1 & -4 & 1 & & & -1 & & & & & \\ & 1 & & & 1 & 1 & -4 & & & 1 & & & & & \\ & & & & & & & 1 & & & & & & & \\ & & & & & & & & 1 & & & & & & \\ & & & & & -1 & 1 & -1 & 1 & & & & & & \\ \hline & & & & & & & & & & 1 & & & & \\ & & & & & & & & & & & 1 & & & \\ & & 1 & & 1 & & & & & & & & -4 & 1 & -1 \\ & & & 1 & -1 & & & & & & & & 1 & -4 & 1 \\ & & & & & & & & & & -1 & 1 & -1 & 1 & \end{array}\right)$$

that can also be written in the compact form

$$M = \left(\begin{array}{c|c|c} M_{(L,L),(L,L)} & M_{(L,L),(R,L)} & M_{(L,L),(L,R)} \\ \hline M_{(R,L),(L,L)} & M_{(R,L),(R,L)} & \mathbf{0} \\ \hline M_{(L,R),(L,L)} & \mathbf{0} & M_{(L,R),(L,R)} \end{array}\right) = \left(\begin{array}{c} M_{(L,L)} \\ M_{(R,L)} \\ M_{(L,R)} \end{array}\right) \tag{3.43}$$

The diagonal blocks $M_{(R,L),(R,L)}$ and $M_{(L,R),(L,R)}$ for the cells (R, L) and (L, R) contain equality equations introduced by the no-slip boundary conditions.

In numerous experiments with different kinds of data we found that the number of different blocks is limited and considerably less than the number of cells, i.e.

$$\{M_{w,w} \in \mathbb{R}^{2n+1} : w \in \mathcal{C}\} \ll |\mathcal{C}| \tag{3.44}$$

Therefore, the performance can be improved by using precomputed inverted diagonal and sparse neighbor blocks. In our implementation we used that approach for cells that reached the limit of space partitioning.

3.4.2 Gauß-Seidel for Block Linear Systems

If a diagonal invertible block system matrix is given then we can use the Gauß-Seidel algorithm as iterative solver. It is also possible to use the Jacobi algorithm so solve block linear systems. But due to the slower convergence and higher memory requirement we restrict to the Gauß-Seidel algorithm.

Definition 40 (Gauß-Seidel with Relaxation for BLS)**.** *Let $M = (M_{q,w} \in \mathbb{R}^{k^2})_{w,q\in\mathcal{C}} \in \mathbb{R}^{k^2\cdot|\mathcal{C}|^2}$ be a diagonally dominant and invertible block system matrix with the corresponding linear system $M \cdot c = f$ then we can derive the Gauß-Seidel algorithm for $w \in \mathcal{C}$ with*

$$M_w \cdot c = \sum_{q\in\mathcal{C}} M_{w,q} \cdot c_q = f_w \tag{3.45}$$

where the value c_w can be extracted as

$$M_w \cdot c = M_{w,w} \cdot c_w + \sum_{q\in\mathcal{C}_w^L} M_{w,q} \cdot c_q + \sum_{q\in\mathcal{C}_w^R} M_{w,q} \cdot c_q = f_w \tag{3.46}$$

The iterative approach is given by

$$\widehat{c}_w^{i+1} = M_{w,w}^{-1} \cdot \left(f_w - \sum_{q\in\mathcal{C}_w^L} M_{w,q} \cdot c_q^{i+1} - \sum_{q\in\mathcal{C}_w^R} M_{w,q} \cdot c_q^i \right) \tag{3.47}$$

where we restrict to neighbor cells. The proceeding value c_w^{i+1} is defined by

$$c_w^{i+1} = (1-\alpha) \cdot c_w^i + \alpha \cdot \widehat{c}_w^{i+1} \tag{3.48}$$

with the relaxation parameter $\alpha \in (0, 2) \subset \mathbb{R}$.

In numerous experiments the following Krylov subspace methods have been considered: Biconjugate gradient stabilized method (BiCGStab), Generalized minimal residual method (GMRES), Conjugate Gradient Squared (CGS). But in the context of block linear systems they provide no significant advantages over the Gauß-Seidel algorithm. The opposite is true. The memory requirements are higher and the convergence behavior is too unstable. In many cases convergence could not be obtained.

3.5 Refinement Rules

The input function described in section 2.9 is used as an initial guess for the tree structure, i.e.

$$\Omega_0(w) = \psi^{-1}(\zeta^k(\xi_w^L(\mathbf{b}))). \tag{3.49}$$

But after some Gauß-Seidel iterations we can use the actual velocity field to refine the tree and improve the accuracy where it is needed. Therefore, we use iterative refinement. The initial guess may already have numerical adverse properties, e.g. the cell size ratio between neighboring cells.

3.5.1 Maximum Neighbor Size Ratio

The preservation of a maximal neighbor size ratios increases the numerical stability and accuracy. Therefore, we introduce a iterative modifier function that converges a given tree structure until a specified maximal ratio is given.

Definition 41 (Size Ratio preservation)**.** *Let $k \in \mathbb{N}$ be a threshold then we define the modifier function ι by*

$$\iota : \mathbf{I}^{*\times n} \to S \tag{3.50}$$

$$w \mapsto \left(\bigvee_{e \subseteq v \in D} \exists_{q \in \mathcal{C}_{we}^v} |q|_i - |w|_i > k \right)_{\substack{e \in S \\ e_i = I}} \tag{3.51}$$

$$w \mapsto \left(\bigvee_{e \subseteq v \in D} \exists_{q \in \mathcal{C}_{we}^v} |\xi_w|_{e/\perp} > k \cdot |\xi_q|_{e/\perp} \right)_{\substack{e \in S \\ e_i = I}} \tag{3.52}$$

that returns splits for a cell w if there exists neighbor cell with exceeded size ratio. The first implementation uses the length of words and the second implementation considers actual cell sizes. We suggest $k = 1$ for the first and $k = 3$ for the second approach.

Before numerical calculations takes place the initial tree is modified by the presented neighbor size ratio preservation. These operations are computationally expensive and should be done once after geometry analysis. Additional refinement approaches should includes the preservation of neighbor size ratio implicitly. Thus, we describe the set of edges that are allowed to be split by

$$\mathcal{E}_{\text{split}} = \{(w, e) \in \mathcal{C} \times E : \forall_{i=1}^n e_i = I \Rightarrow (|\xi_w|_i > t \wedge |\xi_w|_i \geq |\xi_{q \in \mathcal{C}_w}|_i)\} \tag{3.53}$$

where t is the limit of partitioning, see Section 2.9. The additional condition considering the cell size preserves the cell size ratio between neighboring cells. This restriction allows to easily formulate refinement algorithms with that property.

3.5.2 Difference Reduction

We introduce a difference reduction approach for refinement that uses the actual solution (e.g. velocity field) to predict where a higher numerical accuracy

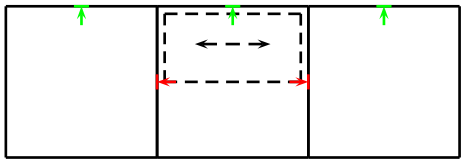


Figure 3.6: Edge analysis for difference reduction. The dashed entity denotes the analyzed edge. In parallel direction the upper vertical velocity values of the neighbors and the horizontal velocity values of the cell itself are considered.

is needed. The basic idea of solution dependent refinement is to use value differences between neighboring cells. We split a cell if its values differs more than a certain threshold with respect to the values of the neighbor cells. The cell size is also taken into account. We assume that we have to increase the accuracy where the value difference is high. The aim of that method is to reduce the overall value differences.

Definition 42. *Let x be the variable we want to use for refinement (e.g. velocity) where the position of its values in each cell are defined by V_k. Then we define the modifier function*

$$\iota : \mathbf{I}^{*\times n} \to S \tag{3.54}$$

$$w \mapsto \left(\begin{cases} \lambda_{w,e} > \alpha \cdot \max \lambda & \text{if } (w,e) \in \mathcal{E}_{split} \\ 0 & \text{else} \end{cases} \right)_{e \in E} \tag{3.55}$$

where we use the auxiliary values defined by

$$\lambda_{w,e} = \max \begin{cases} |x_w^v - x_w^d| : v, d \in V_k \wedge v \parallel d \wedge v, d \cap e \\ |\widetilde{x}_q^{v,d} - x_w^d| : d \in V_k \wedge e \perp v \wedge e \subseteq d \end{cases} \tag{3.56}$$

For each edge $e \in E$ we consider the neighbor cells in orthogonal directions. If there exist values at the positions $e \subseteq d$ that differ more than a certain threshold then the edge is split. We also consider values in parallel positions inside w that intersects e. The threshold is based on the maximum over all differences $\max \lambda$ *and the coefficient α.*

The question remains how to choose the coefficient α. We require to control the number of cells generated for each refinement. Therefore, we allow to specify a desired number of cells and introduce an approach to determine a suited coefficient α.

Definition 43. *Let $c \in \mathbb{N}$ be the desired number of cells and $\gamma \in \mathbb{N}$ be a growth factor (e.g. $\gamma = \frac{1}{2}$) then we aim to increase the number of cells by*

$$h = \min(\gamma \cdot |\mathcal{C}_i|, c) - |\mathcal{C}_i| \tag{3.57}$$

An approximation to that requirement is to choose α such that

$$|\{(w,e) \in \mathcal{E}_{split} : \lambda_{w,e} > \alpha\}| = h \tag{3.58}$$

This can be accomplished by using histograms $[0, \max\lambda] \subset \mathbb{R}$. *Let* $k \in \mathbb{N}$ *then we define the vector* $(\mathcal{E}^i_{split} \in \mathbb{N})^k_{i=1}$ *by*

$$\mathcal{E}^i_{split} = \left| \left\{ (w,e) \in \mathcal{E}_{split} : \frac{i-1}{k} \max\lambda < \lambda_{w,e} \leq \frac{i}{k} \max\lambda \right\} \right| \tag{3.59}$$

Histograms allow to find a suited coefficient easily by

$$\alpha = \frac{1}{k} \min \left\{ i \in \mathbb{N} : \sum_{j=i}^{k} \mathcal{E}^j_{split} < h \right\} \tag{3.60}$$

In our experiments the approach of determining the coefficient α converges to a LIR-tree such that the desired number of cells is approximated. That property allows to analyse the behavior of our method with respect to the number of cells in addition to the cell size.

We also tried to use the pressure field to improve the structure of the tree. But our experiments showed that using the pressure field introduces clutter. Therefore, we stick to the velocity field. Moreover, gradient based refinement is not suited to improve the accuracy where it is needed.

3.6 Parallelization

In the context of large scale datasets, parallelization is a very important topic in numerical mathematics. In this section we give an overview to parallelize the Gauß-Seidel algorithm.

The first levels in the LIR-tree partition the computational domain into macroblocks. Hence, the input function based on the geometry is modified such that the first levels in the LIR-tree yield a regular grid of macroblocks. Figure 3.7 depict the dependencies between macroblocks that are needed to gain a well-formed Gauß-Seidel algorithm.

In this context, $\overline{\Omega} \subset \Omega$ is the sub-tree describing the first levels that are parallelized. Let $w \in \overline{\mathcal{C}}$ be a macroblock in the j_w-th iteration step then we can do the Gauß-Seidel step if each left neighbor in $\overline{\mathcal{C}}^v_w$ is in the same or one iteration ahead. That depends on whether the neighbor is a perodic neighbor, i.e.

$$\forall_{q \in \overline{\mathcal{C}}^L_w} \begin{cases} j_q > j_w & \text{if } \xi^{RL}_{vv}(\xi_w(\mathbf{b})) \subseteq \mathbf{b} \\ j_q = j_w & \text{else} \end{cases} \tag{3.61}$$

As a consequence, threads are working on different iterations asynchronous. Boundaries of macroblocks are sent after processing and not on demand. If the computation is done on a shared memory machine then there is no need of data duplication for the boundaries.

Let $m \in \mathbb{N}$ be the number of threads and $\mathfrak{b} \in \mathfrak{B}$ be the computational domain. Then we need at least $k = m$ macro blocks per direction that can be processed in parallel for the two-dimensional case. For the the three-dimensional case we need $k = \left\lceil \sqrt{\frac{1}{4} + 2m} - \frac{1}{2} \right\rceil$ macro blocks. We introduce the three operators

$$\lceil k \rceil_2 := 2^{\left\lceil \frac{\log(k)}{\log(2)} \right\rceil} \qquad \lfloor k \rfloor_2 := 2^{\left\lfloor \frac{\log(k)}{\log(2)} \right\rfloor} \qquad \lfloor k \rceil_2 := \underset{x \in \{\lfloor k \rfloor_2, \lceil k \rceil_2\}}{\arg\min} |k - x| \tag{3.62}$$

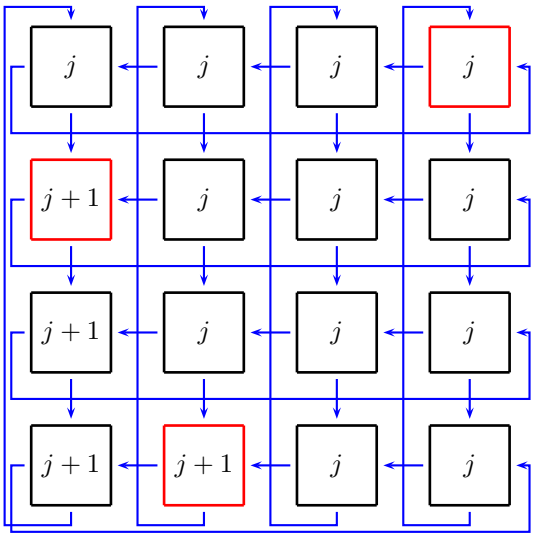


Figure 3.7: Parallelization scheme for the Gauß-Seidel algorithm. Squares represent macro blocks and the blue arrows indicate their dependencies. Three parallel threads are working on the macro blocks highlighted in red. The numbers inside the squares show the current iteration.

that quantize to upper or low powers of two, respectively. Then we define the auxiliary domain $\mathfrak{c} = \frac{\min \lfloor \mathfrak{b} \rfloor}{\lceil k \rceil_2}$ where we use the upper power of two from k. The domain $\mathfrak{c}$ describes the regular grid of macro blocks. We quantize to the nearest power of two by $\widehat{\mathfrak{c}} = [\lfloor \mathfrak{c}_L \rceil_2, \lfloor \mathfrak{c}_R \rceil_2]$. Finally, the modifier function that construct an appropriate parallelized grid $\overline{\Omega}$ is given by

$$\iota : \mathbf{I}^{*\times n} \to S \qquad w \mapsto \left(|\xi_w(\widehat{\mathfrak{c}})|_i > \frac{1}{2} \max |\xi_w(\widehat{\mathfrak{c}})| \wedge |\xi_w(\widehat{\mathfrak{c}})|_i > 1 \right)_{\substack{e \in E \\ e_i = I}} \tag{3.63}$$

The first term splits in elongated directions first. This is an important property since most geometries are not cubic.

The advantage of this parallelization approach is the small number of global synchronizations. In addition, sequential and parallel processing yield the exact same result for each cell. A synchronization of all threads is only needed after a given number of iterations. After synchronization there is a chance to do refinement on the LIR-tree and to check if a specified termination condition is reached.

3.7 Splines

After solving the Stokes equations there is a need for retrieving a velocity and pressure value for each point in the computational domain. Important applications areas are:

- Visualization for analysis and advertisement
- Feature extraction

- Tracking (e.g. particles)

Therefore, we introduce a pice-wise constant and a smooth representation for the flow and pressure field.

3.7.1 Piece-wise constant Spline

In order to analyse the behavior of the iterative solution process it is useful to look at the raw data. Thus, we can use piece-wise constant functions to get an insight to the data.

Definition 44 (Piece-wise constant functions)**.** *A simple approach for defining a velocity field in $\mathbb{R}^n$ and a scalar pressure field in $\mathbb{R}$ for a LIR-tree is using pice-wise constant functions*

$$u_{\mathsf{v}\in\mathsf{D}} : x \in \xi_{w(v\in\mathsf{v})}(\mathfrak{B}) \mapsto u_w^v \tag{3.64}$$

$$p : x \in \xi_w(\mathfrak{B}) \mapsto p_w \tag{3.65}$$

where we assign variable values to their corresponding domain.

3.7.2 Subdivision Spline

To get a deeper insight to the data one can also use a smoothed version. In this section we present smooth G^1 steady approximation that can be applied to the velocity and pressure field. The basic idea is to use a recursive taylor series. Assume we want to use $\Omega_0 \in \mathbf{\Omega}_P$ as initial data for approximation.

Definition 45. *Then the sequence of oracles defined by the modifier function*

$$\iota_j = \vartheta_{l_j} \qquad l_0 = 2^{\max(\lceil \log|\mathbf{b}_i|/\log 2\rceil)_{i=1}^n} \qquad l_{j+1} = \frac{l_j}{2} \tag{3.66}$$

with the increasing thresholds such that Ω_0 converges to a regular grid. The threshold sequence l_i ensures that the largest cell are split first.

Corollary 3. *There exists a $k \in \mathbb{N}$ such that for each $i \geq k$ the trees are equal, i.e.*

$$\lim_{i\in\mathbb{N}} \Omega_i \in \mathbf{\Omega}_P = \Omega_i = \Omega_k \tag{3.67}$$

The limit always exists and all cell sizes are equal, that is

$$\forall_{i\geq k}\forall_{w,q\in\mathcal{C}_i} |\xi_w| = |\xi_q| \tag{3.68}$$

Definition 46. *Let $(\Omega_i \in \mathbf{\Omega}_P)_{i\in\mathbb{R}}$ be a sequence of oracles for the variable x. Let $w \in \mathcal{C}_{j\in\mathbb{N}}$ be the origin cell such that $\Omega_j(w) = \emptyset$ and $\Omega_{j+1}(w) \neq \emptyset$. Let $v \in \Omega_{j+1}(w)$ a child vector for the next iteration $j+1$ then the value for the child cell $wv \in \mathcal{C}_{j+1}$ at position $d \in \mathsf{d} \in \mathsf{A}^n$ is defined by*

$$x_{j+1,wv}^d = x_{j,w}^d + (\xi_{wv} - \xi_w)\downarrow_d \cdot \square_{j,w}\nabla_{j,w}x_j \tag{3.69}$$

We use first terms of the taylor series with the origin $\xi_w \downarrow_d$ and discretized derivations. The coefficients are determined from the cell sizes.

Theorem 6. *The subdivision spline is a G^1 steady approximation.*

Proof. We know that the initial tree converges to a regular grid. Hence, until that point we can see the subdivision scheme as control point generation. On that point we subdivide regular grids with straight interpolation between neighbor cells. □

3.8 Termination

At this point we know how to refine the discretization defined by a LIR-tree to converge to the analytical solution and how to solve the block linear system defined by the actual discretization. However the question still remains in which order refinement and solving should take place. The order affects the termination condition as well. In this section we describe an approach to control the refine and solution process.

Let $i \in \mathbb{N}$ be the solver iteration number and $(l, r) = \mathsf{v} \in \mathsf{D}$ be the direction where the apply a force or a pressure difference, respectively. Then

$$\widehat{u}_i^j = \frac{1}{2|\mathfrak{D}|} \sum_{w \in \mathcal{C}} (u_{i,w}^r + u_{i,w}^l) |\xi_w(\mathfrak{D})| \tag{3.70}$$

defines the mean velocity along that direction. It is reasonable to check the termination condition after a given number of iterations. Therefore, let $h \in \mathbb{N}$ be a check-interval then the relative difference to the previous considered mean velocity is given by

$$t_i = \left| \frac{\widehat{u}_i^j - \widehat{u}_{i-h}^j}{\widehat{u}_{i-h}^j} \right| \tag{3.71}$$

These values are also called tolerances and used to decide wether the solution of the current discretization is found.

Let $(j_i \in \mathbb{N})_{i \in \mathbb{N}}$ be a sequence with $j_{i+1} = j_i \vee j_{i+1} = j_i + 1$ then Ω_{j_i} describes the schedule of refinement. Similar, we define the refine-tolerance

$$r_i = \frac{||\mathcal{C}_i| - |\mathcal{C}_{i-1}||}{|\mathcal{C}_{i-1}|} \tag{3.72}$$

where i describes the refinement iteration number.

Let $0 < \epsilon_{\text{sol}} \in \mathbb{R}$ be a given tolerance. If no refinement is used then $t_i < \epsilon_{\text{sol}}$ is a suited termination condition. Otherwise we introduce a converge-first order. Let $0 < \epsilon_{\text{ref}} \in \mathbb{R}$ and $\epsilon_{\text{keep}} \in \mathbb{N}$. In converge-first order refinement is done if the current tolerance t_i is below the given threshold ϵ_{ref}, i.e.

$$j_{i+1} = \begin{cases} j_i + 1 & \text{if } t_i < \epsilon_{\text{ref}} \wedge r_{j_i} \geq \epsilon_{\text{keep}} \\ j_i & \text{else} \end{cases} \tag{3.73}$$

Refinement is done until the number of cells that have been introduced or removed is below the threshold ϵ_{keep}. The termination condition is reached if $t_i < \epsilon_{\text{sol}}$ and $r_{j_i} < \epsilon_{\text{keep}}$, i.e. we found an optimal discretization and its corresponding solution.

3.9 Results

In this section we describe experiments where we applied the LIR-tree and the LIR-solver on different kinds of datasets to analyze numerical properties. That are convergence of:

- Conservation law based cellular structure to analytical solutions

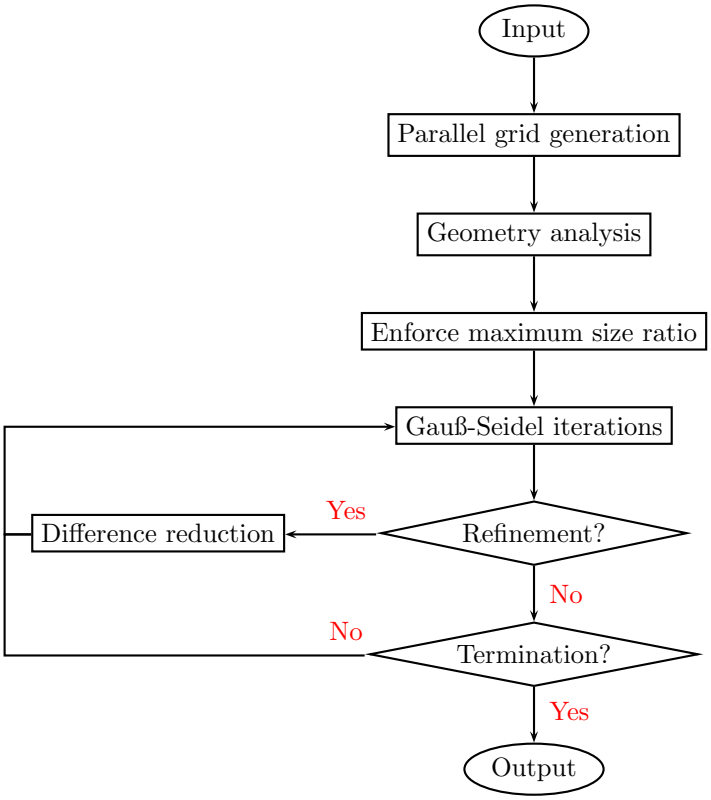


Figure 3.8: Outline of the solver. The initial guess for the discretization is constructed by the parallel grid generation, geometry analysis and the maximum neighbor size ratio condition. The Block-Gauß-Seidel is used for a number of iterations. Then there is a chance for refinement or termination.

- Refinement to analytical solutions
- Gauß-Seidel for block linear systems

Computational aspects are of greater importance too. We investigate the following properties

- Parallelization
- Runtime
- Memory requirements

that have to be concerned in the context of large geometries. We get an inside to the mentioned aspects by applying different geometries. Table 3.1 lists different computational resources that we used in our experiments where performance is considered. The sketch in Fig. 3.8 shows the outline of the whole algorithm.

3.9.1 The Geometries

First, we introduce four geometries used in our experiments. That are

Name	CPU	Clock rate	Cores	Memory
Hermes	Intel(R) Xeon(R)	3.0 GHz	8	16 GB
Hannover	Intel(R) Xeon(R) X5690	3.46 GHz	24	96 GB
Golem	AMD Opteron(tm) 6282 SE	2.6 GHZ	64	512 GB

Table 3.1: Computational resources used in our experiments

- periodic arrays of spheres
- generated fiber microstructure
- computed tomography scans of sandstone carbonate
- computed tomography scans of berea sandstone

The geometries are suited for different kind of analysis.

Arrays of Spheres

A widely accepted geometry for convergence analysis are periodic arrays of spheres. They are used to show convergence of a method to known analytical solutions. It can also be used for the Stokes equations where we know the permeability with respect to varying diameters, see [30]. We use a unit cube with periodic boundary condition where a sphere with given diameter is located at the center. The interior of the sphere is assumed be solid. Figure 3.9 shows a sphere inside a unit box with a diameter of $\chi = 0.5$ and a spatial resolution of 128 voxel per direction.

Fiber Microstructure

The next dataset is a computer generated fiber microstructure. The geometry has similar properties to the one we used in section 2.11.1. The dataset consists of two different types ($10\mu m$ and $6\mu m$ diameter) of fibers in micrometer scale. The distribution count of both types is 50%. The fiber orientation is anisotropic distributed but more isotropic in $x - y$ direction. The porosity with approximately 92% is very high. We have an analytical description of the geometry. Therefore, we can use a high resolution instance for convergence analysis. The fiber structure can be used to measure computational performance as well. Figure 3.10 shows two- and three-dimensional cuts through the datasets.

Sandstone Carbonate

The underlying structure of the sandstone carbonate dataset is a very complex pore network with many connections and dead ends. The pores are distributed inhomogeneous. That property can be challenging for parallelization. The porosity value of 18% is very low, in contrast to the fiber microstructure. The dataset is used to analyize the performance of our approach compared to other methods. The geometry is challenging for the LIR-tree due to the large boundary area. Figure 3.9 shows different views of the datasets and reveals its complex nature. The dataset is given as a 750^3 binary voxel geometry with a voxellength of $2.99\mu m$

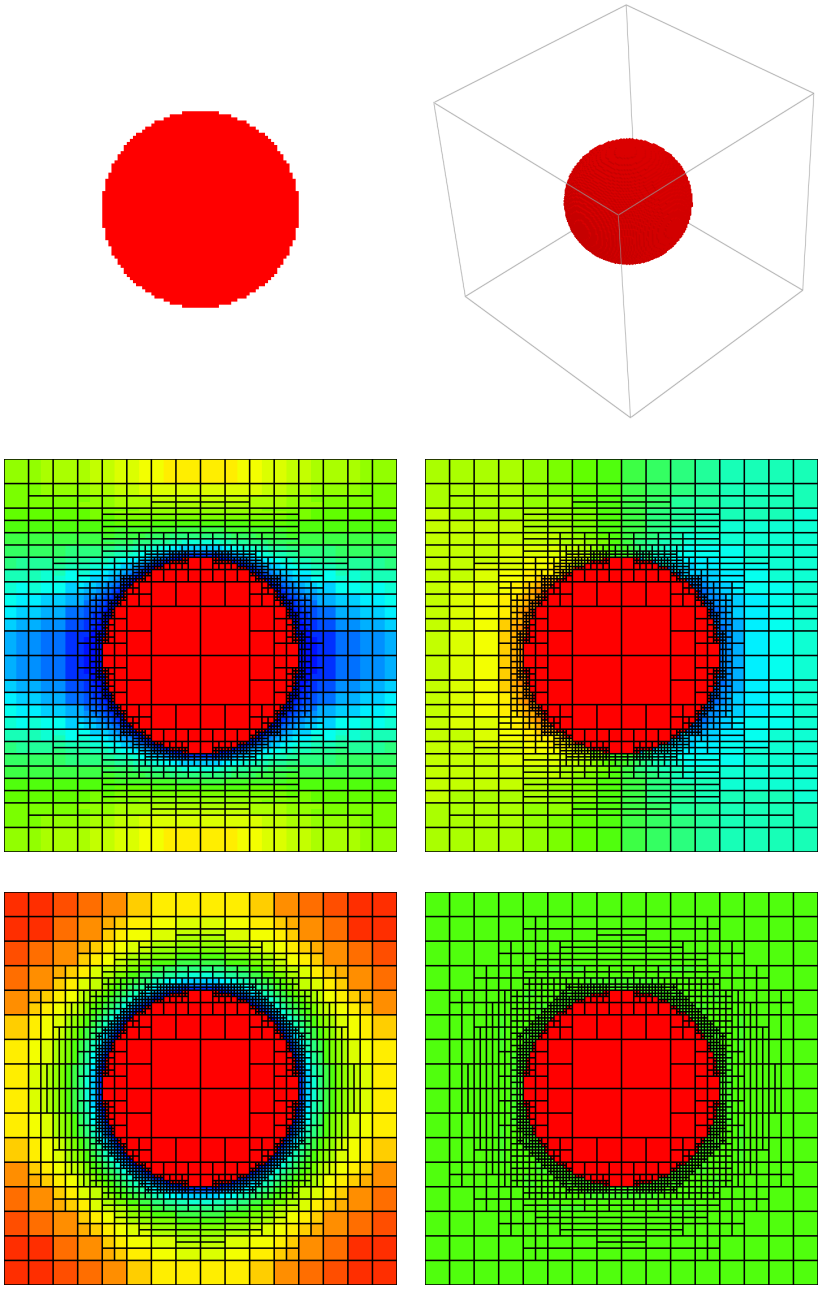


Figure 3.9: Unit box with a spherical obstacle in 2D (upper left) and 3D (upper right). The sphere has a diameter of $\chi = 0.5$ and a resolution of 128 voxel per direction. The centered images show velocity and pressure drop from left to right. The bottom images show velocity and pressure toward the view plane.

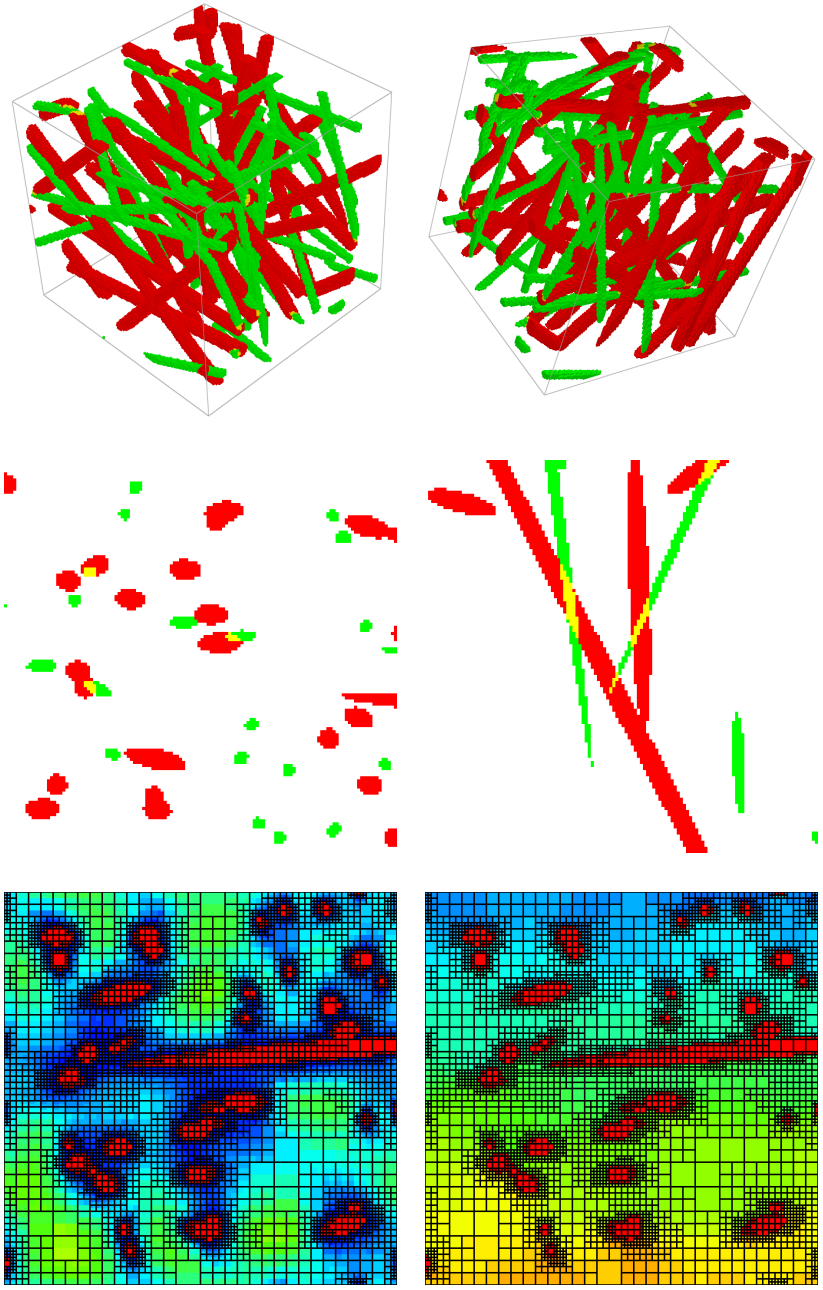


Figure 3.10: Different views of a fiber microstructure. The center left image shows a cut along the $y-z$-plane and the center right image shows a cut along the $x-y$-plane. Red fibers have $10\mu m$ diameter and green fibers have $6\mu m$ diameter. The bottom images show velocity and pressure drop from bottum to top.

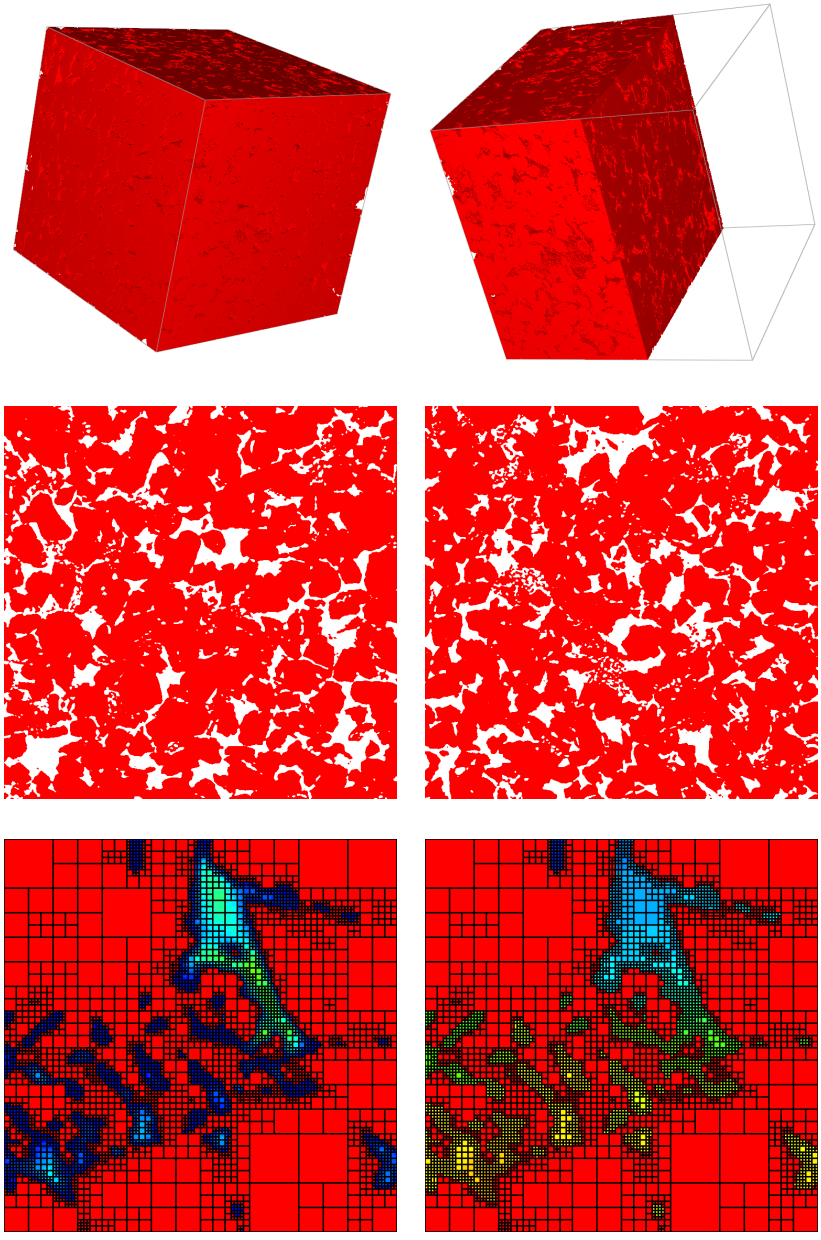


Figure 3.11: Different views of a computed tomography scan of sandstone carbonate. The centered images show a cut through the voxel geometry in $y-z$-plane and $x-z$-plane. The complex pore structure forms a large number of labyrinths for flow simulations. The bottom images show velocity and pressure drop from bottom to top for a small part of the geometry.

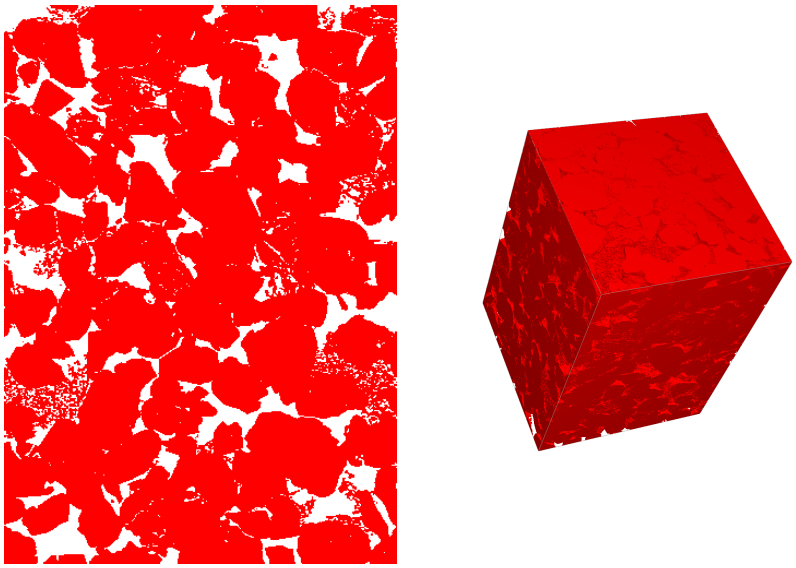


Figure 3.12: Different views of a computed tomography scan of Berea sandstone. The left image shows a cut through the voxel geometry in $y-z$-plane. The right image shows a three dimensional view that reveals the complex nature of the dataset.

Berea Sandstone

The complexity of Berea sandstone is very similar to the previous sandstone carbonate. The interior structure is a network of pores with a low porosity. The dataset is also given as computed tomography scan and well mentioned in literature, see [1] and [2]. The geometry has a resolution of $720\times720\times1024$ voxels with a voxellength of $0.74\mu m$ and a low porosity of 18%. The visual appearance of the geometry and the corresponding solution fields are very similar to the sandstone carbonate. Fig. 3.12 shows different view of the Berea sandstone dataset.

3.9.2 Convergence Analysis

In this section we show that our method to solve the Stokes equations converges to the analytical solution with respect to decreasing cell sizes. We analyze the convergence behavior with respect to the effective permeability.

For periodic arrays of spheres we know the exact permeability with respect to different diameters. We consider the values given in [30] as exact solution.

In our first experiment we used a regular grid represented by the LIR-tree and the tolerance $\epsilon_{\text{sol}} = 10^{-6}$ with a check interval of 100 as termination condition. Sphere diameters from 0.1 to 1.0 and a viscosity of $\eta = 1$ were considered.

Table 3.2 shows the computed permeabilities with respect to different sphere diameters and regular grid sizes and Fig. 3.13 shows the corresponding relative errors to the exact solution. A first order convergence behavior is clearly visible. The two coarse grids are sufficiently accurate to get an intuition of the real effective permeability.

χ	**32**	**64**	**96**	**128**	[**30**]
0.1	1.30564	0.905443	0.886617	0.901776	0.9112
0.2	0.379663	0.377878	0.379734	0.378741	0.3822
0.3	0.20027	0.206559	0.205385	0.207009	0.2081
0.4	0.121694	0.121754	0.122561	0.12226	0.1233
0.5	0.0720238	0.0732844	0.0740099	0.0740499	0.07467
0.6	0.0441904	0.0441682	0.044089	0.0442098	0.04450
0.7	0.0246388	0.0249431	0.0249371	0.0249993	0.02525
0.8	0.0129482	0.0129733	0.0130692	0.013085	0.01320
0.9	0.00595013	0.00607539	0.0060988	0.00609873	0.006153
1	0.00244752	0.002478	0.00249463	0.00248944	0.002520

Table 3.2: Permeabilites for different sphere diameters χ and regular grid sizes from 32^3 to 128^3 voxels. The right table considers the exact solutions.

χ	**32**	**64**	**96**	**128**	[**30**]
0.1	1.31418	0.908414	0.886001	0.896216	0.9112
0.2	0.384652	0.378452	0.377504	0.377788	0.3822
0.3	0.203219	0.206371	0.206478	0.205842	0.2081
0.4	0.12373	0.121762	0.121647	0.121613	0.1233
0.5	0.0725093	0.0736114	0.0737264	0.0741953	0.07467
0.6	0.0443653	0.0437755	0.04366	0.0440202	0.04450
0.7	0.0249182	0.0250236	0.024754	0.0249736	0.02525
0.8	0.0130624	0.0130361	0.0129964	0.0130453	0.01320
0.9	0.00602412	0.00611886	0.00605846	0.00605128	0.006153
1	0.00245001	0.00246954	0.00243877	0.00246438	0.002520

Table 3.3: Permeabilites for different sphere diameters χ and LIR-tree grid sizes from 32 to 128 voxels. The right table considers the exact solutions.

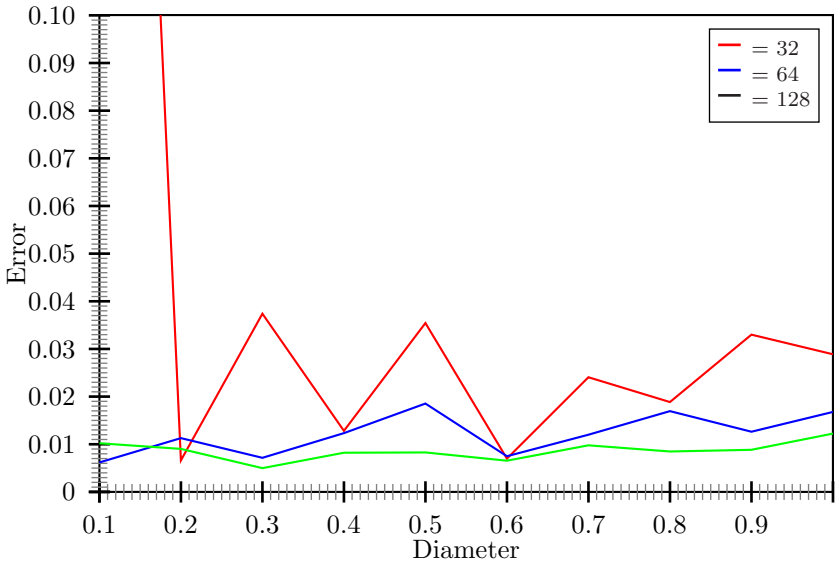


Figure 3.13: Relative errors with respect to the exact solutions for different sphere diameters and grid sizes. First order convergence for a regular grid structure is clearly visible.

In the second experiment we used the general LIR-tree with the additional parameters $\epsilon_{\text{sol}} = \epsilon_{\text{ref}} = \epsilon_{\text{keep}} = 10^{-7}$ for refinement. The voxels per direction are used as limit of space partitioning. The desired number of cells is set according to a regular grid but with a slow growth rate, i.e. $\gamma = \frac{1}{4}$. We used the diameters $\chi \in \{0.2, 0.5, 0.8\}$ to show different behaviors.

Table 3.3 shows the computed permeabilites with respect to different sphere diameters and limits of space partitioning where we used the LIR-tree restricted to $\mathsf{A}^n \subseteq P$ and no difference reduction is used. The permeability matches really well for a small number of cells. Figure 3.14 show the relative permeability error with respect to the number of cells where the LIR-tree with degenerative and non-degenerative refinement is used. The permeability convergences to permeability of the regular grid solution thus converges to the analytical solution with respect to the number of cells. But we observed a non-monotone convergence behavior, see $\chi = 0.2$. The error behavior becomes more stable with increasing diameters, i.e. interface area. In praxis, $2 - 5$ refinements are suggested for an optimal tradeoff between computational effort and numerical accuracy. We observed that the quality of the solution strongly dependents on the refinement approach. The LIR-tree provides a good approximation of the effective permeability even with a very small number of cells depending on the geometric complexity.

The evolution of refinement for a flow around a sphere with diameter $\chi = 0.2$ is shown in Fig. 3.15-3.17. The evolution of refinement for a flow around the fiber geometry is show in Fig. 3.18 and Fig. 3.18. The topology of the flow is revealed more clearly after each refinement. Therefore, researchers can study the behavior of flows inside different materials by visualizations of the tree structure.

To conclude, our discretization of the Stokes equations converges to the analytical solution with respect to the effective permeability. The regular grid

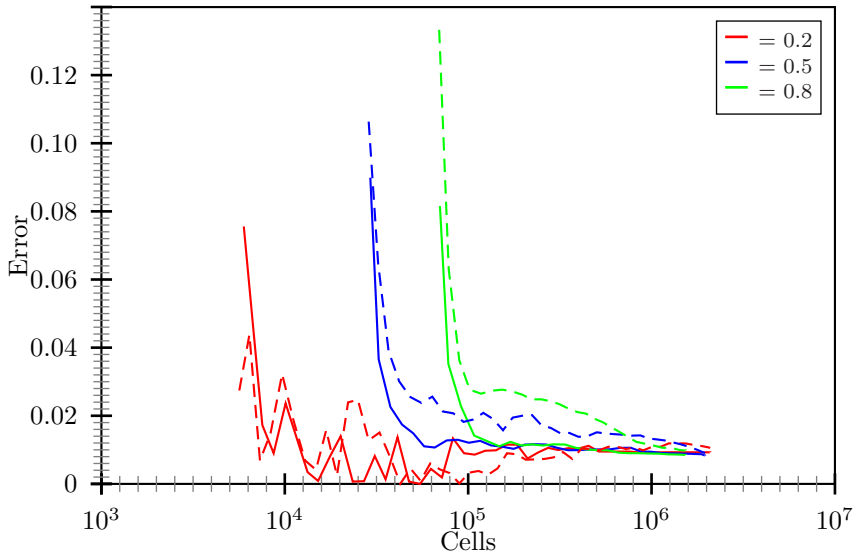


Figure 3.14: Relative permeability error with respect to the number of cells. The growth of cells has been slowed down to point out the behavior. For the dashed curves we used a degenerative refinement while for the solid curves we used non-degenerative refinement.

and the LIR-tree provide a first-order convergence with a good approximation using a small number of cells. The solution based non-degenerative refinement can be used to reveal the topology of flows and increases the numerical accuracy where it is needed. Moreover, our method provides a maximum of accuracy for a specified number of cells that are allowed to be used.

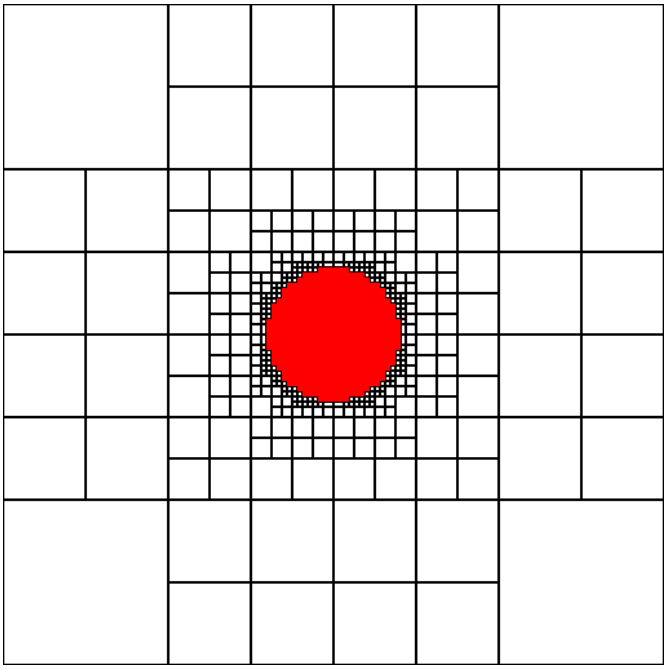
3.9.3 Convergence Speed

In this section we consider the number of iterations that are required to solve the Stokes equations. The number of iterations is dependent on the given geometry and the number of cells. Therefore, we restrict to the sphere and fiber geometry and compare the regular grid to the LIR-tree under grid refinement.

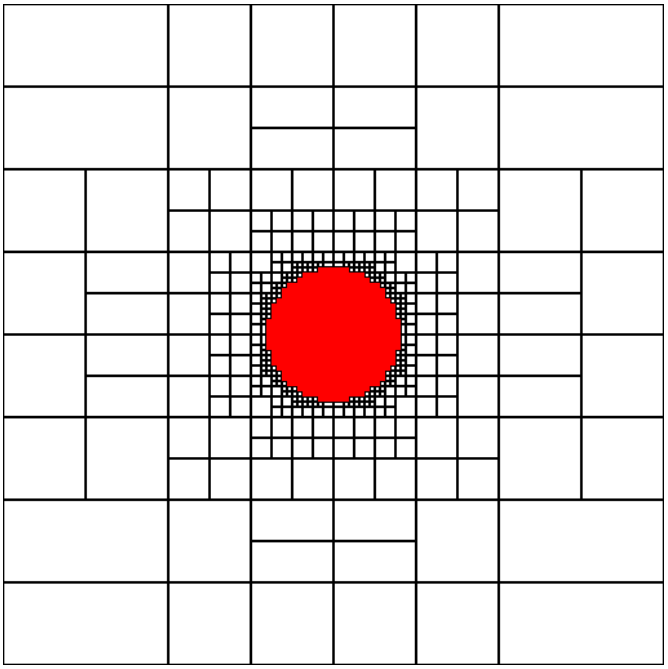
First, we consider a sphere with diameter $\chi = 0.5$ and use the computations with the same parameters from the previous section. We compare the regular grid against the LIR-tree discretization with two different limits of space partitioning.

Figure 3.20 shows the convergence speed with respect to the number of iterations. We observed that the LIR-tree discretizations needs far less iterations to solve the problem than the regular grid. The intended effective permeability is reached and the tolerance decreases very fast. This can be explained by fast information traversal through larger cells. The number of cells is almost doubled after each refinement step until the desired number of cells is approximated. Notice that the tolerance may be increased by an order of magnitude after refinement due to the different linear system.

In the second experiment we used the fiber geometry introduced in Sec. 3.9.1 with two different resolutions. The termination parameters are $\epsilon_{\text{sol}} = 10^{-5}$,

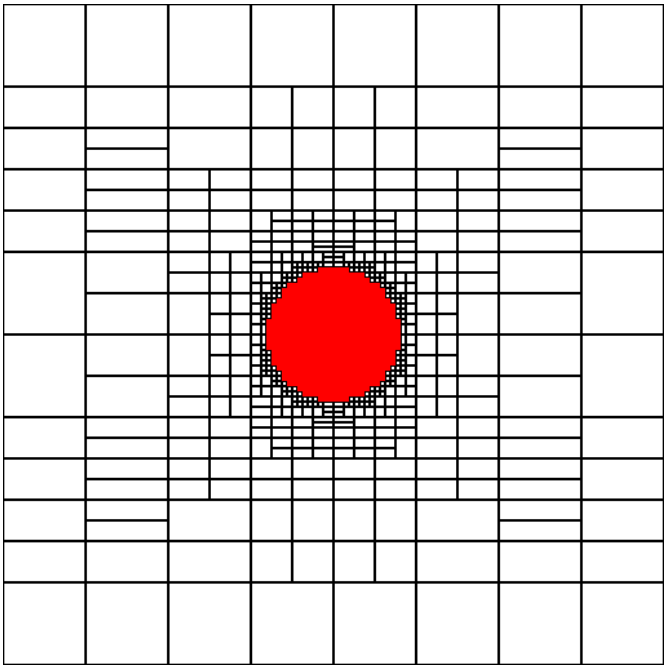


Initial geometry

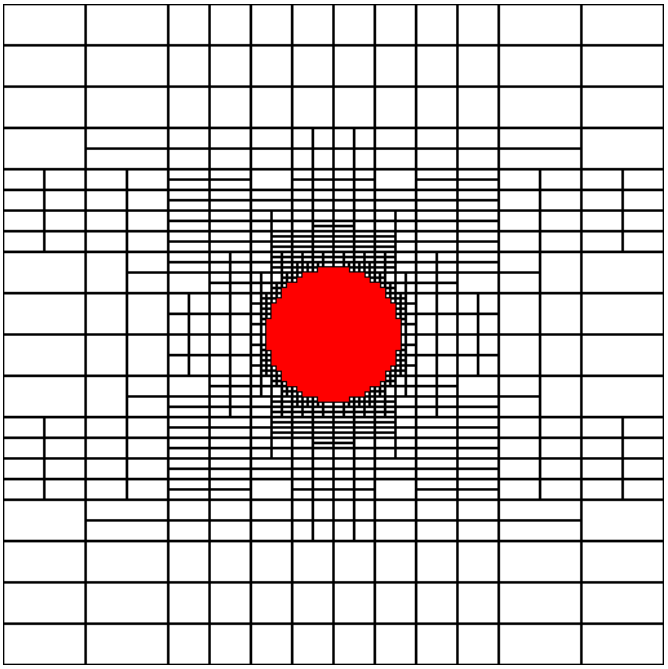


First refinement

Figure 3.15: Flow around a sphere with radius $\chi = 0.2$ in X-direction

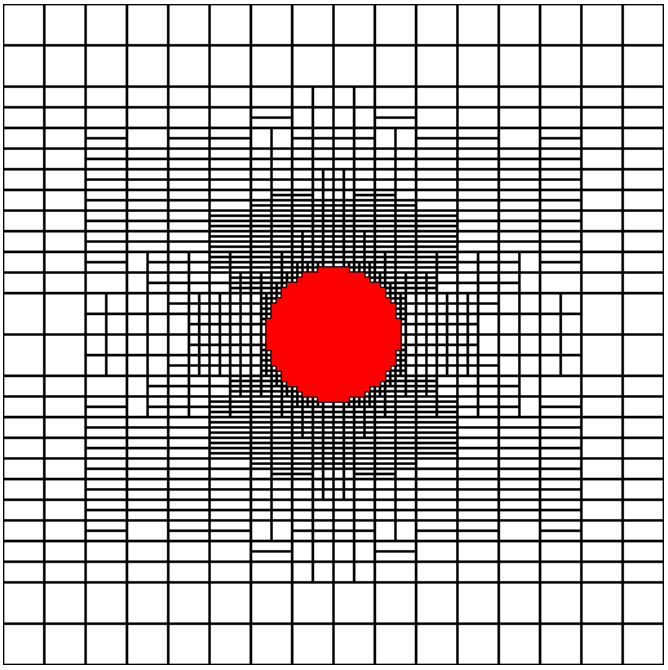


6. Refinement

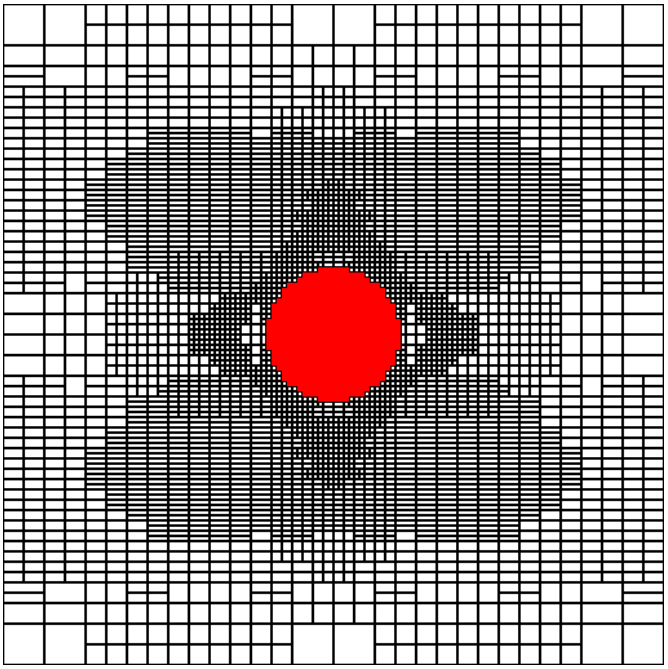


14. Refinement

Figure 3.16: Flow around a sphere with radius $\chi = 0.2$ in X-direction

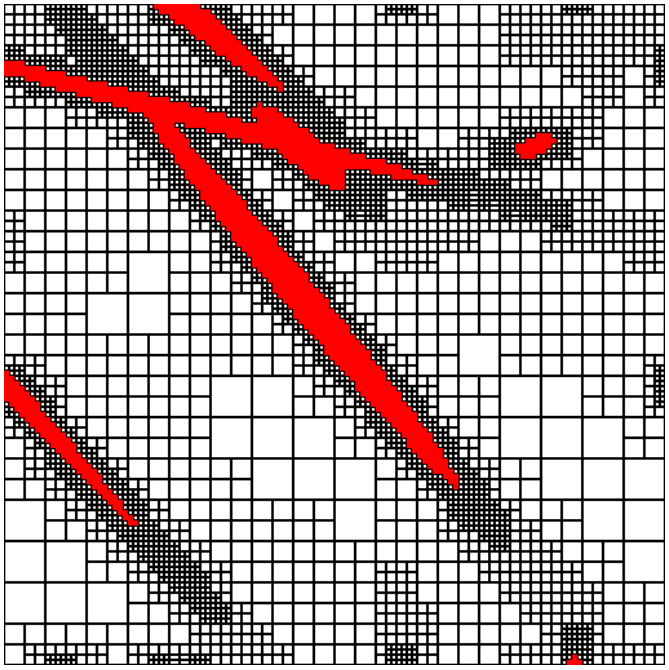


25. Refinement

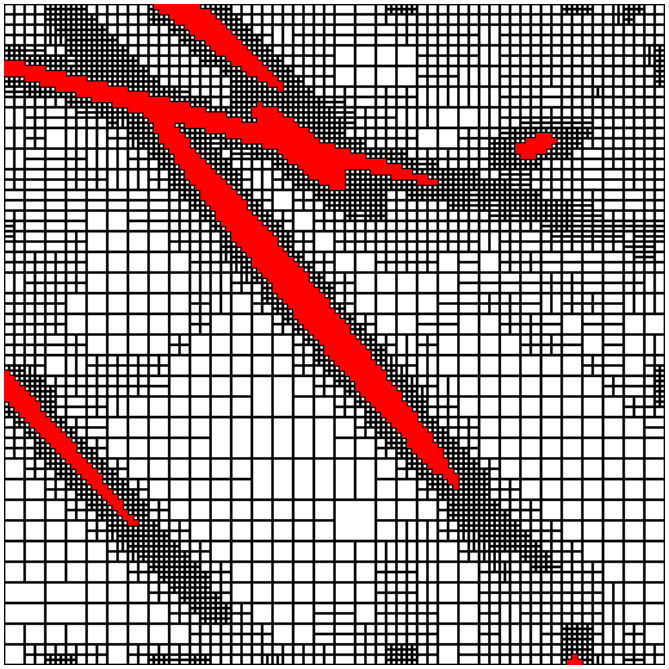


43. Refinement

Figure 3.17: Flow around a sphere with radius $\chi = 0.2$ in X-direction

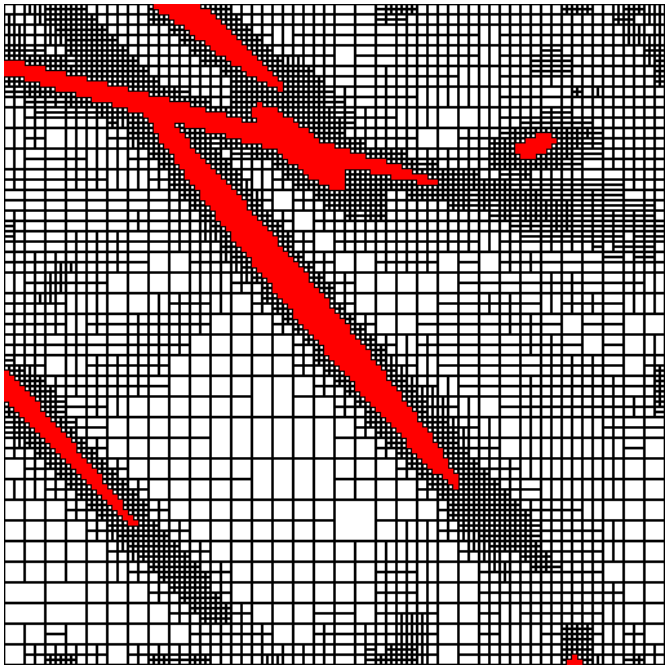


Initial geometry

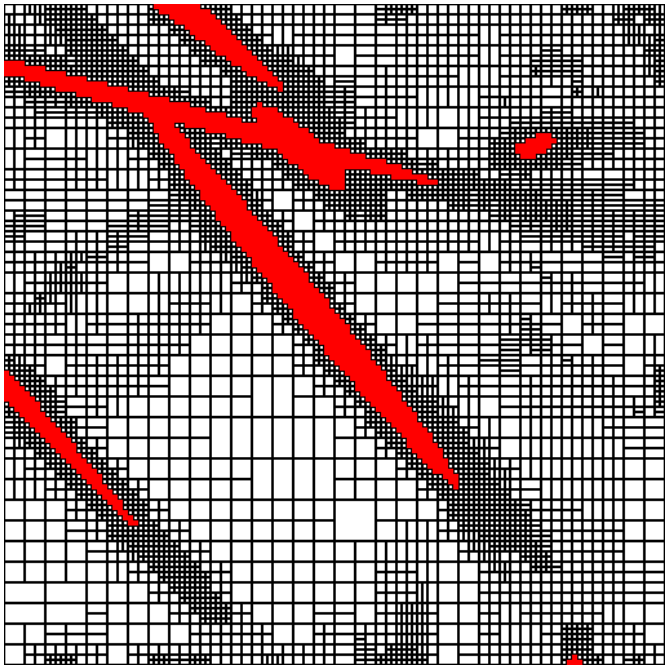


First refinement

Figure 3.18: Flow around a fiber geometry in Z-direction



Second refinement



Third refinement

Figure 3.19: Flow around a fiber geometry in Z-direction

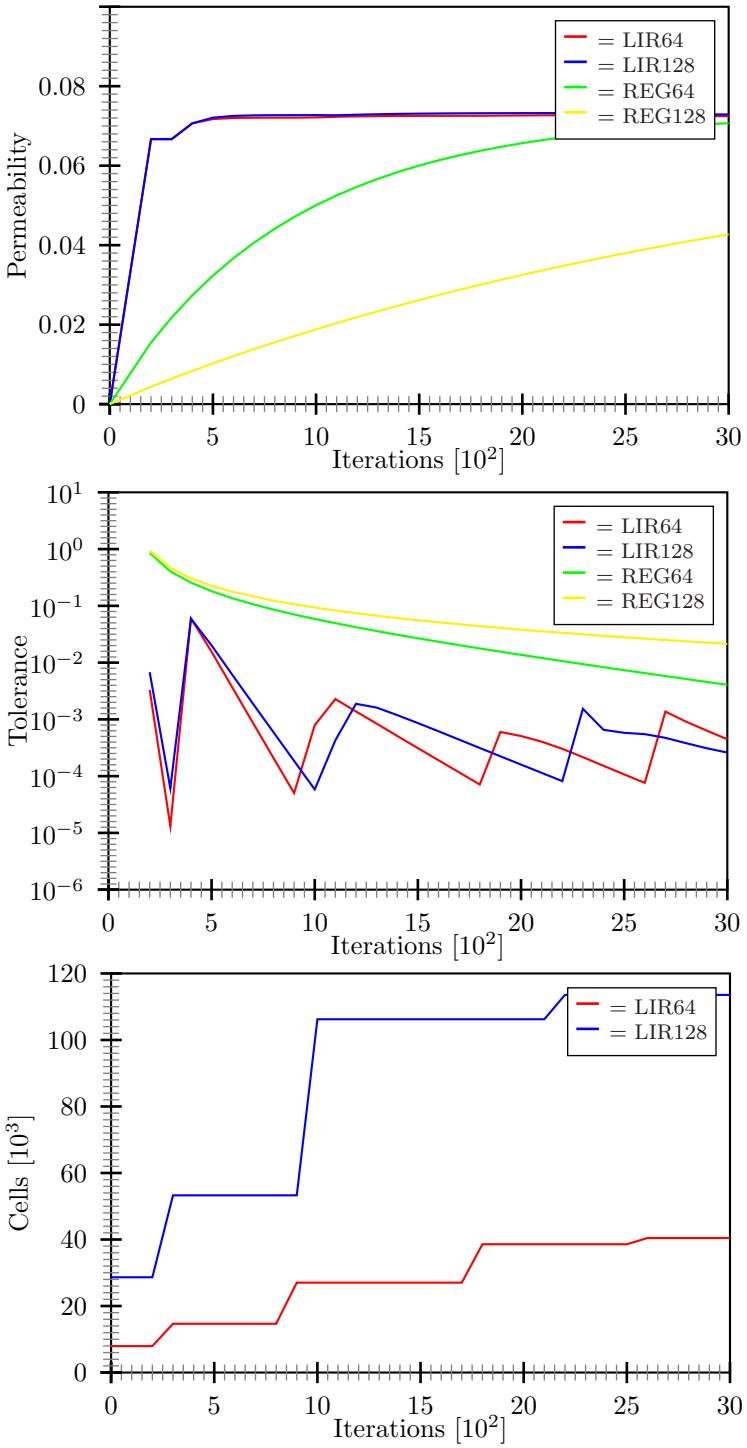


Figure 3.20: Convergence speed of our method for a sphere with $\chi = 0.5$ with respect to the number of iterations. We compared the regular grid and the LIR-tree discretization with 64 and 128 voxels per direction.

Task	Threads	1	2	4	8	16	32	64
Build	Time $[s]$	703	371	195	110	64	44	31
	Speedup	1.0	1.9	3.6	6.4	11	16	22.7
Solve	Time $[s]$	1592	857	426	226	129	77	50
	Speedup	1.0	1.9	3.7	7	12.3	20.7	31.8

Table 3.4: Speedup and time for building and solving with respect to the number of threads. We used the Golem-computer with a 64 core CPU.

$\epsilon_{\text{ref}} = 10^{-3}$ and $\epsilon_{\text{keep}} = 0.05$ with the check interval 50.

Similar to the previous experiment, Fig. 3.21 shows the convergence speed with respect to the number of iterations. With a regular grid 1850 iterations for the lower resolution and 4900 iterations for the higher resolution are needed. The LIR-tree needs 150 before refinement and 350 iterations in total for the lower resolution and 400 iterations for the higher resolution. The evolution of the permeability is more stable and less iterations are needed than for the sphere geometry.

To conclude, the LIR-tree converges much faster than a regular grid with respect to the number of iterations. Of course, this advantages depends on the complexity of the geometry, particularly in the presence of large cell sizes. It is also very important to choose the termination parameter ϵ_{sol} and the refinement parameters ϵ_{ref}, ϵ_{keep} carefully. We found that $\epsilon_{\text{keep}} = 0.05$, $\epsilon_{\text{ref}} = 10 \cdot \epsilon_{\text{sol}}$ are appropriate choices. After each refinement, tolerance increases but drops off fast again. Our approach of threshold determination for difference reduction yield a good approximates for a given desired number of cells.

3.9.4 Parallelization

Parallelization of algorithms has become increasingly important in the last years. There are two kinds of parallelization, these are thread-parallelism on shared memory computers and parallelization across computer networks. In our work we focus primarily on thread-parallelism due to the increasing interest in shared memory computers. But we claim that our methods are applicable on computer networks.

The computed tomography scan of sandstone carbonate is used measure parallel scaleability. The inhomogeneous pore structure causes an inhomogeneous cell distribution and makes parallelization more challenging. We use the Golem-computer due to the high number of CPUs but want to emphasize that the clock rate is reduced automatically when using a higher number of threads.

Table 3.4 and its Fig. 3.22 show that building the initial LIR-tree scales efficiently with respect to the number of threads. We suppose that building scales less than solving due to the intensive memory access caused by geometry analysis.

Table 3.4 and its Fig. 3.22 also show that our method of solving the Stokes equations scales very efficiently with respect to the number of threads. A 64 core CPU increases the performance by a factor of almost 32. We suppose that this behavior can be explained by the short memory distances for neighborhood retrieval.

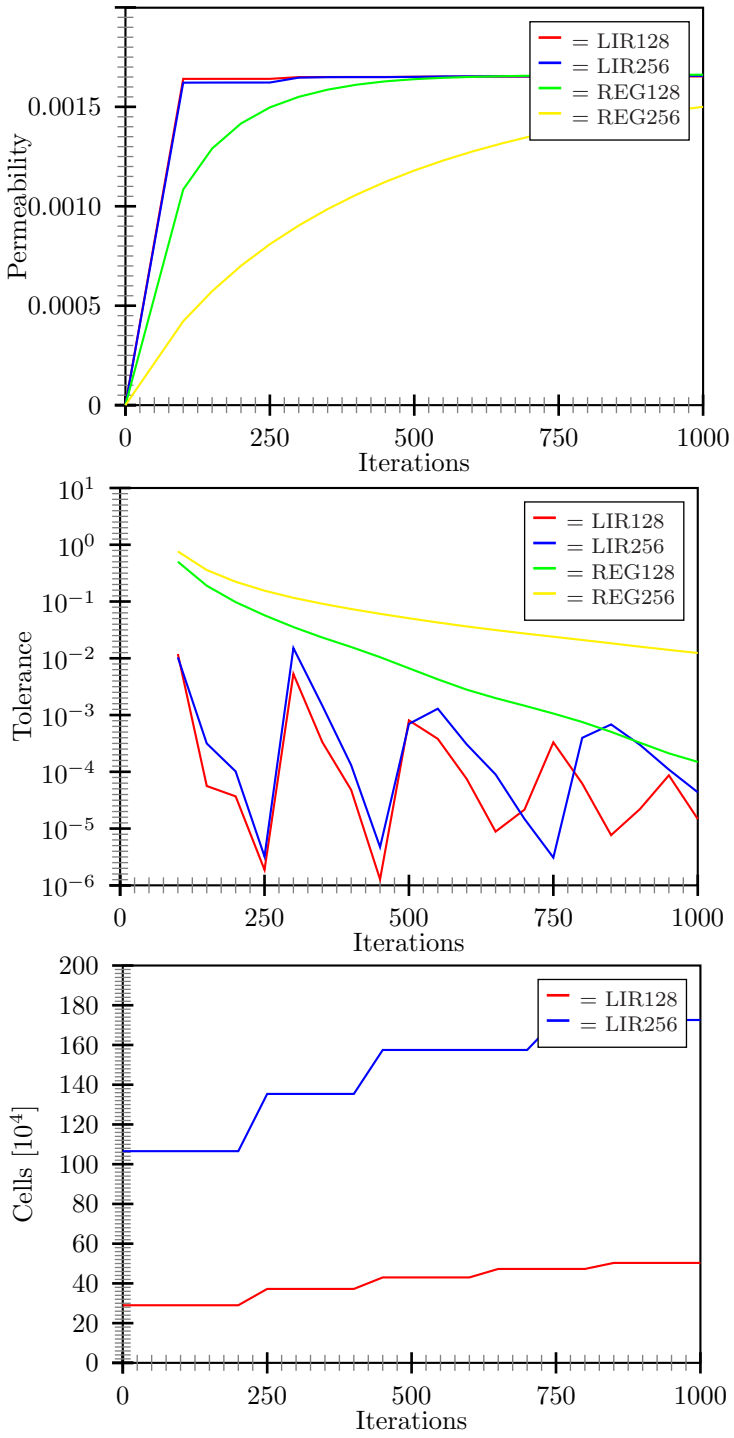


Figure 3.21: Convergence speed of our method for a fiber geometry with respect to the number of iterations. We compared the regular grid and the LIR-tree discretization for 128 and 256 voxels per direction.

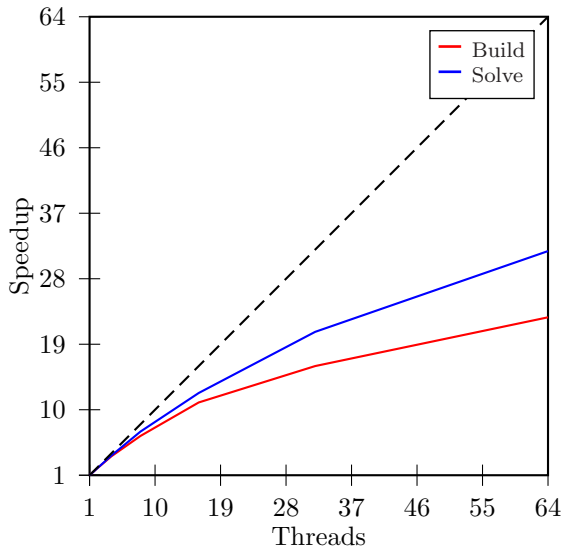


Figure 3.22: Speedup for building and solving with respect to the number of threads.

To conclude, the presented methods scales very efficiently with respect to the number of threads even for inhomogeneous geometries. Thus, we are prepared for large scale datasets.

3.9.5 Speed

The performance of a solver is a key factor in the context of big data. The total computational time can be used to compare our method to other approaches rather than comparing the number of iterations. Different datasets are applied to three other solvers mentioned in Sec. 3.2. In addition, we describe the composition of the computational time for our method.

In that experiment we used the sandstones and the fiber geometry. The Hermes-computer with 8 threads, the Golem-computer with 16 threads and the Hannover-computer with 24 threads serve as computational resources for the three other available solvers and our method. Termination conditions are $\epsilon_{\text{SOL}} = 10^{-4}$ for the fiber geometry and $\epsilon_{\text{SOL}} = 10^{-3}$ for the sandstone geometries. The sandstones are very challenging datasets for our method due to the complex labyrinth structure. Numerous thin tunnels prevent the generation of large cells. Thus, we can not profit from fast spatial information propagation. In that case we have to show that our method has comparable runtimes, i.e. negligible overhead caused by the LIR-tree.

Table 3.5 shows the comparison of the different solvers with respect to respect to the main permeability, time and memory consumption and time after reaching the termination condition. The permeabilities are very close for the fiber geometry. Since the EJ and EFV solvers work on the same linear system the results have to be almost equal. The LB solver has the biggest runtime for both geometries. The EJ solver and our method have a very low runtime for the fiber geometry due to the large regions with same material properties. Our

Geometry	Aspect	EJ	EFV	LB	LIR
Fibers	PermXX $[10^{-3}m^2]$	1.706	1.705	1.697	1.692
	Runtime [min]	19	90	330	3
	Memory [GB]	1.67	2.24	12	$0.10 - 0.27$
Sandstone	PermZZ $[10^{-13}m^2]$		7.665	8.013	8.27
	Runtime [min]		120	390	150
	Memory [GB]		37	168	$3.3 - 10$
Berea	PermZZ $[10^{-13}m^2]$	1.368			1.349
	Runtime [min]	540			137
	Memory [GB]	43			$2.9 - 9.8$

Table 3.5: Comparison of different solvers with respect to permeability in flow-direction, time and memory consumption.

Geometry	Retrieval	Averaging	System	Gauß
Fibers	50	29	14	7
Sandstone	56	23	14	7

Table 3.6: Composition of computational time in percentage for fibers and sandstone carbonate.

method can profit from the small number of cells and a low iteration count. These results hold for high porosity geometries but are different for low porosities. In that context the EFV solver and our methods have the lowest runtime. The permeabilites differ more for the sandstone carbonate due to the lax termination condition. Although the geometry analysis is not able to create a major number of large cells our method can compete with the EFV solver. This is a result of sophisticated optimizations and the use of precomputed tensors and block linear systems for regular neighborhoods.

The total computational time of our method is made of different tasks. These are in order

1. Building the initial tree structure based on geometric analysis
2. Iterative solving and refinement
   - Neighborhood retrieval
   - Averaging neighbor values
   - Formulation of local linear systems
   - Solving of the local linear systems by Gauß algorithm
3. Spline based construction of a regular grid and write-out to a destination

For most geometries the distribution of time consumption is almost equal. Table 3.6 shows the composition of the computational time for solving and refining while solving the two different datasets. Neighborhood retrieval is the most expensive task followed by the determination of representative neighbor values. The formulation of block linear systems and its solution by an adjusted

Geometry	LIR-tree	Neighborhood	Variables
Fibers	12MB	171MB	58MB
Sandstone	555MB	6.5GB	2.2GB

Table 3.7: Composition of memory requirements.

Gauß-algorithm takes the least amount of time. Thus, it is very beneficial to eliminate neighbor retrieval by precomputed neighborhoods.

To summarize the computational speed analysis, the presented method is very fast and has its strengths in high porosity geometries where it can exploit fast information traversal through large cells. In addition, the smaller number of cells compared to regular grids decreases the runtime per iteration. In low porosity environment our method still has similar runtime requirements due to intense optimizations. The memory requirements are low since we do not store values at solid voxels.

3.9.6 Memory

Memory efficiency is an important topic due to the cubic increasing number of cells. The size of local memory is a limiting factor for large geometries. Again, we use the sandstone carbonate and the fiber dataset to compare the memory requirement to other solvers. We also state the general memory requirements of our method.

The description of our method in chapter 3 shows that we just use the cell centered pressure and face centered velocity variables. We do not need any gloabel auxiliary variables by using the block Gauß-Seidel algorithm. The method does not require additional memory for threaded-parallelization by using the parallelization scheme described in sec. 3.6. The memory layout described in sec. 2.6.3 allows a faster neighbor retrievals due to the short memory distance.

Table 3.5 shows the memory requirements for different methods when solving the Stokes equations for the examples mentioned in Sec. 3.9.5. Our method needs significantly less memory than the others. In a high porosity geometry the LIR-tree needs up to 20 times less memory whereas in a low porosity geometry the LIR-tree needs 10 times less memory. The two different memory specification for the LIR-tree arises from the optional neighborhood storage that increases the memory consumption by a factor of three.

Table 3.7 show the distribution of memory consumption for our method. On average, the LIR-tree itself needs 20% of the total memory. This is due to the memory layout described in Fig. 2.9. The most memory consuming part of our method is to store the cell neighborhoods. Since our implementation allows to enable and disable that feature, the user can choose a memory requirement and computational speed trade-off.

In almost all geometries our method needs much less memory compared to the other methods. On average, our implementation needs 10 times less memory and the user can decide to sacrifice more memory for speed. In high porosity geometries our methods provides the best compression ratios.

Chapter 4

Conclusions

In this work we considered two major objects. The development of a memory efficient and numerically suited representation of large voxel geometries and the application in computational fluid dynamics to solve the Stokes equations.

We presented a novel space partitioning system based on a ternary alphabet. Recursive and dimensional application to systems of functions leads to a tree structure that combines the advantages of the Octree and the KD-tree without having their disadvantages. This is also achieved by using different types of look-up tables and compile-time generated data structures.

We compared the LIR-tree to the Octree and KD-tree and observed that the LIR-tree needs at most half the number of cells in a three dimensional context. This is due to the fact that the LIR-tree is a generalization and can use more sophisticated methods to analyze and decompose the geometry. The LIR-tree has the interior cell complexity of the Octree and the leaf complexity of the KD-tree. Numerous experiments showed that in a three dimensional context most of the partitioned cells have 2-4 children. For convergence analysis it makes sense to take the number of cells into account in addition to the spatial length.

In the second part, we introduced a novel approach of variable arrangement for pressure and velocity to solve the Stokes equations. The basic idea of our method is to arrange variables in a way such that each cell is able to satisfy a given physical law. We formulate a small local linear system where the matrix is called block. These local linear systems describe the momentum and mass conservation and allows to include no-slip boundary condition by equation replacement. Since the blocks are invertible the Gauß-Seidel algorithm can be used to solve the global block linear system. This formalism makes it very easy to discretize and solve partial differential equations considered in computational fluid dynamics. In addition, we presented several approaches of refinment to increase the numerical accuracy where it is needed. The threshold determination attempts to find an optimal discretization with respect to a given number of cells.

Our method of variable arrangement converges to the analytical solution with respect to the effective permeability. This is valid both for the regular grid and the LIR-tree discretization. The LIR-tree allows to specify a given number of cells that can be used for the computation. A higher number of cells lead to a better accuracy but requires more iterations and memory. Therefore, the user can set a trade-off between computational effort and numerical accuracy.

The LIR-tree discretization converges much faster than a regular grid with respect to the number of iterations in the presence of large cells. The termination parameters have to be chosen carefully to avoid unnecessary computations. After each refinement of the LIR-tree the tolerance increases for the first subsequent iterations and then decreases fast again. Histogram based threshold determination allows to specify a desired number of cells such that our approach of refinement optimizes with respect to that restriction. The specified number of cells can be reached after a few iterations.

The presented methods scale efficiently in the context of parallel computing and allows to deal with large datasets. The parallelization scheme has a low number of global synchronization where different threads can work on different iterations.

In terms of computational speed, our method is very fast and has its strengths in high porosity geometries where it can exploit the fast information traversal through large cells. The smaller number of cells compared to regular grids decreases the runtime per iteration. In low porosity environment our method still has similar runtime requirements due to intense optimizations. The use of precomputed block tensors and matrices allows to eliminate introduced overhead in regular neighborhoods.

In almost all geometries our method needs much less memory compared to the other methods. On average, our implementation needs 10 times less memory and the user can decide to sacrifice more memory for speed. In high porosity geometries our methods provides the best compression ratios.

Lebenslauf

Persönliche Daten

Vor- und Zuname:	Sven Linden
Anschrift:	Buchenlochstraße 43
	67663 Kaiserslautern
E-Mail:	linden@itwm.fhg.de
Geburtsdatum und -ort:	17.08.1984 in Wittlich
Familienstand:	ledig

Schulische Ausbildung

1991–1995	Grundschule Büchel
1995–2001	Realschule Cochem
2001–2004	Technisches Gymnasium Wittlich

Studium

2005–2008	**Bachelor of Science**, TU Kaiserslautern
Schwerpunkt:	Computergrafik
Titel:	Flächenextraktion aus zeit-dynamischen PAR3-EGFP getaggten Volumina des Auges von Danio rerio
2008–2010	**Master of Science**, TU Kaiserslautern
Schwerpunkt:	Geometrische Modellierung und Technomathematik
Titel:	Hexaeder- und Tetraederunterteilungen für die Simulation von Flüssigkeiten

Berufliche Erfahrungen

2007–2013	Studentische Hilfskraft, Fraunhofer ITWM
	Software Entwicklung für GeoDict (GUI, Visualisierung)
seit 2013	Studentische Hilfskraft, Math2Market GmbH
	Software Entwicklung für GeoDict (GUI, Visualisierung)
2008–2008	Studentische Hilfskraft, TU Kaiserslautern
	Betreuer der Vorlesung Software Entwicklung 2

Bibliography

[1] H. Andrä, N. Combaret, J. Dvorkin, E. Glatt, J. Han, M. Kabel, Y. Keehm, F. Krzikalla, M. Lee, C. Madonna, M. Marsh, T. Mukerji, E. H. Saenger, R. Sain, N. Saxena, S. Ricker, A. Wiegmann, and X. Zhan. Digital rock physics benchmarks—Part I: Imaging and segmentation. *Computers & Geosciences*, 50(0):25 – 32, 2013. Benchmark problems, datasets and methodologies for the computational geosciences.

[2] H. Andrä, N. Combaret, J. Dvorkin, E. Glatt, J. Han, M. Kabel, Y. Keehm, F. Krzikalla, M. Lee, C. Madonna, M. Marsh, T. Mukerji, E. H. Saenger, R. Sain, N. Saxena, S. Ricker, A. Wiegmann, and X. Zhan. Digital rock physics benchmarks—part II: Computing effective properties. *Computers & Geosciences*, 50(0):33 – 43, 2013. Benchmark problems, datasets and methodologies for the computational geosciences.

[3] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[4] J. L. Bentley. K-d trees for semidynamic point sets. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 187–197, New York, NY, USA, 1990. ACM.

[5] H. H. Chen and T. S. Huang. A Survey of Construction and Manipulation of Octrees. *CVGIP*, 43(3):409–431, September 1988.

[6] L. Cheng and A. Wiegmann. Finite Volume Flow Solver on 3D Images. 6th International Conference on Porous Media & Annual Meeting, Milwaukee, USA, 27-30 May, 2014.

[7] L. Cheng, A. Wiegmann, and S. Rief. SIMPLE-FFT for flow computations in low porosity μCT images. 5th International Conference on Porous Media & Annual Meeting, Czech Republic, 21-24 May, 2013.

[8] W. K. Chow and Y. L. Cheung. Comparison of the Algorithms Piso and Simpler for Solving Pressure-Velocity Linked Equations in Simulating Compartmental Fire. *Numerical Heat Transfer Part A - Applications*, 31:87–112, January 1997.

[9] C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Balanced Aspect Ratio Trees: Combining the Advantages of k-d Trees and Octrees. *J. Algorithms*, 38(1):303–333, 2001.

[10] J. H. Ferziger and M. Perić. *Computational Methods for Fluid Dynamics*. Springer London, Limited, 2002.

[11] M. S. Floater, T. Lyche, M. L. Mazure, K. Mørken, and L. L. Schumaker, editors. *Mathematical Methods for Curves and Surfaces - 8th International Conference, MMCS 2012, Oslo, Norway, June 28 - July 3, 2012, Revised Selected Papers*, volume 8177 of *Lecture Notes in Computer Science*. Springer, 2014.

[12] D. R. Forsey and R. H. Bartels. Hierarchical B-spline refinement. *SIGGRAPH Comput. Graph.*, 22(4):205–212, June 1988.

[13] C. Giannelli, B. Jüttler, and H. Speleers. THB-splines: The truncated basis for hierarchical splines. *Computer Aided Geometric Design*, 29(7):485 – 498, 2012. Geometric Modeling and Processing 2012.

[14] I. Ginzburg and K. Steiner. Lattice Boltzmann model for free-surface flow and its application to filling process in casting . *Journal of Computational Physics*, 185(1):61 – 99, 2003.

[15] F. H. Harlow and J. E. Welch. Numerical Calculation of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface. *Physics of Fluids*, 8(12):2182–2189, 1965.

[16] D. Hietel, K. Steiner, and J. Struckmeier. A finite-volume particle method for compressible flows. *Mathematical Models and Methods in Applied Sciences*, 10(09):1363–1382, 2000.

[17] G. M. Hunter. *Efficient computation and data structures for graphics.* PhD thesis, Princeton, NJ, USA, Princeton, NJ, USA, 1978.

[18] T. Inamuro. Lattice Boltzmann methods for viscous fluid flows and for two-phase fluid flows . *Fluid Dynamics Research*, 38(9):641 – 659, 2006. Recent Topics in Computational Fluid Dynamics.

[19] C. L. Jackins and S. L. Tanimoto. Oct-trees and Their Use in Representing Three-Dimensional Objects. *CGIP*, 14(3):249–270, November 1980.

[20] C. L. Jackins and S. L. Tanimoto. Quad-Trees, Oct-Trees, and K-Trees: A Generalized Approach to Recursive Decomposition of Euclidean Space. *PAMI*, 5(5):533–539, September 1983.

[21] D. S. Jang, R. Jetli, and S. Acharya. Comparison of the piso, simpler, and simplec algorithms for the treatment of the pressure-velocity coupling in steady flow problems. *Numerical Heat Transfer*, 10(3):209–228, 1986.

[22] D. Jeulin. Morphology and effective properties of multi-scale random sets: A review. *Comptes Rendus Mécanique*, 340(4–5):219 – 229, 2012. Recent Advances in Micromechanics of Materials.

[23] T. Kanit, S. Forest, I. Galliet, V. Mounoury, and D. Jeulin. Determination of the size of the representative volume element for random composites: statistical and numerical approach. *International Journal of Solids and Structures*, 40(13–14):3647 – 3679, 2003.

[24] S. Linden, A. Wiegmann, and H. Hagen. The LIR Space Partitioning System Applied to Cartesian Grids. In Floater et al. [11], pages 324–340.

[25] R. A. Lorentz. *Multivariate Birkhoff Interpolation*. Lecture Notes in Mathematics. Springer, 1992.

[26] W. Niethammer. The SOR method on parallel computers. *Numerische Mathematik*, 56(2-3):247–254, 1989.

[27] S. V. Patankar. *Numerical heat transfer and fluid flow*. Taylor & Francis, 1980.

[28] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[29] H. Samet and A. Kochut. Octree approximation and compression methods. In *3DPVT*, pages 460–469. IEEE Computer Society, 2002.

[30] A. S. Sangani and A. Acrivos. Slow flow through a periodic array of spheres. *International Journal of Multiphase Flow*, 8(4):343 – 360, 1982.

[31] K. Schladitz, S. Peters, D. R. Bitzer, A. Wiegmann, and J. Ohser. Design of acoustic trim based on geometric modeling and flow simulation for nonwoven. *Computational Materials Science*, 38(1):56 – 66, 2006.

[32] G. Thömmes, J. Becker, M. Junk, A. K. Vaikuntam, D. Kehrwald, A. Klar, K. Steiner, and A. Wiegmann. Numerical investigation of a combined lattice Boltzmann-level set method for three-dimensional multiphase flow. *International Journal of Computational Fluid Dynamics*, 23(10):687–697, 2009.

[33] R. F. Tobler. The rkd-Tree: An Improved kd-Tree for Fast n-Closest Point Queries in Large Point Sets. *in Proceedings of Computer Graphics International 2011 (CGI 2011)*, 2011.

[34] J. P. Van Doormaal and G. D. Raithby. Enhancements of the SIMPLE Method for Predicting Incompressible Fluid Flows. *Numerical Heat Transfer*, 7(2):147–163, 1984.

[35] A. Wiegmann. GeoDict: Geometric Models and PreDictions of Properties. http://www.geodict.com, 2001. Virtual material laboratory.

[36] A. Wiegmann. *Computation of the permeability of porous materials from their microstructure by FFF-Stokes*. Fraunhofer ITWM, 2007.

[37] A. Wiegmann and K. P. Bube. The Explicit-Jump Immersed Interface Method: Finite Difference Methods For PDE With Piecewise Smooth Solutions. *SIAM J. Numer. Anal*, 37:827–862, 1997.

[38] A. Wiegmann and A. Zemitis. *EJ-HEAT: A fast explicit jump harmonic averaging solver for the effective heat conductivity of composite materials*. Fraunhofer ITWM, 2006.