

**Mesh Refinement for Parallel Adaptive FEM  
Theory and Implementation**

**Von der Fakultät Mathematik und Physik der  
Universität Stuttgart zur Erlangung der Würde eines  
Doktors der Naturwissenschaften (Dr. rer. nat.)  
genehmigte Abhandlung**

**Vorgelegt von**

**Martin Jürgen Alkämper**

**aus Duisburg**

<b>Betreuer:</b>	<b>Prof. Dr. Kunibert Siebert</b>
<b>Hauptberichter:</b>	<b>Prof. Dr. Dominik Göttsche</b>
<b>1. Mitberichter:</b>	<b>Prof. Dr. Alfred Schmidt</b>
<b>2. Mitberichterin:</b>	<b>Prof. Dr. Miriam Mehl</b>

**Tag der mündlichen Prüfung: 11.07.2019**

**Institut für Angewandte Analysis und Numerische  
Simulation der Universität Stuttgart**

**2019**

To Simon Alkämper, who was born,  
while I was in the middle of this work  
and could already sing, when I finished.

# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
1.1	Grids and Adaptivity . . . . .	7
1.2	Parallel, Adaptive Grids . . . . .	11
1.3	Thesis Outline and Contribution . . . . .	14
<b>2</b>	<b>Adaptive Meshes</b>	<b>19</b>
2.1	Some Notations and Concepts . . . . .	20
2.2	Isometric Refinement . . . . .	26
2.3	Edge Bisection . . . . .	28
2.3.1	Longest Edge Bisection . . . . .	31
2.3.2	Newest Vertex Bisection . . . . .	33
2.4	Refinement Closure . . . . .	38
<b>3</b>	<b>Weak Compatibility</b>	<b>45</b>
3.1	NVB and Induced Refinement . . . . .	46
3.2	Compatibility Conditions . . . . .	52
3.2.1	Weak Compatibility Condition . . . . .	53
3.2.2	Strong Compatibility Condition . . . . .	54
3.2.3	Sufficiency of Weakest Compatibility Condition . . . . .	61
3.3	Renumbering Algorithm . . . . .	65
3.3.1	Vertex Sets . . . . .	67
3.3.2	Vertex Orderings . . . . .	69
3.4	Numerical Experiments . . . . .	71
3.4.1	Mesh Quality . . . . .	73
3.4.2	Threshold Study . . . . .	74

3.4.3	Conforming Closure . . . . .	80
3.4.4	Summary and Recommendations . . . . .	85
3.4.5	Algorithm Complexity . . . . .	86
3.4.6	Arbitrary Dimensional Grid . . . . .	86
<b>4</b>	<b>Parallel Refinement</b>	<b>90</b>
4.1	Distributed Newest Vertex Bisection . . . . .	91
4.2	Communication Bounds . . . . .	93
4.2.1	Generic Analysis . . . . .	93
4.2.2	Analysis Using a Special Partitioning . . . . .	98
4.3	Weakly Compatible Refinement Propagation . . . . .	109
4.4	Numerical Experiments . . . . .	111
4.4.1	Verification of Theoretical Results . . . . .	112
4.4.2	Strong Scaling Experiments . . . . .	114
<b>5</b>	<b>Some Implementational Aspects</b>	<b>119</b>
5.1	DUNE-ALUGrid . . . . .	120
5.1.1	Cubes and Quadrilaterals . . . . .	122
5.1.2	Tetrahedrons and Triangles . . . . .	124
5.1.3	Surface 2d Grids . . . . .	126
5.2	DUNE-ACFEM - Adaptive Convenient Finite Elements . . . . .	127
5.3	$n$ Dimensional Simplex Grid . . . . .	130
<b>6</b>	<b>Case Study: Using Dune-ACFem for Non-smooth Minimization of Bounded Variation Functions</b>	<b>135</b>
6.1	Problem Formulation . . . . .	136
6.1.1	Discretization . . . . .	138
6.1.2	Primal-Dual Algorithm . . . . .	141
6.2	Implementation Details . . . . .	144
6.2.1	Image Read-In and Noise Simulation . . . . .	144
6.2.2	Implementation of the Primal-Dual Algorithm . . . . .	145
6.3	Numerical Experiments . . . . .	148
6.3.1	Local Refinement . . . . .	148
6.3.2	Comparison of Discrete Spaces . . . . .	149

6.3.3	Image Denoising . . . . .	153
6.3.4	Strong Scaling . . . . .	154

# Abstract

We investigate parallel adaptive grid refinement and focus in particular on hierarchically adaptive, parallel and conforming simplicial grids that use Newest Vertex Bisection (NVB) as their refinement strategy.

One challenge of NVB is its applicability to arbitrary simplex grids, which is not possible with the current compatibility condition. We define a novel, more natural weak compatibility condition for the initial grid and show that using this condition the iterative refinement algorithm terminates using NVB. We design an algorithm to relabel an arbitrary  $d$ -dimensional simplicial grid to fulfil this weak compatibility condition. The algorithm is of complexity  $O(n)$ , where  $n$  is the number of elements in the grid.

We also consider NVB on partitioned grids for parallel computing. Another challenge is that refinement may propagate over partition boundaries. This is resolved by adding an outer loop to the refinement algorithm, that requires global communication. We prove that the amount of global communication needed and the number of outer iterations in the refinement propagation to reach a conforming situation is bounded.

We extend the grid manager DUNE-ALUGRID to provide parallel, adaptive, conforming 2d grids. Furthermore we develop the software package DUNE-ACFEM which is able to conveniently describe mathematical problems within efficient C++ code. We demonstrate the utility of DUNE-ACFEM and DUNE-ALUGRID at the problem of noise removal on images with adaptive finite elements.

# Zusammenfassung

Wir erforschen paralleles adaptives Verfeinern von Gitter und legen dabei besonderes Augenmerk auf hierarchisch adaptive, parallele, konforme Simplexgitter, die mittels Newest Vertex Bisection (NVB) verfeinert werden.

Eine Herausforderung ist die Anwendung von NVB auf beliebige Simplexgitter. Mit der derzeitigen Standardkompatibilitätsbedingung ist dies nicht möglich. Wir definieren eine neue schwache Kompatibilitätsbedingung auf dem Anfangsgitter und zeigen, dass der Verfeinerungsalgorithmus für NVB unter dieser Bedingung terminiert. Außerdem entwickeln wir einen Algorithmus, der ein beliebiges  $d$ -dimensionales Simplexgitter so umnummeriert, dass die schwache Kompatibilitätsbedingung erfüllt ist. Dieser Algorithmus hat Komplexität  $O(n)$ , wobei  $n$  die Anzahl der Gitterelemente ist.

Ein weiteres Problem von verteilten, konformen, adaptiven Gittern ist, dass Verfeinerung über Prozessorgrenzen kommuniziert werden muss, weswegen eine äußere Schleife mit globaler Kommunikation benötigt wird. Wir zeigen, dass die Anzahl an globalen Kommunikationen und äußeren Iterationen, die für den konformen Abschluss auf verteilten Gittern benötigt werden, beschränkt ist.

Der Gitter-Manager ALUGRID wird auf parallele, adaptive 2d Gitter erweitert. Außerdem wurde das DUNE Modul DUNE-ACFEM mitentwickelt, welches dazu dient mathematische Probleme in C++ Code abzubilden. Die Nützlichkeit von DUNE-ACFEM und DUNE-ALUGRID wird am Beispiel des Bildentrauschen mit adaptiven Finiten Elementen demonstriert.

# Chapter 1

## Introduction

This thesis is concerned with a theoretical and practical investigation of the combination of parallel and adaptive grids. This work mostly elaborates on a certain kind of grid, namely conforming, unstructured, parallel, adaptive simplex grids using Newest Vertex Bisection (NVB) as refinement strategy/procedure, but also discusses hypercubes. We first introduce in Section 1.1 the basic idea of grids, hierarchical refinement and refinement strategies. Section 1.2 extends the concepts to partitioned grids for parallel computing and outlines software aspects, and in Section 1.3 we explain the contribution of this thesis and outline the different chapters.

### 1.1 Grids and Adaptivity

Grids (also meshes, triangulations or tessellations) are the foundation of many discretization schemes for problems given as systems of (partial) differential equations. Examples include Finite-Volumes (FVM), Finite-Differences (FDM) and Finite-Elements (FEM), including Virtual-Elements (VEM) and Discontinuous-Galerkin-Methods (DG-FEM).

The concept of a grid is quite simple. A (bounded) domain  $\Omega \in \mathbb{R}^d$  is split into disjoint polytopal elements and each element consists of lower-dimensional polytopal subelements. As an example in 2d there may be triangles composed of edges, and edges composed of vertices. A grid has to



fulfil certain properties depending on the grid-based method that shall be used. For example, many Finite-Elements require a maximum angle condition [BS01, Chapter 3][Ape99, Chapter II] to converge. Another example is the so-called CFL-condition [CFL28], which bounds the minimum diameter in relation to a time step size.

In conforming meshes every intersection of two grid elements is a full subelement for each of the elements or empty. This implies that there are no hanging nodes, i.e., vertices that are inside or on the boundary of an element, but not a vertex of this element. As a side note, this implies that meshes that allow arbitrary shapes can be seen as by design conforming. Any vertex of a neighbouring element can always be interpreted as a vertex of the element itself by simply increasing its number of vertices and so changing its polytopal type (e.g., from simplex to cube).

Unstructured grids are able to represent complex geometries, as they do not impose any restriction on the grid layout. An advantage of unstructured grids using FV/FEM in contrast to FD, which are usually defined on structured grids, is the existence of a variety of error estimators. An often-used simplification is that the grid only contains elements of the same type (e.g., cubes, simplices). We will mostly focus on grids that contain only simplices. As subelements of simplices are lower-dimensional simplices, conformity then implies that adding a vertex to the grid requires a subtriangulation of each adjacent simplex. The ability to introduce additional vertices/elements (refinement) and to remove existing vertices/elements (coarsening) is called adaptivity.

The adaptivity process needs to be driven by some kind of error estimator (e.g., [Dör96, CKNS08, BDD04, SS05]), which estimates the error of the discretization with respect to the grid elements. There are two types of error estimators: A priori estimators, that estimate the error based on the given data, and a posteriori estimators, that estimate the error after a solution has been computed and may for example use the residual. Based on an a posteriori estimator a marking strategy then marks elements by their local

estimate for refinement or coarsening. After adaptation of the mesh the process is repeated, which results in the following conceptual loop:

$$\text{SOLVE} \rightarrow \text{ESTIMATE} \rightarrow \text{MARK} \rightarrow \text{ADAPT} \quad (1.1)$$

**SOLVE** describes the process of solving an approximation of the given problem on the current mesh by the chosen method. **ESTIMATE** applies the local a posteriori estimator, which yields an estimate on each element of the grid. **MARK** marks the grid by a given marking strategy (e.g., based on the maximum error or Dörfler marking [Dör96]) and **ADAPT** refines and coarsens the mesh. In this work, we will only investigate the **ADAPT** step and assume the other steps to be given.

Adaption methods for conforming, unstructured simplex meshes include: Newest Vertex Bisection (NVB) [Dör96, CKNS08, GSS14, GHS15, Mit16, AK17], Longest Edge Bisection (LEB) [Riv97, HKK10, ACH<sup>+</sup>15, KKK08, Hor97, Sch97], Red-Green refinement [BBJ<sup>+</sup>97, Gra19, Fre42, Bey00, Bey95], Node-movement and remeshing [Hec12, The18, HS18] or Anisotropic refinement [Ape99, Hua06, POG08, Sch13].

Bisection methods refine a simplex by splitting it into two halves. To achieve conformity they iterate over neighbouring elements and bisect these until all elements are conforming. This is done by bisecting a designated edge called refinement edge. NVB needs a coupling (compatibility) condition between neighbouring elements, as it is defined element-wise and has to guarantee that the refinement edge on the shared face of two neighbouring simplices coincides. As LEB is defined edge-wise, refinement edges from neighbouring simplices are automatically compatible.

“Red” refinement splits a simplex into  $2^d$  smaller simplices using the centres of all edges of the simplex. To achieve conformity it needs a “green” closure afterwards. The combination of red refinement and green closure is called red-green refinement.

We will not consider node-movement, remeshing and anisotropic refine-

ment in this work. Node-movement involves moving vertices to different positions, remeshing involves keeping all vertices in place and creating a completely new grid that has exactly these vertices. Anisotropic refinement can combine any of the above methods but NVB. It chooses anisotropic direction(s) and adjusts the refinement strategy accordingly. As an example, anisotropic refinement can use LEB and calculate the length of an edge using a weighted norm.

The advantage of bisection over red-green refinement is that the arising sequences of triangulations are nested and thus particularly suitable for FEM, where the resulting hierarchical spaces are also nested. An advantage of red-green refinement is that red refinement is isometric in 2 dimensions, so all descendants of an element are similar to the element itself. Node-movement and remeshing are more tailored to cases that include tracking a moving object or front (e.g. [CRW17]) while resolving it exactly.

Both bisection methods and red refinement are hierarchical adaptation strategies. This means that an element is split into disjoint child elements, which cover the whole volume of the element. This results in a (refinement) tree of descendants for each element of the initial grid. Multiple trees are called forest. The leaves of the forest (i.e., elements that do not have children) form the current triangulation and the roots form the initial (or macro) triangulation. We will often call elements of the initial triangulation macro elements.

In contrast to LEB, NVB is a mere topological construct, where refinement simply depends on an ordering of the vertices. LEB uses geometric information to always refine the longest edge, which is why it can be altered to be used in anisotropic refinement. While it is clear for NVB that it only produces a finite number of shape classes, for LEB this is non-trivial and requires some combinatorial effort [ACH<sup>+</sup>15].

The tree of each initial element of NVB in 2 dimensions may be traversed by a space-filling curve, the Sierpinski-curve, so that every two following triangles of the leaf triangulation share a common face. This leads to traversal

based implementations [SBB12]. In 3d this is no longer possible (see [Bad12]), so one has to use quasi-space-filling curves to that aim, or extend triangles as prisms into the third dimension [MB15], which violates our goal of using a single element type. An overview on NVB can be found in Chapter 2 and in [Mit16].

However, one big drawback of NVB so far is its non-applicability to generic unstructured meshes in three (or more) dimensions, as it requires a compatibility condition between neighbouring elements on the initial grid. Traxler and Maubach [Mau95, Tra97] have developed the concept of a strong compatibility condition, which has been generalized by Stevenson [Ste08]. It ensures that every level of uniform refinement is conforming and that there is a bound on the effort of the conforming closure, i.e. the amount of elements that need to be inserted to achieve conformity. In this work, we design a more natural “weak” compatibility condition that can be fulfilled easier.

Standard simplicial mesh generators, such as GMSH [GR09] or TETGEN [Si15], do not guarantee this condition. To overcome this problem in 3d one can follow the idea of [AMP00] and introduce non-standard initial elements. Another approach is to multiply the number of elements by  $\frac{1}{2}(d+1)!$ , while halving each angle  $d$  times, as in [Kos94, Ste08].

## 1.2 Parallel, Adaptive Grids

We additionally want the adaptive, conforming, unstructured, simplex NVB grid to be parallel in a domain decomposition sense. We collect elements into disjoint partitions and distribute the partitions to different processors. In our work we assume that every processor gets one partition and call this process partitioning.

In many discretization schemes the number of unknowns is proportional to the number of elements. This often implies that the computational workload is proportional to the number of elements, which in turn implies that equidistributing the elements into partitions of similar size is a good initial strategy.

The communication on the other hand is often proportional to the size of the partition boundary. So we aim to equidistribute the unknowns (or elements), while minimizing the total partition boundary size. This goal-oriented form of partitioning is called loadbalance. On heterogeneous hardware the matter is more complex and other aspects like memory/cache sizes, communication speed and computation power have to be taken into account (e.g. [KDB02]).

On an adaptive grid additional elements may be introduced or present elements may be removed, which may drastically alter the size of different partitions. So after adaptation a repartitioning may be necessary. The adaptation loop (1.1) looks as follows:

$$\text{SOLVE} \rightarrow \text{ESTIMATE} \rightarrow \text{MARK} \rightarrow \text{ADAPT} \rightarrow \text{LOADBALANCE} \quad (1.2)$$

There are several forms to handle the **LOADBALANCE** step for adaptive meshes. Some use a space-filling curve approach to sort all leaf elements of the mesh [BWG11, BHK07, LWH12], others simply sort the initial elements [ADKN16]. These sorted elements are split into equal blocks of a certain length and the space-filling curve takes care of the connectivity of the mesh. It can be shown, that this approach is quasi-optimal [Zum02] regarding the connectivity of partitions. Also graph partitioning is often used (e.g., open-source partitioners **ZOLTAN** [BCCD12], **METIS** [KK70] and **PT-SCOTCH** [CP08]).

Our grid manager **DUNE-ALUGRID** [Sch99, BDKO06, ADKN16] load-balances by assigning weights to macro elements and distributing the macro elements into partitions according to these weights. The standard weight of a macro element is the number of its current leaves. Partitions in **DUNE-ALUGRID** always contain initial elements and their full refinement tree.

**Mesh-balance and Conformity** Many implementations of parallel adaptive grids use nonconforming hexahedral (or quadrilateral in 2 dimensions) elements with a mesh-balance instead of conformity to acquire a mesh grading (required e.g. for stability estimates [IBG12, GHS15]). A mesh-balance

bounds the difference in number of refinements, when seen from different elements. As an example in 2d, a shared edge of two quadrilaterals may be split from one side, but not from the other. The parallel overhead to enforce this mesh-balance is bounded depending on the structure of the mesh [IBG12].

A similar issue that arises when combining adaptive, conforming meshes with parallelism, is the enforcement of conformity over partition boundaries. In [Zha09] NVB is extended onto partitioned meshes. This method called Distributed NVB executes local refinement on each partition until it is conforming and then communicates the refinement status of the faces. This is repeated until there is no change in the grid. The implementation in [Zha09, LMZ08] is based on ALBERTA [SSH<sup>+</sup>].

Distributed NVB is implemented in the DUNE module DUNE-ALUGRID and the toolbox AMDiS [WLPV15]. The last article contains an estimate on the communication overhead of magnitude  $O(\log P)$ , where  $P$  is the number of partitions in [WLPV15, Sec. 2.4]. We will show, that without additional assumptions on the partitioning this cannot hold (cf. Section 4.2.1). Another implementation of parallel simplex meshes is realized in NETGEN ([Sch97]), which implements the non-standard initial NVB simplices from [AMP00].

Another approach to parallel simplex refinement has been published by Rivara et al. [RCFC06]. It focusses on LEB and a parallel algorithm is introduced that produces an unstructured conforming mesh, which is not parallel in the domain decomposition sense, but instead the whole grid is known on each processor and the algorithm itself is executed in parallel. Jones and Plassmann [JP97] have implemented and analysed a parallel version of LEB using a colouring approach. In [JHJ12] a fully distributed parallel LEB is implemented based on the package DOLFIN [LWH12] and reasonable scaling results up to 1024 cores are presented, however without theoretical backing of the adaptation algorithm.

Further examples of parallel, adaptive simplex grid implementations are the software package FREEFEM++ [Hec12], which uses node-movement and remeshing, and the DUNE module DUNE-UG [BBJ<sup>+</sup>97], which implements

a parallel, adaptive simplex grid using red refinement with or without green closure. This list of parallel, adaptive simplex grids is not exhaustive.

**Dune** Most implementation in this work is done within the software framework DUNE [BBD<sup>+</sup>06, BBD<sup>+</sup>08], in particular DUNE-ALUGRID [ADKN16], DUNE-FEM [DKNO10] and DUNE-ACFEM [Hei14]. DUNE-ALUGRID provides an unstructured, adaptive grid manager that can handle simplicial and cubical elements in 2 and 3 dimensions and also conforming and non-conforming refinement. DUNE-FEM provides discrete functions, linear solvers and structures to deal with parallel FEM computations. DUNE-ACFEM is a convenience module on top of DUNE-FEM that allows to simply write mathematical expressions as code, which invokes the corresponding DUNE-FEM based implementation.

### 1.3 Thesis Outline and Contribution

In Chapter 2 we formally introduce the concept of hierarchically adaptive meshes and their properties for both simplicial and cubic grids. Different refinement strategies are related to each other. Section 2.1 introduces hierarchical adaptive refinement and some necessary concepts. Sections 2.2 and 2.3 introduce different refinement strategies and in particular NVB. In Section 2.4 the concepts of mesh-balance and conformity are introduced and we show that these concepts lead to refinement propagation. In summary, this chapter sets the stage for the theoretical improvements and the unified framework developed in Chapter 3 and Chapter 4.

Chapter 3 contains one of the major original contents of this work. We introduce a new weaker compatibility condition [AGK18] which allows NVB to be applied to any simplicial mesh. Section 3.1 contains properties of Newest Vertex Bisection and a unified theory for a novel refinement tree concept. This concept allows us to formulate new compatibility conditions in Section 3.2 and compare them to the standard strong compatibility condition [Ste08, Mau95, Tra97]. We design an algorithm to relabel any arbitrary

dimensional simplicial mesh to fulfil said condition in Section 3.3, which is one of the main benefits of the new compatibility condition: In particular, previous compatibility concepts do not allow to achieve compatibility in such a straightforward and elegant way. Last we perform some numerical experiments to investigate the quality of the produced meshes in Section 3.4. This chapter presents and partially extends results from the article [AGK18] by the author, Robert Klöforn and Fernando Gaspoz published in *SIAM Journal of Scientific Computing* end of 2018.

In more detail, the applicability to generic meshes allows the combination of NVB and mesh generators for complex geometries such as TETGEN [Si15] or GMSH [GR09]. We gain this applicability at the cost of losing some convenient properties obtained by the stronger compatibility condition. The weak compatibility condition generalizes the concept of a mesh being conformingly marked as introduced by Arnold et al. in [AMP00] and directly relates to what Stevenson calls compatibly divisible [Ste08]. We will show that NVB terminates successfully using this condition.

Furthermore, we present an algorithm that is capable of relabeling any grid to be weakly compatible and has in principle an effort of  $O(n)$ , where  $n$  is the number of elements in the grid. This alleviates the loss of the aforementioned convenient properties. In our implementation, however, the algorithm shows a complexity of  $O(n \log n)$  which is related to a neighbour search that is carried out prior to the relabeling algorithm. In addition, the algorithm is capable to recover a strongly compatible situation for certain classes of meshes.

To estimate the effort of the conforming closure, we will investigate novel metrics that relate to distances to a strongly compatible situation, as a strongly compatible situation minimizes the effort by design.

Chapter 4 contains 4 sections. Section 4.1 briefly introduces the parallel version of NVB refinement, which consists of an outer loop, that requires communication and an inner loop, that executes NVB locally on each partition. Section 4.2 estimates the number of outer iterations needed and relates



it to the amount of global communication needed. The results within this section depend on the strong compatibility condition.

While the first result (Theorem 4.5) holds without additional assumptions and for any dimension, the stronger results (2d: Theorem 4.9, 3d: Theorem 4.14) additionally assume that the grid is distributed on a certain element level. With this assumption we can show that the parallel overhead of reaching a conforming state with NVB is bounded, in particular by a constant independent of the number of processors. Section 4.3 extends part of the results to weakly compatible grids. Theorem 4.18 shows that refinement cannot propagate more than a layer of macro elements around the macro element that the refinement originates from, once the mesh has been refined to a certain level. Finally, in Section 4.4 we execute experiments to verify the theoretical results and show proper scaling of the algorithm. This chapter constitutes a substantial extension of the previously published work in the *Journal of Parallel and Distributed Computing* in 2017 [AK17].

We make the common assumption that load balancing is done after the refinement algorithm is finished. Hence we do not consider its effect on the computational cost of the refinement algorithm. However, we will show that it is possible to achieve decent strong scaling on a supercomputer using our distributed NVB implementation.

Unless stated otherwise, throughout this chapter we assume strong compatibility (Condition 3.22). This work was done before the introduction of the weak compatibility condition in our work [AGK18] and its extension is non-trivial. Theorem 4.18 and its Corollary 4.20 are a first extension to a weakly compatible situation.

In Chapter 5 we cover aspects of the software that we developed and extended over the course of this thesis. While this chapter contains only a small scientific contribution, the implementation within represents a major part of the overall work. A considerable amount of time and effort has been spent to implement the presented features.

We extended the grid manager DUNE-ALUGRID by unifying the 2d and 3d mesh implementations. Among other new features of DUNE-ALUGRID this has been published in *Archive of Numerical Software* by the author, Andreas Dedner, Robert Klöfkorn and Martin Nolte [ADKN16]. The idea behind the implementation is introduced in Section 5.1. The chosen approach aims at reducing the amount of code and hence increasing its maintainability by reusing part of the 3d grid implementation. The given 2d grid results in a 3d grid construction internally and part of the 3d surface is exported as the managed 2d grid to the user. This is possible as DUNE-ALUGRID consists of two layers: the internal layer has been originally implemented by Bernhard Schupp [Sch99] as part of a PhD thesis and is the actual grid implementation, and a second layer implements the DUNE-Grid interface.

Section 5.2 briefly introduces the DUNE module DUNE-ACFEM which has been developed by Claus-Justus Heine, Birane Kane, Stephan Hilb and the author. DUNE-ACFEM aims to simplify the implementation of Finite-Elements on adaptive, parallel grids using the DUNE infrastructure. DUNE-ACFEM uses expression templates for functions and equation terms, which can be added and multiplied. This greatly simplifies the implementation of complex equations as each part of the equation may be implemented separately and the whole equation is afterwards assembled by the compiler. This has been optimized such that known zero-expressions are discarded during compile-time. A drawback of this approach are long compilation times.

In Section 5.3 we describe the proof-of-concept implementation of an arbitrary-dimensional simplex grid with NVB refinement fulfilling the weak compatibility condition, which is used for the experiments in Section 3.4.6. We choose to store the connectivity information on the edges of the grid, which allows for a convenient implementation of the refinement algorithm. This algorithm appears to scale linearly with the number of elements refined in arbitrary dimension.

In Chapter 6 we demonstrate the ease of implementation with DUNE-ACFEM and the usability of the new 2d grid implementation within DUNE-

ALUGRID with a complex image processing application. This is done by minimizing a non-smooth functional consisting of a combined  $L^1/L^2$ -data fidelity term and a total variation term for regularization. Such an optimization problem has been shown to effectively remove Gaussian and salt-and-pepper noise simultaneously, see [HL13, Lan17a]. In order to compute an approximate solution we use the primal-dual algorithm proposed in [CP11], which requires a saddle point formulation of the problem. For the numerical implementation we discretize using various finite-element spaces defined over locally refined conforming grids. Similar to the works [Bar12, Bar15b, Bar15a, HRC14], where the considered functional is composed solely of an  $L^2$ -data term and a total variation term, we refine the grid adaptively using an a priori criterion.

We use the new 2d grid implementation of DUNE-ALUGRID (cf. Section 5.1) which provides a distributed grid with local refinement capabilities. In combination with DUNE-ACFEM (cf. Section 5.2) the resulting algorithm is automatically parallelized. The advantage of using DUNE-ACFEM is that it allows us to conveniently express the given algorithm in code and use all of the underlying parallel infrastructure of DUNE effortlessly from the application perspective. This includes (non)-linear solvers, discrete functions (vectors), (linear) operators (system matrices) and interpolation operators.

In Section 6.1 we formulate the continuous and the discrete problem with the respective discrete spaces. In particular for a certain discretization we state that the discrete problem converges to the continuous one as the mesh-size goes to zero. In Section 6.2 we discuss how DUNE-ACFEM is used to implement the primal-dual algorithm. Numerical examples showing the applicability of our proposed implementation and experiments testing different discretizations are presented in Section 6.3. This chapter follows closely the article [AL17] published in *Archive of Numerical Software* by the author and Andreas Langer.

# Chapter 2

## Adaptive Meshes

We introduce adaptive grids with focus on form-stable, hierarchical methods for simplices and cubes. Particular interest will be given on conforming refinement of simplicial grids using Newest Vertex Bisection (NVB).

Other forms of adaptivity that we do not consider include remeshing with point insertion (as done e.g. in Freefem or CGAL [Hec12, The18]), anisotropic refinement ([Ape99, Hua06, Sch13]) or arbitrary polygonal meshes, as used for Virtual Element or Hybrid Higher Order methods (implemented e.g. in Disk++ [CPE18]).

We consider a bounded domain  $\Omega \in \mathbb{R}^d$  (usually  $d = 2, 3$ ) with a polyhedral boundary and a grid  $\mathcal{T}$  of the domain as a set of  $d$ -dimensional simplices or (hyper-)cubes that cover the domain. The intersection of two elements is assumed to have zero  $d$ -dimensional measure, which implies that the intersection is always of a lower dimension. All elements  $T \in \mathcal{T}$  are chosen to be of the same type (i.e. cube or simplex) to simplify the analysis and implementation of the grids.

In Section 2.1 we introduce the concepts of grids, elements and the standard idea of hierarchical grid refinement. This is specialized in Section 2.2 for isometric refinement of cubes and simplices and in Section 2.3 for bisection-based methods. In particular in Section 2.3.2 introduces Newest Vertex Bisection (NVB) and some of its properties. NVB is analysed thoroughly in

the later chapters. The last Section 2.4 introduces the concept of refinement propagation, which depends on a mesh-balance or a conformity condition.

Some results of this chapter have been published in [AGK18].

## 2.1 Some Notations and Concepts

A *simplex* (also *d-simplex* for dimension  $d$ ) is the natural  $d$ -dimensional extension of the triangle.

**Definition 2.1 (Simplex).** *A simplex  $T \subset \mathbb{R}^d$  is the convex hull of  $d + 1$  vertices  $z_i \in \mathbb{R}^d, i \in \{0, \dots, d\}$ , where the set of edges  $z_i - z_0, i \in \{1, \dots, d\}$  is linearly independent.*

*We will often use the identification of a simplex  $T$  with the set of its vertices  $\{z_0, \dots, z_d\}$  and write  $T = \{z_0, \dots, z_d\}$ .*

The natural  $d$ -dimensional extension of the quadrilateral is the *cube* (or *hypercube*). A definition is more complex, as lower-dimensional parts of the cube may be curved, unless all vertices of a lower-dimensional object are restricted to a common plane. If we allow curved lower-dimensional parts, this imposes a loss of convexity, but allows for more general element shapes. So we define the reference cube and call any element hypercube, if it can be created by an invertible (usually multilinear) transformation from the reference cube.

**Definition 2.2 (Reference Cube and (Hyper-)Cube).** *The reference cube (also unit cube)  $\hat{T}$  is the convex hull of the  $2^d$  vertices  $z = (\delta_1, \dots, \delta_d) \in \mathbb{R}^d$  with  $\delta_i \in \{0, 1\} \forall i \in \{1, \dots, d\}$ .*

*Any element  $T$  is called a (hyper-)cube, if and only if there is an invertible transformation  $\Gamma : \mathbb{R}^d \rightarrow \mathbb{R}^d$  with  $\Gamma(\hat{T}) = T$ .*

Similarly we can define the reference simplex. As we only allow grids of one element type, we also call it  $\hat{T}$ .

**Definition 2.3 (Reference Simplex).** *The reference simplex  $\hat{T}$  is the convex hull of the  $d + 1$  vertices  $z_0 = (0, \dots, 0)$  and  $z_i = (0, \dots, 0, 1, 0, \dots, 0)$  with 1 at position  $i$  for  $i \in \{1, \dots, d\}$ .*

As lower-dimensional parts of a simplex always have precisely the number of vertices needed to define a plane, any simplex can be created by an affine transformation from the reference simplex. Note that lower-dimensional subentities of hypercubes are hypercubes and lower-dimensional subentities of simplices are simplices.

To create a simplex mesh from a cube grid in arbitrary dimension, one can use the so-called Kuhn-cube, that fills a cube with a set of  $d!$  similar simplices [Kuh60]. It chooses two vertices  $v_0, v_1$  connected by a diagonal and creates a simplex for each possible path of length  $d$  from vertex  $v_0$  to  $v_1$  that uses edges of the cube. In case of the reference cube  $\hat{T}$  the standard layout uses  $v_0 = (0, \dots, 0), v_1 = (1, \dots, 1)$ . Each edge of the cube corresponds to a unitvector of  $\mathbb{R}^d$ . As we want the path from  $v_0$  to only use edges and be of length  $d$ , we have to choose every unit vector once in positive direction. Then we have  $d$  options to choose the initial direction,  $d - 1$  to choose the second direction,  $d - 2$  to choose the third and so on. This leads to  $d!$  different simplices. See Figure 2.1 for an illustration.



Figure 2.1: Kuhn-cube in 2 and 3 dimensions. In 3 dimensions a simplex is formed by the red diagonal and each green line.

It is also possible to create a (possibly ill-shaped) mesh of cubes from any simplex mesh. A simplex is split into a set of  $d + 1$  (number of vertices) cubes. For each vertex a cube is created at that vertex by using all centres of adjacent lower-dimensional simplices and the barycenter of the simplex itself as corners. See Figure 2.2 for an illustration.

We now define the concept of a grid as a set of simplices or cubes. We use a custom definition that does not allow a grid to contain both cubes and simplices and also forbids other elements (e.g., pyramids, prisms).



Figure 2.2: How to construct a set of cubes from a simplex in 2 and 3 dimensions.

**Definition 2.4 (Triangulation/Grid).** *Given a polyhedral bounded domain  $\Omega \in \mathbb{R}^d$ , a triangulation  $\mathcal{T}$  (or grid/mesh) of this domain of size  $N$  is a set of elements  $T_i, i \in \{1, \dots, N\}$  with the following properties:*

- *For each  $x \in \Omega$ , there exists a  $T \in \mathcal{T}$ , such that  $x \in T$ .*
- *All elements  $T \in \mathcal{T}$  are either cubes or simplices.*
- *The intersection of any two elements  $T_i, T_j$  with  $i \neq j$  is a lower-dimensional cube or simplex or the empty set.*

To introduce refinement on the elements, it suffices to introduce the refinement on the reference elements and use the mapping of the parent element to map the children of the reference element on the children of the actual parent element.

**Definition 2.5 (Hierarchical Refinement).** *(Hierarchical) refinement divides a single parent (parent element, sometimes father) into children (children elements) such that the set of all children covers the volume of the parent and the children only intersect on lower-dimensional parts.*

This definition of hierarchical refinement results in an infinite binary tree, if an element has two children, and an infinite quaternary tree, if each element has four children, and so on. Multiple trees are called forest.

**Definition 2.6 (Tree, Forest, Macro Triangulation).** *The tree of an element  $T$  is called  $\mathfrak{F}(T)$  and the set of all trees of the elements in a given triangulation forest  $\mathfrak{F}(\mathcal{T})$ .*

*The roots of the forest  $\mathfrak{F}(\mathcal{T})$  are called macro triangulation or initial triangulation  $\mathcal{T}_0$ .*

Important in the course of this work is the *generation* of an element, which describes distance to the macro element that it derives from within the tree of the macro element.

**Definition 2.7 (Generation).** *For each  $T \in \mathcal{T}$  there is a  $T_0 \in \mathcal{T}_0$  such that  $T \in \mathfrak{F}(T_0)$ . The generation  $\text{gen}(T)$  is the number of its ancestors in the tree  $\mathfrak{F}(T_0)$ , or, equivalently, the number of refinements needed to create  $T$  from  $T_0$ .*

In particular an element  $T \in \mathcal{T}_0$  has generation  $\text{gen}(T) = 0$ .

The following concept of *uniform refinements* describes splitting an element into descendants of a certain generation. This is used often for bisection-based methods as one refinement only splits a single edge, and we need to know, when all edges of an element have been split.

**Definition 2.8 (Uniform Refinements).** *Let  $n \in \mathbb{N}_0$ . Then  $n$  uniform refinements of an element  $T$  of generation  $\text{gen}(T) = g$  are defined by refining the element and all descendants until generation  $g + n$ .*

Uniform refinements of a whole grid are defined by refining each element within the grid uniformly. We use  $\mathcal{T}_j$  with a subindex  $j$  for  $j$  uniform refinements of the initial  $\mathcal{T}_0$ .

Since we also consider lower-dimensional parts of the grid, we need to define the skeletons of these parts. We will use  $\mathcal{T}^i$  with a superindex  $i$  for lower-dimensional subentities as follows.

**Definition 2.9 (Skeleton of a Triangulation).** *Let  $\mathcal{T}$  be a triangulation of a domain  $\Omega \subset \mathbb{R}^d$ . For  $0 \leq i \leq d$  we denote by  $\mathcal{T}^i$  the set of all elements of dimension  $i$  that are contained in  $\mathcal{T} = \mathcal{T}^d$  and call  $\mathcal{T}^i$  the  $i$ -skeleton of  $\mathcal{T}$ . We also define  $\mathcal{T}^i(T)$  to be the  $i$ -skeleton of the triangulation that consists of the single element  $T$ .*

*In particular  $\mathcal{T}^0 = \mathcal{V}$  is the set of all vertices,  $\mathcal{T}^1 = \mathcal{E}$  the set of all edges,  $\mathcal{T}^{d-1} = \mathcal{F}$  the set of all faces and  $\mathcal{T}^d = \mathcal{T}$  the whole grid.*



Note that  $\mathcal{T}_j^i$  is the  $i$ -skeleton of the  $j$ -times refined grid and not the  $j$ -times refined  $i$ -skeleton. This is important, as during refinement lower-dimensional entities (e.g. edges) are created, that cannot be constructed by refining already existing lower-dimensional entities.

These introduced notions may be combined. As an example  $\mathcal{F}_i(T)$  is the set of faces of the  $i$  times uniformly refined element  $T$ . Also the grid  $\mathcal{T}^d(T) = \mathcal{T}(T)$  is the grid consisting of the single element  $T$ .

Because  $\mathcal{V}_i$  is never needed in the sense of generations,  $\mathcal{V}_0$  and  $\mathcal{V}_1$  will have a different meaning over the course of the later chapters. We will also see, that although they are not the vertices of a certain generation, the concept used in these chapters is very close.

To investigate the relationship between neighbouring elements we now define several neighbouring concepts. First we define the *environment* of a vertex.

**Definition 2.10 (Environment).** *For each vertex  $z \in \mathcal{V}$  we define the environment of  $z$  as*

$$\mathcal{T}_z := \{T \in \mathcal{T} \mid z \in T\}.$$

A bit stronger than the concept of environment is the concept of a *neighbour*, which is sharing a non-zero dimensional subentity (i.e., at least an edge). The strongest neighbouring concept is the *direct neighbour*, that shares a full  $d - 1$  dimensional subentity (i.e., a face).

**Definition 2.11 ((Direct) Neighbours).** *We say  $T, T' \in \mathcal{T}$  are direct neighbours if and only if there is a face  $F \in \mathcal{F}$  with  $F \subset T \cap T'$ .  $T, T' \in \mathcal{T}$  are neighbours if and only if there is an edge  $E \in \mathcal{E}$  with  $E \subset T \cap T'$ .*

Most above definitions are topological. To study the quality of a mesh, we also need geometric properties. We start with the *mesh-size*, which is often used for convergence analysis of algorithms.

**Definition 2.12 (Mesh-Size).** *We let  $h_{\mathcal{T}} \in L_{\infty}(\Omega)$  be the piecewise constant mesh-size function with  $h_{\mathcal{T}|_T} = h_T := |T|^{1/d} \approx \text{diam}(T)$  for  $T \in \mathcal{T}$ . We use  $h_{\min}(\mathcal{T})$  and  $h_{\max}(\mathcal{T})$  for the smallest and largest element size of  $\mathcal{T}$ .*

Also the concept of *Shape regularity* is needed for convergence of finite-elements on our meshes. It can be defined in various forms, see [BKK09] and it guarantees that there are no degenerate elements within  $\mathfrak{F}(\mathcal{T}_0)$  and that the minimum angle  $\theta_{min} \not\rightarrow 0$  and the maximum angle  $\theta_{max} \not\rightarrow \pi$  do not converge into the limit.

**Definition 2.13 (Shape Regularity/ Form Stability).** *A sequence of meshes  $\{\mathcal{T}_k\}_{k \in \mathbb{N}_0}$  generated by refinement is called shape regular or form stable, if there is a constant  $c_{SR} > 0$  for all  $T \in \mathfrak{F}(\mathcal{T}_0)$ , such that*

$$|T|^{1/d} \geq c_{SR} \text{diam}(T).$$

*The bound itself is called form stability constant.*

The constant  $c_{SR}$  depends on the element shapes in the initial triangulation  $\mathcal{T}_0$  and the chosen refinement strategy.

As all elements are refined by the same refinement strategy in this work, it is sufficient to prove that the refinement strategy produces a shape regular sequence of meshes for the initial grid consisting of the reference element,  $\mathcal{T}_0 = \mathcal{T}(\hat{T})$ . There are only finitely many elements in the initial grid  $\mathcal{T}_0$  and hence finitely many transformations from the reference element  $\hat{T}$  to the elements in  $\mathcal{T}_0$ . This leads to the full shape regularity constant  $c_{SR}(\mathcal{T}) = c_{SR}(\hat{T}) \cdot c_m$ , where  $c_m$  is a constant depending on the worst mapping from the reference element  $\hat{T}$  to an element  $T \in \mathcal{T}$ .

A stronger concept than shape regularity is the *strong regularity*, for which we need to define a *similarity class*.

**Definition 2.14 (Similarity Class).** *Two elements  $T_1, T_2$  are similar to each other, if and only if there is an affine transformation  $S : \mathbb{R}^d \rightarrow \mathbb{R}^d$ , with  $S(T_1) = T_2$  and  $S$  only contains a rotation, translation, scaling or flip. The set of all elements that are similar to each other is called similarity class.*

Since similarity is obviously an equivalence relation the similarity class is well-defined.

The following definition of regularity is called strong, because it not only ensures that the geometric properties of all children do not diverge, but also that after a certain number of refinements all newly created elements are similar to previous elements.

**Definition 2.15 (Strong Regularity).** *If a refinement strategy produces finitely many similarity classes in  $\mathfrak{F}(\mathcal{T}(\hat{T}))$ , it is called strongly regular.*

Shape regularity of the sequence of triangulations follows directly from strong regularity of the refinement strategy. A triangulation contains finitely many elements and the refinement strategy produces finitely many similarity classes for each element, so the number of similarity classes is finite.

**Example 2.16.** *The refinement strategy Newest Vertex Bisection (introduced in Section 2.3.2) creates 4 different similarity classes in 2d (compare Figure 2.11). If we have an initial triangulation of size  $N$ , then we have in total  $4N$  different similarity classes in the complete forest  $\mathfrak{F}(\mathcal{T}_0)$ . This guarantees a shape regularity constant  $c_{SR} > 0$ .*

## 2.2 Isometric Refinement

After the general considerations in the last section, we now introduce actual refinement strategies in both this and the next section. *Isometric refinement* splits a reference element (cube or simplex)  $\hat{T}$  into  $2^d$  subelements  $\mathcal{T}_1(\hat{T})$ . It aims at every child  $\hat{T}' \in \mathcal{T}_1(\hat{T})$  being similar to the parent  $T$ . For cubes this is the case and it is defined as follows.

**Definition 2.17 (Isometric Cube Refinement).** *For each vertex  $z \in \mathcal{V}(\hat{T})$  of the reference cube we define the child  $\hat{T}_z$  as the cube created by the affine linear transformation  $A_z : \hat{T} \rightarrow \hat{T}_z$  by  $A_z v = \frac{v+z}{2}$ .*

The transformation  $A_z$  keeps  $z$  in place and forms the new cube from the centres of all at  $z$  adjacent (sub-)entities.

Obviously isometric cube refinement is strongly regular as all resulting children of the unit cube are similar to the unit cube, see Figure 2.3.

**Lemma 2.18** (Shape Regularity of Isometric Cube Refinement). *Isometric cube refinement is strongly regular in any dimension.*

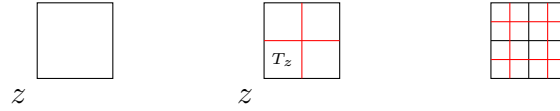


Figure 2.3: Isometric cube refinement in 2 dimensions

Isometric simplex refinement is difficult to define precisely in arbitrary dimensions without introducing complex topological structures. Also it is not exactly isometric in more than 2 dimensions since not every child is similar to the parent. It usually is referred to as red refinement [Fre42, Gra19, Bey95, Ban83] and it produces a limited number of similarity classes [Bey00].

In 2 dimensions we have the following definition.

**Definition 2.19 (Red Refinement in 2 dimensions).** *Let  $\hat{T} \subset \mathbb{R}^2$  with vertices  $z_0, z_1, z_2$  and let  $z_{ij} = \frac{z_i + z_j}{2}$ . Then red refinement splits  $\hat{T}$  into the four simplices*

$$T_0 = \{z_0, z_{01}, z_{02}\}, \quad T_1 = \{z_1, z_{01}, z_{12}\}$$

$$T_2 = \{z_2, z_{12}, z_{02}\}, \quad T_3 = \{z_{02}, z_{01}, z_{12}\}.$$

See Figure 2.4 for an illustration. The first  $d + 1$  children at the vertices can also be formed similarly in higher dimension, but the central triangle in 2d is not a simplex in higher dimensions and has to be subrefined.

**Lemma 2.20 (Shape Regularity of Red Refinement).** *Red refinement is strongly regular in any dimension.*

*Proof.* [Bey00] □

Both isometric cube and simplex refinement lead to a forest  $\mathfrak{F}(\mathcal{T}_0)$ , where each level has  $2^d$  times as many elements as the previous level.

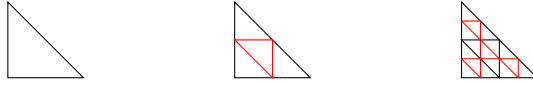


Figure 2.4: Isometric simplex refinement in 2 dimensions

## 2.3 Edge Bisection

We define *bisection* only on simplices. Cubes may also be bisected, but this does not as easily result in a conforming mesh. Bisection-based methods refine a simplex by bisecting an edge and splitting the simplex into two halves. The two children are created by substituting the center of the split edge with one of the vertices of the split edge.

**Definition 2.21 (Bisection).** Bisection of a simplex  $T = \{z_0, \dots, z_d\}$  with refinement edge  $E = \overline{z_i z_j}$  and its center  $z_{ij} = \frac{z_i + z_j}{2}$  leads to two children of the form  $T_0 = \{\mathcal{V}(T) \setminus \{z_i\}\} \cup \{z_{ij}\}$  and  $T_1 = \{\mathcal{V}(T) \setminus \{z_j\}\} \cup \{z_{ij}\}$ .

Bisection creates in any dimension a single new face inside the parent element and the two resulting children are direct neighbours sharing this face. It also creates new faces and lower-dimensional elements at the boundary of the parent element.

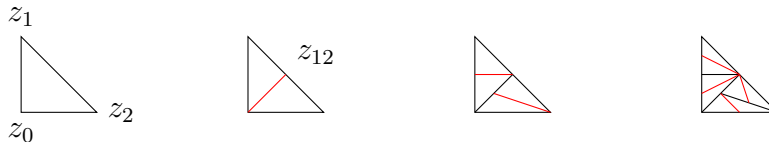


Figure 2.5: Arbitrary simplex bisection in 2 dimensions.

Figure 2.5 illustrates an arbitrary choice of refinement edges for bisection. Note that in general we cannot expect bisection to be shape regular. We want the bisection rule to choose the refinement edge such that the generated sequence of meshes is shape regular.

In addition to shape regularity, we also need a second property called *compatibility*. It ensures that for all pairs of elements  $T, T' \in \mathfrak{F}(\mathcal{T}_0)$  the refine-

ment edges of shared faces  $F = T \cap T'$  (with  $F$  being a full  $d - 1$ -dimensional subsimplex for both  $T$  and  $T'$ ) coincide from both adjacent elements.

**Definition 2.22 (Refinement Tree).** *Let  $T$  be a simplex. We execute uniform bisections until for each edge  $E \in \mathcal{E}(T)$  every element  $T' \in \mathfrak{F}(\mathcal{T}_0(T))$  with  $E \subset T'$  has been bisected. We define the refinement tree  $\mathcal{RT}(T)$  as follows. The nodes of  $\mathcal{RT}(T)$  are pairs  $(T', E')$  of an element  $T' \in \mathfrak{F}(\mathcal{T}_0(T))$  with their corresponding refinement edge  $E'$ . The edges of  $\mathcal{RT}(T)$  connect elements with their children.*

In other words  $\mathcal{RT}(T)$  is the forest  $\mathfrak{F}(T)$  enriched with the refinement edges of the elements, where only elements are included that are created until every edge of  $T$  has been refined from each adjacent element. For  $d > 2$  edges are shared by multiple children. These children have descendants that refine the shared edge, but the generation of these does not necessarily coincide. Hence we refine until the last descendant has refined the edge. The use of uniform refinements guarantees, that all leaves of the refinement tree are of the same generation.

As an example we consider the refinement tree of the Element  $T = \{z_0, z_1, z_2\}$  refined as in Figure 2.5.

**Example 2.23 (Refinement Tree of Figure 2.5).** *For simplicity we use the  $\Delta$  notation, e.g.  $T = \Delta z_0 z_1 z_2$ . This element has the refinement edge  $E = \overline{z_1 z_2}$ , so the root of the refinement tree is  $(\Delta z_0 z_1 z_2, \overline{z_1 z_2})$ . It gets bisected using the center of the refinement edge  $z_{12} = \frac{z_1 + z_2}{2}$  and produces the elements  $T_1 = \Delta z_0 z_{12} z_1$  with refinement edge  $E_1 = \overline{z_0 z_1}$  and  $T_2 = \Delta z_0 z_{12} z_2$  with refinement edge  $\overline{z_0 z_{12}}$ . After one further uniform refinement, every edge appears in the tree and we stop the construction. The resulting tree is depicted in Figure 2.6.*

We now consider the influence of the refinement structure of an element on its faces. To this aim we define the *induced refinement tree*. It describes the order in which edges contained in the face will be refined from within the element.

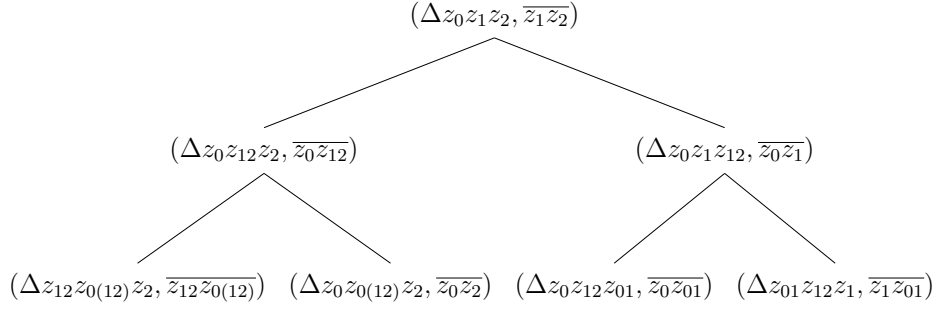


Figure 2.6: Refinement tree of Figure 2.5

**Definition 2.24 (Induced Refinement Tree (IRT)).** A sub-simplex  $F \in \mathcal{T}^{d-1}(T)$  of the simplex  $T$  has the induced refinement tree  $\mathcal{RT}_T(F)$  by only including nodes  $(T', E') \in \mathcal{RT}(T)$  where  $E' \subset F$ . The resulting node within  $\mathcal{RT}_T(F)$  is  $(T' \cap F, E')$ . Subtrees of nodes not containing  $E$  are neglected and edges leading to nodes, that are not contained are concatenated to a single edge.

**Example 2.25 (IRT of Face  $\overline{z_0z_1}$  in Figure 2.5).** The induced refinement tree of face  $\overline{z_0z_1}$  coincides with the right subtree in Figure 2.6 as all refinement in this triangle is directed at the face  $\overline{z_0z_1}$ .

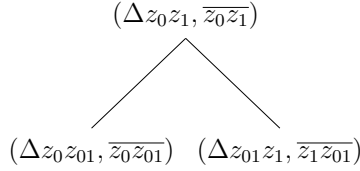


Figure 2.7: Induced refinement tree of face  $\overline{z_0z_1}$

Very different internal refinement trees may lead to the same induced refinement tree. Because of this, compatibility of two direct neighbours should only consider the induced refinement tree of the shared face, which is how we formulate the compatibility condition. It is called weakest compatibility, as we will later introduce stronger compatibility conditions and because it is necessary for conforming (Definition 2.36) refinement.

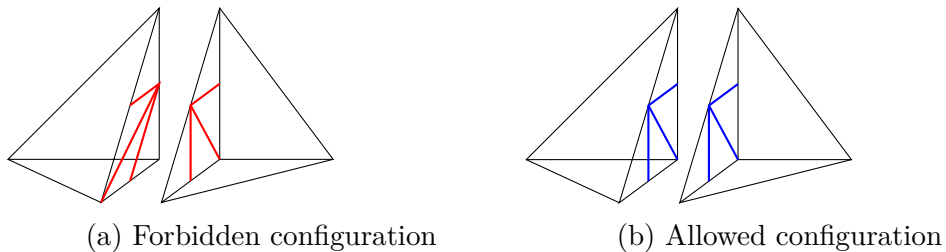


Figure 2.8: An illustration of Condition 2.26.

**Definition 2.26 (Weakest Compatibility Condition).** *A triangulation  $\mathcal{T}$  fulfils the weakest compatibility condition, if and only if for any pair of direct neighbours  $T_i, T_j \in \mathfrak{F}(\mathcal{T})$  and their shared face  $F = T_i \cap T_j$  it holds  $\mathcal{RT}_{T_i}(F) = \mathcal{RT}_{T_j}(F)$ .*

This condition is illustrated in Figure 2.8. The condition  $\mathcal{RT}_{T_i}(F) = \mathcal{RT}_{T_j}(F)$  enforces that the order in which refinement edges are chosen on the face coincides, so that the mismatch in Figure 2.8a cannot happen.

The condition has to hold for each pair of direct neighbours within the whole forest  $\mathfrak{F}(\mathcal{T})$  which ensures that refinement always is compatible from both adjacent elements of any face independent of their position in the forest.

An equivalent condition is that the initial refinement edge of a face shared by two elements (i.e. the root of the induced refinement tree) matches from both sides. Although this equivalent formulation may sound easier using the refinement tree simplifies some proofs in Chapter 3.

### 2.3.1 Longest Edge Bisection

The basic idea of *Longest Edge Bisection* (LEB) [Riv97, HKK10, ACH<sup>+</sup>15, KKK08, Hor97] is to always refine the longest edge of a simplex. If two edges are (up to computer accuracy) of the same size it needs a well-defined decision rule. This decision rule can for example be based on edge-indices within the implementation.

In 2 dimensions it is known, that if the grid contains elements that have small angles, LEB improves the element shape, which means, that starting



with  $\mathcal{T}_i, i > 0$  as initial triangulation instead of  $\mathcal{T}_0$ , the form-stability constant of the generated sequence of meshes is better [KPS16, Remark 9].

**Definition 2.27 (Longest Edge Bisection).** *Let  $T \in \mathcal{T}$  be a simplex and  $E = \overline{z_i z_j}$  be the longest edge. Then Longest Edge Bisection (LEB) is defined by the two children*

$$T_0 = \text{conv}(\mathcal{V}(T) \setminus \{z_i\} \cup \{z_{ij}\})$$

and

$$T_1 = \text{conv}(\mathcal{V}(T) \setminus \{z_j\} \cup \{z_{ij}\})$$

In principle we expect LEB to be shape regular in any dimension. To prove this is difficult, as there are many cases on how the edges can be ordered and there also may be edges of the same length. However, there is the following result.

**Lemma 2.28 (Shape Regularity).** *For  $d = 2$  LEB is strongly regular. For  $d > 2$  we have, LEB is shape regular, if and only if it is strongly regular.*

*Proof.* [KPS16, Theorem 8 and 10] □

A more general version of LEB can be defined by introducing an ordering on all edges in  $\{\mathcal{E}_k\}_{k \in \mathbb{N}_0}$ . The refinement edge is then chosen according to this ordering. A requirement on the ordering is that for any fixed  $i \in \mathbb{N}_0$  the position of all edges in  $\mathcal{E}_i$  is finite inside this ordering. As an example this is fulfilled if the length of the edge is chosen as the ordering criterion, as every bisection halves it. This general version of LEB may not be shape regular, but is compatible by design. This is due to the fact, that the refinement edge is always chosen according to a global decision rule.

**Lemma 2.29 (Compatibility).** *Grids generated by LEB fulfil condition 2.26.*

*Proof.* Let  $T$  be an element with a face  $F$ . We have a global ordering of all edges  $\mathcal{E}_k, k \in \mathbb{N}_0$ . This implies that, if we consider  $F$  as a  $d - 1$  dimensional mesh element with LEB, we get a unique refinement tree  $\mathcal{RT}(F)$ , based on the edge ordering of  $\mathcal{E}_k(F)$ . The ordering of  $\mathcal{E}_k(F)$  is embedded in the ordering of  $\mathcal{E}_k(T)$ . So it holds that  $\mathcal{RT}_T(F) = \mathcal{RT}(F)$  for all  $T \in \mathfrak{F}(\mathcal{T})$  with  $F \subset T$ . This directly yields Condition 2.26.  $\square$

### 2.3.2 Newest Vertex Bisection

We investigate *Newest Vertex Bisection* (NVB) more in detail. NVB was introduced by Sewell [Sew72] and enhanced by Mitchell with a recursive refinement algorithm [Mit88, Mit89]; compare also with [Bän91, Mau95, Ste08, Tra97]. We follow the notation of [GHS15] and [AGK18].

The basic idea of NVB is to choose the refinement edge always opposite of the most recently introduced vertex, which means that this vertex is never contained in the refinement edge. In 2 dimensions this leads to a single possible refinement edge, but in higher dimensions there are more options. These options have to be chosen such that the refinement strategy stays form stable. To this aim each element gets a *type* to guarantee strong regularity.

**Definition 2.30 (Newest Vertex Bisection).** *We identify a  $d$ -dimensional simplex  $T$  with the pair of an ordering of its vertices  $z_i$  and a type  $t_T \in \{0 \dots, d - 1\}$ , i.e.,*

$$T = ([z_0, z_1, \dots, z_d], t_T),$$

or in short

$$T = [z_0, z_1, \dots, z_d]_{t_T}.$$

Then the Newest Vertex Bisection (NVB) of the simplex  $T$  is defined by its two children

$$T_0 = \left[ z_0, \frac{z_0 + z_d}{2}, z_1, \dots, z_{d-1} \right]_{t_T+1 \bmod d}$$

and

$$T_1 = \left[ z_d, \frac{z_0 + z_d}{2}, \underbrace{z_1, \dots, z_{t_T}}_{\rightarrow}, \underbrace{z_{d-1}, \dots, z_{t_T+1}}_{\leftarrow} \right]_{t_T+1 \bmod d}.$$

We call the edge  $E_T = \overline{z_0 z_d}$  the refinement edge of the simplex  $T$ .

This procedure automatically presets the children's refinement edges by the local ordering of their vertices. So the combination of ordering and type of an element completely describes the full refinement of all possible descendants. NVB thereby determines the refinement edge of any descendant produced by recurrent bisection of a given initial element  $T_0$  from the vertex order and the type of  $T_0$ . Figures 2.9 and 2.10 give an impression on NVB in 2d and 3d. In all figures the current refinement is depicted in red and the previous refinement are the dashed lines.

Figure 2.9 depicts two uniform refinements of an element  $T = [z_0, z_1, z_2]_{t_T}$ . The behaviour of the two possible types  $t_T = 0, 1$  coincides, as we only have one option to choose the refinement edge opposite of the newest vertex.

Figures 2.10a and 2.10b show three uniform refinements of the simplices  $T = [z_0, z_1, z_2, z_3]_0$  and  $T = [z_0, z_1, z_2, z_3]_1$ . We see that in both cases three uniform refinements suffice to refine each edge of the initial simplex once, but also that the refinement behaviour of these two types differs.

Finally Figure 2.10c shows three uniform refinements of a simplex of type 2. We see that in this case these are not sufficient to refine each edge of the initial simplex since the edge  $E = \overline{z_1 z_2}$  is not refined. In all three 3d subfigures we observe that the initial refinement edge solely depends on the current ordering and not on the type. The type only affects later refinement edges, as we will see in the following remark. This remark is important over the course of this thesis and the identification therein will often be used.

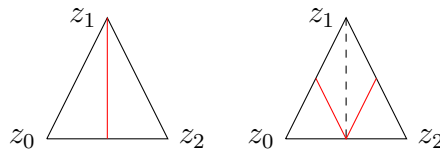


Figure 2.9: Newest Vertex Bisection in 2d

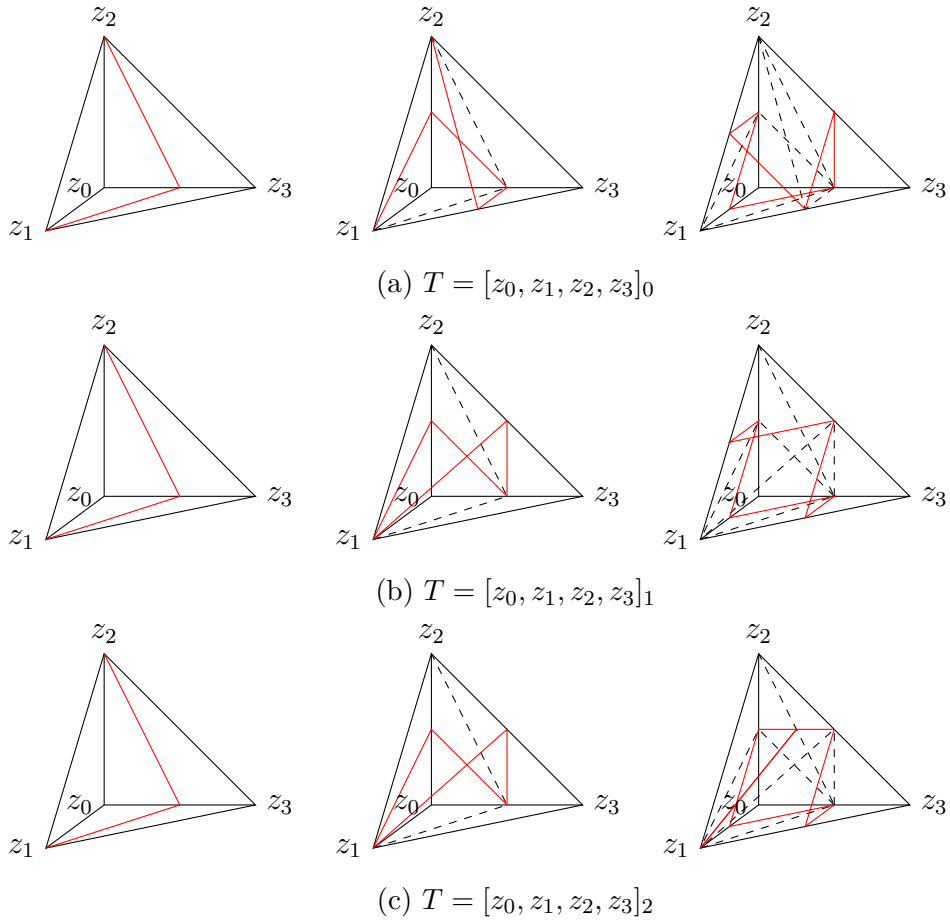


Figure 2.10: Newest Vertex Bisection in 3d.

**Remark 2.31 (Importance of Type).** *The type  $t_T$  of an element  $T$  indicates how many vertices within the element are labelled as “new” and guards them at positions  $1, \dots, t$ .*

*Definition 2.30 encodes the identification of the simplex  $T = [z_0, \dots, z_d]_t$  with two ordered sets*

$$\mathcal{V}_0 = [z_0, z_{t+1}, \dots, z_d], \mathcal{V}_1 = [z_1, \dots, z_t].$$

The children  $T_0, T_1$  of  $T$  from Definition 2.30 then correspond to

$$\begin{aligned} T_0 : \mathcal{V}_0 \setminus \{z_0\} &= [z_{t+1}, \dots, z_d], \mathcal{V}_1 = \left[\frac{z_0 + z_d}{2}, z_1, \dots, z_t\right] \\ T_1 : \mathcal{V}_0 \setminus \{z_d\} &= [z_0, z_{t+1}, \dots, z_{d-1}], \mathcal{V}_1 = \left[\frac{z_0 + z_d}{2}, z_1, \dots, z_t\right]. \end{aligned}$$

Note that  $\mathcal{V}_1$  coincides for both children. These are the vertices that are “guarded” from refinement. Once  $d$  vertices have been guarded, ( $\#\mathcal{V}_1 = d$ ,  $t_T = d$ ), the guarding set  $\mathcal{V}_1$  is cleared and the identification of the children looks as follows.

$$\begin{aligned} T_0 : \mathcal{V}_0 \setminus \{z_0\} &= [z_d, \frac{z_0 + z_d}{2}, z_1, \dots, z_t], \mathcal{V}_1 = \emptyset \\ T_1 : \mathcal{V}_0 \setminus \{z_d\} &= [z_0, \frac{z_0 + z_d}{2}, z_1, \dots, z_t], \mathcal{V}_1 = \emptyset. \end{aligned}$$

**Remark 2.32.** Further note that the ordering of  $\mathcal{V}_0$  may be reversed and the resulting children and all further refinement coincides, only  $T_0$  and  $T_1$  switch positions. This has also been observed in [Ste08, Section 2], where two simplices are identified if they have the same set of children.

**Example 2.33 (NVB in 3d).** Let  $d = 3$ . The simplex  $T = [z_0, z_1, z_2, z_3]_{t_T}$  is of type  $t_T \in \{0, 1, 2\}$ . Bisecting the refinement edge  $E_T = \overline{z_0 z_3}$  at its center  $z_{03} = \frac{z_0 + z_3}{2}$  leads to the two simplices

$$T_0 = [z_0, z_{03}, z_1, z_2]_{t_T + 1 \bmod 3} \text{ and } \begin{cases} T_1 = [z_3, z_{03}, z_2, z_1]_1 & \text{if } t_T = 0, \\ T_1 = [z_3, z_{03}, z_1, z_2]_2 & \text{if } t_T = 1, \\ T_1 = [z_3, z_{03}, z_1, z_2]_0 & \text{if } t_T = 2. \end{cases} \quad (2.1)$$

The refinement edges of the children are  $E_{T_0} = \overline{z_0 z_2}$  and  $E_{T_1} = \overline{z_3 z_1}$ , if  $t_T = 0$  and  $E_{T_1} = \overline{z_3 z_2}$ , if  $t_T \in \{1, 2\}$ .

From this we can directly deduce the initial refinement edge of all four

faces of the simplex, namely

$$\begin{aligned}
F_0 = \{z_0, z_1, z_2\} : & \quad E_{F_0} = \overline{z_0 z_2}, \\
F_1 = \{z_0, z_1, z_3\} : & \quad E_{F_1} = \overline{z_0 z_3}, \\
F_2 = \{z_0, z_2, z_3\} : & \quad E_{F_2} = \overline{z_0 z_3}, \\
F_3 = \{z_1, z_2, z_3\} : & \quad E_{F_3} = \begin{cases} \overline{z_1 z_3} & \text{if } t_T = 0, \\ \overline{z_2 z_3} & \text{else.} \end{cases}
\end{aligned}$$

We will see later in Corollary 3.10 that NVB in 3d dimensions leads to valid 2d NVB on the faces. Any face (2-simplex) refinement tree only depends on the initial refinement edge, as types do not matter in 2d. So two 3d direct neighbours will be compatible if the initial refinement edge on their shared face coincides.

From this insight follows the idea of Arnold et al. [AMP00] to first make elements compatible by choosing the initial refinement edge of each face as the longest edge. Then the initial edges coincide for each pair of neighbours. Afterwards they try to determine an internal ordering and type for each simplex, such that the initial refinement edge of each face coincides with the chosen edge.

Unfortunately, there are possibilities to label these edges that are not covered by standard NVB refinement. So they have to introduce non-standard initial elements and their respective refinement in order to combat the problem that arises from the fact that the simplices are not necessarily NVB simplices. Up to now this is the only possibility to relabel any 3d grid to allow (almost) NVB. This idea is not extensible to higher dimensions as faces are of dimension  $> 2$  and hence their refinement tree does not solely depend on the initial edge.

**Shape Regularity** In 2 dimensions NVB produces shape regular descendants since all  $T \in \mathfrak{F}(T_0)$  belong to at most four similarity classes; compare

with Figure 2.11. This is a consequence of the fact that NVB always bisects the angle at the newest vertex. In the end, any angle of any simplex is bisected at most once.

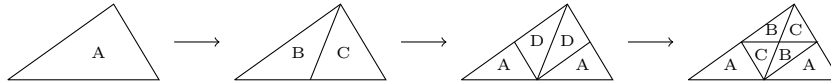


Figure 2.11: NVB: The four similarity classes for an initial element  $T_0$ .

This can be extended to any dimension  $d$ , see e.g. [Ste08]. This leads to the following statement.

**Lemma 2.34 (Strong Regularity of NVB).** *NVB is strongly regular in any dimension.*

*Proof.* [Ste08, Theorem 2.1] □

Unfortunately, as NVB is defined element-wise, compatibility is out of reach without introducing an additional compatibility condition. Also we want a compatibility condition that is constructible for any given mesh. This will be the topic of Section 3.2.

**Remark 2.35.** *In principle Longest Edge Bisection seems to be a reasonable choice initially, but it fails to show strong regularity in any dimension. If one could show that it converges into NVB, we would gain the benefits of both methods combined. Here convergence is meant in the sense that after refinement up to a certain element all further longest edges of descendants chosen for refinement, would also be chosen by a NVB labelling of the element.*

## 2.4 Refinement Closure

We only define conformity for simplex meshes as hierarchically refined cube grids are often non-conforming and the conforming cube meshes require a complex closure operation.

**Definition 2.36 (Conforming).** *A simplex triangulation  $\mathcal{T}$  is called conforming, if and only if the intersection  $F = T_1 \cap T_2$  of any two simplices  $T_1, T_2$*

is either a complete sub-simplex from both sides, i.e.,  $F \in \mathcal{T}^i(T_1), F \in \mathcal{T}^i(T_2)$  for an  $i \in \{0, \dots, d-1\}$ , or  $F = \emptyset$ .

In other words a triangulation is conforming if there are no hanging nodes, so vertices  $z \in T$ , but  $z \notin \mathcal{V}(T)$ .

Another important property for non-conforming adaptive grids is the *mesh balance*. Both conformity and mesh-balance lead to a *mesh grading* which is for instance required for inverse estimates and  $H^1$ -stability of the  $L^2$ -projection of finite-element spaces [GHS15].

**Definition 2.37 (Mesh Balance).** *A refined mesh  $\mathcal{T}$  originating from an initial grid  $\mathcal{T}_0$  is called balanced if and only if for any neighbours  $T_1, T_2 \in \mathcal{T}$  the difference in generation is uniformly bounded, i.e.,  $\exists c_{MB} > 0$  such that  $\forall T_1, T_2 \in \mathcal{T}$  with  $T_1 \cap T_2 \neq \emptyset$  it holds  $T_1 \in \mathcal{T}_i$  and  $T_2 \in \mathcal{T}_j$  with  $|i - j| < c_{MB}$ .*

Mesh-balance can also be defined just using direct neighbours. These definitions are exchangeable by replacing the  $c_{MB}$  with a different  $\tilde{c}$  that depends on the dimension.

For shape regular refinement strategies a balanced mesh and a small constant  $c_{MB}$  imply that neighbours do not differ much in size if the elements of  $\mathcal{T}_0$  are of similar size.

The challenge of mesh-balance and conformity is that they imply refinement propagation. If an element is refined, its neighbours may also need to be refined to keep the mesh balanced or conforming. So we need an iterative or recursive algorithm to resolve this propagation.

Several algorithms have been introduced in the case of NVB (e.g., [Kos94, Algorithm 1][Bän91, Mit88]). In the case of LEB there is e.g. the Longest Edge Propagation Path (LEPP) algorithm by Rivara [Riv97]. For isometric cube refinement we refer to [IBWG15, IBG12, SSB08].

To execute the refinement on a grid we use an iterative version (Algorithm 2.1), as it works for any of the presented refinement strategies and guarantees mesh-balance or conformity. An actual implementation of a refinement



strategy may use a better algorithm by exploiting special properties of the refinement strategy.

The algorithm expects input in the form of a set of elements  $\mathcal{M} \subset \mathcal{T}$  that have been marked for refinement (e.g., by an a posteriori estimator). Also it needs a codimension  $0 < c < d$ , that labels the entities on which we want to enforce the mesh-balance/conformity.

---

**Algorithm 2.1:** Iterative refinement algorithm

---

- 1 Given: The set of marked elements  $\mathcal{M} \subset \mathcal{T}$  and codimension  $c$ .
  - 2 Set  $\mathcal{M}_0 = \mathcal{M}$  and  $i = 0$
  - 3 **while**  $\mathcal{M}_i \neq \emptyset$  **do**
  - 4     Choose  $T \in \mathcal{M}_i$  Refine  $T$ .
  - 5     Check all subelements  $F \in \mathcal{T}^c(T)$  for mesh-balance (conformity)
  - 6     Add all non-balanced (non-conforming) neighbours to  $\mathcal{M}_{i+1}$
  - 7      $i = i + 1$
- 

The choice of the codimension is often  $c = d - 1$  or  $d = 1$ , so that either the faces  $\mathcal{F} = \mathcal{T}^{d-1}$  or the edges  $\mathcal{E} = \mathcal{T}^1$  propagate the refinement. The face case ( $c = d - 1$ ) simplifies the implementation, as one only needs to iterate over direct neighbours. The edge case ( $c = 1$ ) potentially reduces the total number of iterations of the algorithm for the conforming case, as more elements are added to  $\mathcal{M}_i$  in each step.

Algorithm 2.1 refines more elements than elements that have been marked. The set of additional elements that are refined is called *closure*.

**Definition 2.38 (Closure).** *Given a grid  $\mathcal{T}$  and a marked set  $\mathcal{M}$ . Then the (conforming) closure  $\mathcal{C}_{\mathcal{T}}(\mathcal{M})$  is the set of elements that are created during refinement additionally to the children of  $\mathcal{M}$ .*

This definition enables us to prove that Algorithm 2.1 terminates and obtains the minimal possible closure.

**Lemma 2.39 (Termination of Algorithm 2.1).** *Given a grid  $\mathcal{T}$  and a set of elements marked for refinement  $\mathcal{M}$ . If a finer triangulation  $\mathcal{T}_{fine}$  that*

contains the children of the elements in  $\mathcal{M}$  and is balanced (conforming) exists then Algorithm 2.1 terminates and results in the minimal closure  $\mathcal{C}_{\mathcal{T}}(\mathcal{M})$  that is required for mesh-balance (conformity).

*Proof. Termination of Algorithm 2.1:* The grid  $\mathcal{T}_{fine}$  is finer than  $\mathcal{T}$ , fulfils the required condition and all elements in  $\mathcal{M}$  are refined. The number of refinements needed to reach  $\mathcal{T}_{fine}$  is bounded. As we refine an element in each step, we can at most reach  $\mathcal{T}_{fine}$  and hence the algorithm terminates.

*Minimal Closure:* Assume there is a grid  $\mathcal{T}_{small}$  with a closure  $\mathcal{C}_{\mathcal{T}_{small}}(\mathcal{M})$  that is smaller than the closure obtained by Algorithm 2.1. Then there is a patch of elements in  $\mathcal{C}_{\mathcal{T}}(\mathcal{M}) \setminus \mathcal{C}_{\mathcal{T}_{small}}(\mathcal{M})$  that can be taken out while all elements in  $\mathcal{M}$  are refined and the required property is attained. So Algorithm 2.1 never added them to any  $\mathcal{M}_i$ . This is a contradiction and hence the proof.  $\square$

The following standard construction of the finer mesh needed for termination works for all refinement strategies, where any uniform refinement of the whole mesh results in a balanced/conforming situation.

**Remark 2.40 (Standard Construction of  $\mathcal{T}_{fine}$ ).** *Let  $m = \max_{T \in \mathcal{T}} \text{gen}(T)$  be the maximum level in the mesh  $\mathcal{T}$ . We uniformly refine the initial mesh  $\mathcal{T}_0$   $m$  times and call the resulting mesh  $\mathcal{T}_m = \mathcal{T}_{fine}$ . It is balanced (conforming) and obviously finer than  $\mathcal{T}$ .*

Adaptive algorithms often require *coarsening* in addition to refinement. This is needed for instance for time-dependent problems, where the error indicator may change in time. If the error indicator on some elements becomes too small in comparison to indicators in other areas of the grid it makes sense to replace the elements by their parents. This replacement process is called coarsening.

**Definition 2.41 (Coarsening).** *Let  $\mathcal{T}$  be the current triangulation and  $\mathfrak{U}\mathfrak{F}(\mathcal{T}) = \mathfrak{F}(\mathcal{T}_0) \setminus \mathfrak{F}(\mathcal{T})$  the part of the forest above the current triangulation  $\mathcal{T}$ . Given a set of elements marked  $\mathcal{CM} \subset \mathcal{T} \cup \mathfrak{U}\mathfrak{F}(\mathcal{T})$  for coarsening*

with the following restrictions: For each  $T \in \mathcal{CM} \cap \mathfrak{U}\mathfrak{F}(\mathcal{T})$ , we expect all children  $T' \in \mathcal{CM}$ . Also no  $T \in \mathcal{T}_0$  may be in  $\mathcal{CM}$ .

Coarsening creates a non-conforming/balanced triangulation  $\mathcal{T}_{coarse}$  as leaf triangulation of  $(\mathfrak{U}\mathfrak{F}(\mathcal{T}) \cup \mathcal{T}) \setminus \mathcal{CM}$ . Afterwards it closes the resulting mesh  $\mathcal{T}_{coarse}$  by enforcing either mesh-balance or conformity.

The coarsening procedure is described in Algorithm 2.2 formulates this procedure. We directly get termination of Algorithm 2.2, if Algorithm 2.1 terminates. Also the Algorithm 2.2 does not yield a mesh that is finer than the starting mesh  $\mathcal{T}$ , as the grid  $\mathcal{T}$  is finer than  $\mathcal{T}_{coarse}$  and conforming. In an actual implementation it should be checked whether an element will be recreated, before it removing it from  $\mathcal{T}$  to avoid unnecessary operations.

---

**Algorithm 2.2:** Coarsening algorithm

---

- 1 Given: coarsen marked elements  $\mathcal{CM} \subset \mathfrak{U}\mathfrak{F}(\mathcal{T}) \cup \mathcal{T}$  and codimension  $c$ .
  - 2 Create  $\mathcal{T}_{coarse}$  by removing  $\mathcal{CM}$  from  $\mathfrak{U}\mathfrak{F} \cup \mathcal{T}$
  - 3 Set  $\mathcal{M}$  as all non-conforming/balanced  $T \in \mathcal{T}_{coarse}$
  - 4 Execute Algorithm 2.1 with  $\mathcal{M}$  and codimension  $c$
- 

**Termination of Refinement** Algorithm 2.1 terminates for isometric refinement in both the cube and the simplex case because all grids  $\mathcal{T}_i, i \in \mathbb{N}_0$  are conforming. Hence the standard construction of  $\mathcal{T}_{fine}$  works (Remark 2.40). We briefly prove that Algorithm 2.1 terminates for LEB.

**Theorem 2.42 (Termination for LEB).** *Let  $\{\mathcal{E}_k\}_{k \in \mathbb{N}}$  be ordered and let  $\mathcal{M} \in \mathcal{T}$  be the set of marked elements. Then there is a finer grid  $\mathcal{T}_{fine}$  that is conforming.*

*Proof.* The first edge  $E \in \mathcal{E}$  (in the given ordering) is always the refinement edge of all adjacent elements. So refining all adjacent elements is conforming. So refining the first  $n$  edges is also conforming for any  $n \in \mathbb{N}$ . So let  $m$  be the position of the last refinement edge of an element  $T \in \mathcal{M}$ . Then refining the first  $m$  edges leads to a conforming mesh  $\mathcal{T}_{fine}$ , where all  $T \in \mathcal{M}$  have been refined.  $\square$

**Isometric Refinement Propagation** Isometric refinement often requires a 2:1 mesh-balance ([SSB08]), which is one of the following.

**Definition 2.43 (2:1 Mesh Balance).** *Given a codimension  $c = 1$  or  $c = d - 1$ , then a grid  $\mathcal{T}$  originating from a conforming  $\mathcal{T}_0$  fulfils the 2:1 Balance if and only if for each pair of elements  $T_1, T_2 \in \mathcal{T}$  that share a sub-element  $F = T_1 \cap T_2 \supset E \in \mathcal{T}^c$  of codimension  $c$ , then  $|\text{gen}(T_1) - \text{gen}(T_2)| \leq 1$ .*

Choosing  $c = d - 1$  is the “steeper” case, as the mesh-balance is only enforced on direct neighbours and not on arbitrary neighbours as for  $c = 1$ . In the case of  $c = d - 1$  edges may differ a lot in generation viewed from different neighbours and refinement does not propagate far. See Figure 2.12, where for  $c = d - 1$  balance is enforced on less elements than in the case of  $c = 1$ .

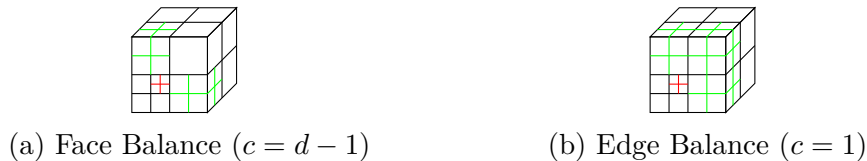


Figure 2.12: 2:1 Mesh-Balance for Cube Refinement in 3d. Black is the initial refinement situation. Red is the requested refinement and green is the closure.

**Green Closure** Isometric, non-conforming simplex (“Red”) refinement can be closed conformingly. This process is called “*Green*” closure. The combination of both is called *Red-Green Refinement*.

**Definition 2.44 (Green Closure).** *Let  $\mathcal{T}$  be a non-conforming simplex grid. The green closure is the smallest subtriangulation of non-conforming simplices into subsimplices, that is conforming.*

It is proven [Gra19], that the green closure is constructible in any dimension for non-conforming red refined simplex meshes.

If the green closure is refined further, shape regularity cannot be guaranteed. So before refinement is performed the green closure is resolved and

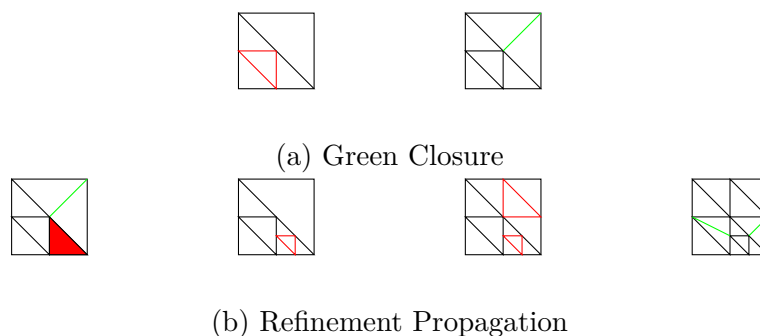


Figure 2.13: Construction of Green Closure and Refinement Propagation in 2d.

the previous red refinement situation is restored. Then the red refinement is performed with its refinement propagation and mesh-balance. Afterwards the green closure is applied again, see Figure 2.13.

**Recurrent Refinement of Triangulations with Bisection** Bisection-based refinement always aims to be conforming. This automatically implies a mesh-balance, if the choice of refinement edges implies shape regularity. So in this case we call the closure the conforming closure.

In principle one could minimize the closure by simply refining all marked simplices and then choosing the refinement edges of the non-conforming elements to be the already refined edges. Unfortunately this choice does not yield any shape regularity and elements tend to degenerate.

So the size of the conforming closure for LEB and NVB is not minimal. We will use the term *effort* to refer to the size of the conforming closure. We will see in the next chapter that we can get a bound on the effort depending on the compatibility condition for NVB.

# Chapter 3

## Weak Compatibility

This chapter focusses on the compatibility of adaptive triangulations using NVB refinement. So unless stated otherwise, the refinement strategy will be implicitly given as NVB. This chapter follows closely our articles [AGK18, AK17] and presents the results published therein.

First, we will investigate the behaviour of NVB refinement of a single element under several uniform refinements. To this aim we use the concept of a refinement tree, which allows us to have a cleaner look at the actual refinement structure.

This leads us to Lemmas 3.4, 3.6, 3.8, which state that lower-dimensional subsimplices also follow an NVB refinement pattern and also how this can be directly described from the element refinement pattern.

Then we define our novel compatibility condition, and compare it to the existing condition. The condition is weaker and has slightly less convenient properties, but on the other hand we design an algorithm to renumber an arbitrary-dimensional mesh to fulfil this property, which is not possible in a similarly straightforward manner for the stronger compatibility condition.

Finally we execute experiments to investigate the parameters involved in the designed algorithm.

### 3.1 NVB and Induced Refinement

As noted in Remark 2.32 reversing the order of  $\mathcal{V}_0$  leads to identical refinement trees. Also for  $d = 2$  the refinement tree solely depends on the initial refinement edge and is independent of the type. To factor out these identifications we define the following concept of *NVB-equivalence*, which defines a similar notion to the *tagged* simplices in [Ste08]. These concepts coincide for  $d > 2$ .

**Definition 3.1 (NVB-equivalence).** *A  $d$ -simplex  $T$  is NVB-equivalent to a simplex  $T'$ , if their refinement trees match, i.e.  $\mathcal{RT}(T) = \mathcal{RT}(T')$ .*

Recall that the refinement tree of  $T$  is completely defined by the ordering and the type of  $T = [z_0, \dots, z_d]_t$ . So for any combination of ordering and type there is a refinement tree. The other direction does not hold. So if a result yields a refinement tree and we want to know, whether it belongs to an NVB refined simplex, we need to prove, that it belongs to one of the possible combinations of ordering and type.

To be able to depict refinement trees of type 0 and 1 simplices, we identify subtrees within the refinement tree.

**Remark 3.2 (Identification of subtrees).** *Let  $T = [z_0, \dots, z_d]_t$  with  $t \in \{0, \dots, d-2\}$  and  $d > 2$ . We denote the center of a refinement edge between  $z_i$  and  $z_j$  as  $z_{ij} = \frac{z_i + z_j}{2}$  and use the identification from Remark 2.31. Then  $T$  corresponds to the two ordered sets*

$$\mathcal{V}_0 = [z_0, z_{t+1}, \dots, z_d], \mathcal{V}_1 = [z_1, \dots, z_t].$$

The children  $T_0, T_1$  of  $T$  then correspond to

$$\begin{aligned} T_0 : \mathcal{V}_0 \setminus \{z_0\} &= [z_{t+1}, \dots, z_d], \mathcal{V}_1 = [z_{0d}, z_1, \dots, z_t] \\ T_1 : \mathcal{V}_0 \setminus \{z_d\} &= [z_0, z_{t+1}, \dots, z_{d-1}], \mathcal{V}_1 = [z_{0d}, z_1, \dots, z_t]. \end{aligned}$$

Now in the second generation the second child  $T_{10}$  of the first child corresponds

to

$$\mathcal{V}_0 = [z_{t+1}, \dots, z_{d-1}], \mathcal{V}_1 = [z_{0(d-1)}, z_{0d}, z_1, \dots, z_t]$$

and the second child of the second child  $T_1$  corresponds to

$$\mathcal{V}_0 = [z_{t+1}, \dots, z_{d-1}], \mathcal{V}_1 = [z_{(t+1)d}, z_{0d}, z_1, \dots, z_t].$$

So until  $\mathcal{V}_1$  is merged back into  $\mathcal{V}_0$  the refinement behaviour of these two simplices coincides, as  $\mathcal{V}_0$  coincides. The merge happens, when  $\#\mathcal{V}_1 = d$ , which is why we need the restrictions on type and dimension.

The identification of subtrees allows us to represent the refinement tree for types 0, 1 in a condensed fashion by identifying subtrees of actually distinct elements.

**Example 3.3 (Condensed Refinement Tree for Type 0).** *Figure 3.1 shows the condensed refinement tree of the simplex  $T = [z_0, \dots, z_d]_0$ . It is condensed, which means that the number of elements at a node is the number of paths leading to this node, i.e.,  $\binom{l}{k}$  with the level  $l$  and  $k$  the position within the row. So each row consists of  $2^l$  descendants. One can easily see that this is also the refinement tree of  $T' = [z_d, z_{d-1}, \dots, z_1, z_0]_0$ , so  $T$  is NVB-equivalent to  $T'$ .*

This example and in particular Figure 3.1 help us to investigate the induced refinement trees of lower-dimensional subsimplices of a type 0 element under refinement.

**Lemma 3.4 (Type 0).** *Let  $T = [z_0, \dots, z_d]_0$  be a  $d$ -simplex of type  $t_T = 0$  that is bisected  $d$  times uniformly.*

1. *Then every edge  $E \in \mathcal{T}^1(T)$  gets bisected exactly once and no newly created edge gets bisected.*
2. *Any sub-simplex  $F \in \mathcal{T}^{d-1}(T)$  is bisected by a valid NVB uniformly  $d - 1$  times. It is of type  $t_F = 0$  and its ordering coincides with the ordering of  $T$  up to NVB-equivalence.*



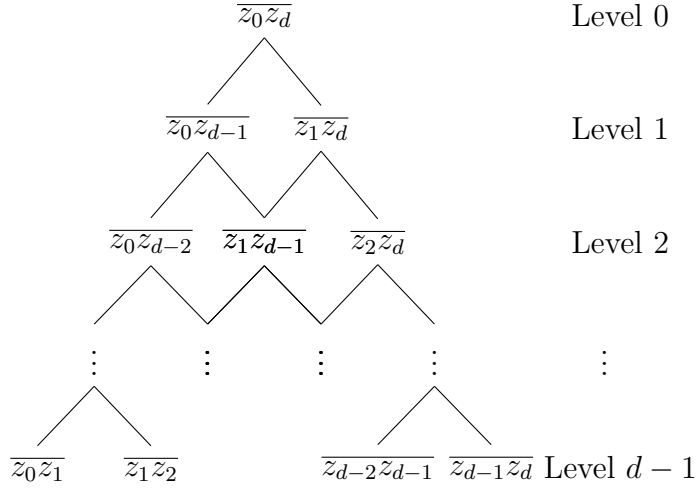


Figure 3.1: Condensed refinement tree of a  $d$ -simplex of type 0 under  $d$  uniform refinements.

*Proof.* The refinement edges of  $T$  are depicted in Figure 3.1. The distance of the indices decreases by 1 each level, so any edge  $\overline{z_i z_j}$ ,  $i < j$  is bisected exactly once at level  $d - j + i$ . Also none of the vertices in this tree are midpoints of former bisections, so no newly created edges are bisected. This proves the first claim.

For the second claim we note that for the  $d$  uniform NVB-refinements of the simplex  $T$ , the refinement edges of the induced refinement of the face

$$F_i = \text{conv}\{z_0, \dots, z_{i-1}, z_{i+1}, \dots, z_d\} \in \mathcal{T}^{d-1}(T)$$

are refinement edges of the refinement tree of  $T$ . Moreover the children of the edge  $\overline{z_n z_m}$  in the induced refinement tree of  $F_i$  are  $\overline{z_n z_{m-1}}$  (or  $\overline{z_n z_{m-2}}$  if  $m - 1 = i$ ) and  $\overline{z_{n+1} z_m}$  (or  $\overline{z_{n+2} z_m}$  if  $n + 1 = i$ ). From this we can clearly see that  $F_i$  is NVB-equivalent to  $[z_0, \dots, z_{i-1}, z_{i+1}, \dots, z_d]_0$ .  $\square$

**Remark 3.5.** *This implies by induction that for  $0 \leq i \leq d - 1$  all elements of  $\mathcal{T}^i(T)$  with  $t_T = 0$  are valid NVB simplices of type 0.*

We proceed with a variant of Lemma 3.4 for elements of type  $t_T = 1$ .

**Lemma 3.6 (Type 1).** *Let  $T = [z_0, z^*, z_1, \dots, z_{d-1}]_1$  be a  $d$ -simplex of type  $t_T = 1$  that is bisected  $d$  times uniformly. Then,*

1. *every edge  $E \in \mathcal{T}^1(T)$  gets bisected exactly once and no newly created edge gets bisected.*
2. *Any sub-simplex  $F \in \mathcal{T}^{d-1}(T)$  is of type  $t_F = 1$ , if it contains  $z^*$  and of type 0 otherwise. The ordering of  $F$  coincides with the ordering of  $T$  up to NVB equivalence.*

*Proof.* The proof follows similar arguments as the proof of Lemma 3.4.

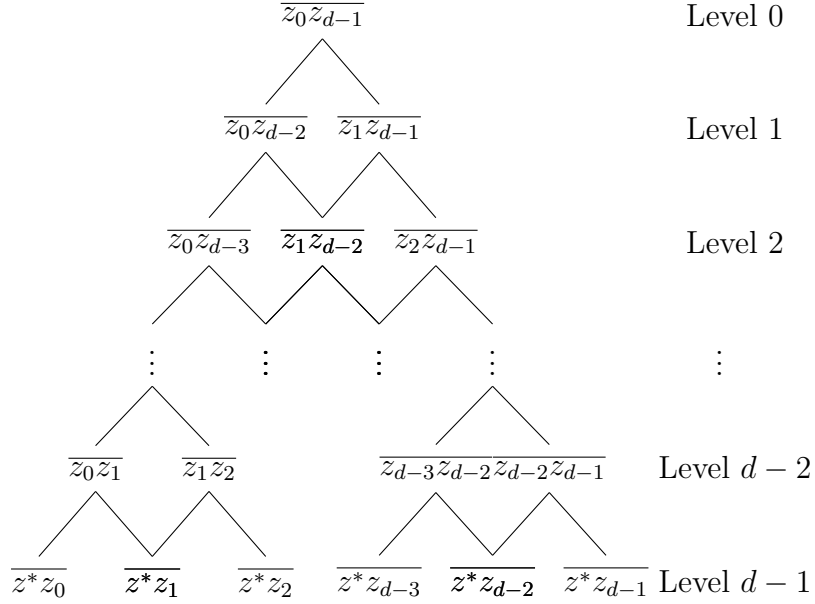


Figure 3.2: Condensed refinement tree of a  $d$ -simplex of type 1.

Figure 3.2 shows the refinement edges of  $T$ . The distance between the indices decreases by 1 each level until level  $d - 2$  and the final level consists of all edges containing  $z^*$ . So any edge  $\overline{z_i z_j}, i < j$  is bisected exactly once at level  $d - 1 - j + i$  and any edge  $\overline{z_i z^*}$  is bisected exactly once at level  $d - 1$ . No edge containing a newly created point is bisected, so this proves the first claim.

For the second claim we note that for the  $d$  uniform NVB-refinements of the simplex  $T$  the refinement edges of the induced refinement of the face

$$F_i = \text{conv}\{z_0, z^*, z_1, \dots, z_{i-1}, z_{i+1}, \dots, z_{d-1}\} \in \mathcal{T}^{d-1}(T)$$

are refinement edges of the refinement tree of  $T$ . Moreover the children of the edge  $\overline{z_n z_m}$  in the induced refinement tree of  $F_i$  are  $\overline{z_n z_{m-1}}$  (or  $\overline{z_n z_{m-2}}$  if  $m-1 = i$ ) and  $\overline{z_{n+1} z_m}$  (or  $\overline{z_{n+2} z_m}$  if  $n+1 = i$ ) for  $m \neq n+1$ . For the case  $m = n+1$  the children are  $\overline{z^* z_n}$  and  $\overline{z^* z_m}$ . Finally for the case  $n = i-1, m = i+1$  the children are  $\overline{z^* z_n}$  and  $\overline{z^* z_m}$ . From this we can clearly see that  $F_i$  is NVB-equivalent to  $[z_0, z^*, \dots, z_{i-1}, z_{i+1}, \dots, z_d]_1$ .

The face  $F' = \text{conv}\{z_0, z_1, \dots, z_{d-1}\}$  is NVB-equivalent to the  $(d-1)$ -simplex  $[z_0, z_1, \dots, z_{d-1}]_0$  since every edge  $E = \overline{z_i z_j}$  gets bisected at level  $d-1-j+i$ . This proves the second claim.  $\square$

**Remark 3.7.** *We can iterate Lemma 3.6 in combination with Lemma 3.4 to see that for a  $d$ -simplex  $T$  with  $t_T = 1$  any lower dimensional sub-simplex  $F \in \mathcal{T}^i(T), 2 \leq i \leq d-1$  is of type  $t_F = 1$  if  $z^* \in F$  and  $t_F = 0$  otherwise.*

For types  $t_T = 0, 1$  we have the convenient result that the number of uniform refinements to bisect every edge is  $d$ . For types  $t_T \geq 2$ , which implies  $d \geq 3$ , this is unfortunately not true, compare Figure 2.10c.

**Lemma 3.8 (Type  $\geq 2$ ).** *Let  $T = [z_0, z_1^*, \dots, z_t^*, z_1, \dots, z_{d-t}]_t$  be a  $d$ -simplex of type  $t_T = t \geq 2$  that is bisected  $2d-t$  times uniformly and let  $\mathcal{V}_1 = \{z_1^*, \dots, z_t^*\}$ .*

- *Then every edge of  $T$  gets bisected at least once.*
- *Any face  $F \in \mathcal{T}^{d-1}(T)$  is of type  $t_F = \#(T' \cap \mathcal{V}_1)$ . The ordering of  $F$  coincides with the ordering of  $T$  up to NVB equivalence.*

*Proof.* Bisecting the element  $T = [z_0, z_1^*, \dots, z_t^*, z_1, \dots, z_{d-t}]_t$  uniformly  $d-t$  times leads to a set of descendants of type 0 that look as follows

$$T' = [\bar{z}_0, \bar{z}_1, \dots, \bar{z}_{d-t}, z_1^*, \dots, z_t^*]_0$$

where  $\bar{z}_k$  are either vertices of  $T$  or centers of refinement edges of  $T$ . This means that none of the refinement edges  $E_{ij} = \overline{z_i^* z_j^*}$ ,  $1 < i < j < t$  has been refined yet. From the proof of Lemma 3.4 we know that  $E_{ij}$  will be refined at level  $d - j + i$  of  $T'$  and hence at level  $d - t + d - j + i = 2d - t - j + i$  of  $T$ . In particular for  $E_{12}$  this means  $2d - t - 1$ , so after  $2d - t$  uniform bisections every edge in  $T$  has been refined at least once.

For the second claim we construct an artificial ancestor  $\tilde{T} = [z_0, \dots, z_d]_0$  by assuming that the simplex  $T$  has been created by  $t$  bisections always being the first child from Definition 2.30.

$$T_0 = \left[ z_0, \frac{z_0 + z_d}{2}, z_1, \dots, z_{d-1} \right]_{t_T + 1 \bmod d}$$

This does not impose a loss of generality, as this ancestor may always be constructed, as we can construct the parent by adding the missing half.

From Lemma 3.4 we know that the ordering of all subsimplices of  $\tilde{T}$  coincides with the ordering of  $\tilde{T}$  and this translates to the descendant. Furthermore  $\tilde{T}$  has been refined  $t$  times introducing the vertices  $\mathcal{V}_i$ , which means that any subsimplex  $\tilde{T}'$  has been refined as many times as it contains vertices from  $\mathcal{V}_1$ , which proves the second claim.

In particular for  $T = [z_0, z_1^*, \dots, z_t^*, z_1, \dots, z_{d-t}]_t$  any face  $T' \in \mathcal{T}^{d-1}(T)$  is of type  $t_{T'} = \#\{z_i^* | z_i^* \in T'\}$  since the subsimplex of the artificial ancestor has been refined as many times.  $\square$

**Remark 3.9.** *Lemma 3.8 specializes Remark 3.4 of [GSS14], which states, that in general one cannot expect every edge of a simplex to have been bisected after  $d$  uniform bisections after considering the counterexample in 3d with type 2 (cf. Figure 2.10c).*

The Lemmata 3.4, 3.6 and 3.8 lead to the following corollary by simple induction, see also Definition 2.9.

**Corollary 3.10 (Trace Meshes).** *The lower dimensional sub-meshes (trace meshes)  $\mathcal{T}_d^i$ , for any positive dimension  $0 < i < d$  can be seen as standard*

NVB meshes of their own. This means, that the refinement induced on these meshes by refining the whole mesh  $\mathcal{T}$  coincides with refining the trace meshes by a valid NVB refinement for a certain initial configuration.

This means for any simplex  $T$  with type  $0 \leq t_T \leq d-1$  for  $0 \leq i \leq d-1$  all elements of  $\mathcal{T}^i(T)$  carry a valid NVB refinement structure.

The following simple properties of NVB are useful for initial grids  $\mathcal{T}_0$  containing only elements of type 0.

**Lemma 3.11 (Properties of NVB).**

1. For  $T \in \mathfrak{F}(T_0)$  with  $T_0 \in \mathcal{T}_0$  we have

$$h_T = 2^{-\text{gen}(T)/d} h_{T_0}.$$

2. Define  $\alpha_0 := \max\{\#\mathcal{T}_{z_0} \mid z_0 \in \mathcal{V}_0\}$ . Then we have for  $z \in \mathcal{V}$  the bound

$$\#\mathcal{T}_z \leq 2^{d-1} \alpha_0$$

*Proof.* Bisection halves the volume of a simplex. The definition of  $\text{gen}(T)$  then gives the first claim. During refinement any angle is bisected at most  $d-1$  times (Lemma 3.4), which yields the second assertion.  $\square$

## 3.2 Compatibility Conditions

As refinement is defined element-wise with NVB, we have to define a compatibility condition that couples neighbours. The weakest possible condition is Condition 2.26, which enforces that the induced refinement of a face is the same from both sides. This is called *compatibly divisible* in [Ste08] or *conformingly marked* in [AMP00] (for  $d=3$ ). This condition is necessary, but not obviously sufficient for the iterative NVB algorithm 2.1 to terminate. So we define a slightly stronger condition and compare it to the standard strong condition. In addition to Condition 2.26 it also requires a finer triangulation, where each element is either of type 0 or 1. Then every  $d$ -th uniform refinement is conforming and we can easily prove termination of Algorithm 2.1.

The compatibility condition should be fulfilled by the initial triangulation  $\mathcal{T}_0$ .

The new condition is motivated as the strong compatibility is not applicable to an arbitrary mesh, but only to a subclass: It can be applied to meshes generated from Kuhn-cubes (cf. Figure 2.1)[SS05]. Also, for an arbitrary mesh it is possible to subdivide each element  $\frac{1}{2}(d+1)!$  times and arrive at a strongly conforming situation [Ste08, Appendix A],[Kos94].

### 3.2.1 Weak Compatibility Condition

The weak condition, that we now introduce, has the advantage to be applicable to any conforming simplex mesh as we will show later.

**Definition 3.12 (Weak Compatibility Condition).** *A triangulation  $\mathcal{T}$  is called weakly compatible, if and only if it fulfils Condition 2.26 and there exists a finer triangulation that is conforming and only contains elements of type  $t \in \{0, 1\}$ .*

**Remark 3.13.** *Any 2-dimensional triangulation is weakly compatible, as it only contains elements of type 0 and 1.*

For the termination of Algorithm 2.1 it is sufficient to have a finer mesh  $\mathcal{T}_{fine}$  that is conforming. The following lemma states that if an initial grid is weakly compatible, then there always is a finer mesh, so refinement always terminates.

**Lemma 3.14 (Every  $d$ -th uniform refinement is conforming).** *Let  $\mathcal{T}$  be conforming and only contain elements of type  $t \in \{0, 1\}$ . Then every  $d$ -th uniform NVB refinement of all elements  $T \in \mathcal{T}$  is conforming and also only contains elements of type  $t \in \{0, 1\}$ .*

*Proof.* We split the triangulation  $\mathcal{T}$  into its elements  $T$  and investigate each  $\mathcal{T}(T)$ . Every triangulation  $\mathcal{T}(T)$  is refined  $d$  times uniformly. As all elements are type 0 or 1, we know from Lemmas 3.4 and 3.6 that all  $d-1$ -dimensional subsimplices  $F \in \mathcal{T}^{d-1}(T)$  have been uniformly refined  $d-1$  times. Now we

reassemble the triangulation  $\mathcal{T}$  from the element-triangulations  $\mathcal{T}(T)$ . Condition 2.26 ensures that subsimplices that are shared by two elements carry the same refinement structure. So the resulting triangulation is conforming.  $\square$

Using Lemma 2.39 this leads to the following corollary.

**Corollary 3.15.** *The iterative NVB refinement algorithm (Algorithm 2.1) terminates on weakly compatible grids for any set  $\mathcal{M}$  of elements marked for refinement.*

**Remark 3.16.** *The recursive algorithm [Mit88, Algorithm 2.1] does in general not terminate under these conditions, as neither Condition 2.26 nor Condition 3.12 imply that the grid is loop-free. This means, that refinement of an element may propagate into the element itself from a different direction. As the recursive algorithm always tries to refine the non-conforming neighbour first, it cannot terminate as it runs into an infinite loop and never refines.*

### 3.2.2 Strong Compatibility Condition

As we will refer to the standard compatibility condition as strong compatibility condition we also introduce the notion of *reflected neighbours* and the *strong compatibility condition* itself. This is taken from [Ste08].

**Definition 3.17 (Reflected Neighbours).** *Two direct neighbours  $T = \text{conv}\{z_0, \dots, z_{d-1}, u\}$  and  $T' = \text{conv}\{z_0, \dots, z_{d-1}, v\}$  are called reflected neighbours, if and only if the types  $t_T = t_{T'}$  and the ordering coincide with  $u$  and  $v$  being at the same position up to NVB equivalence.*

Reflected neighbours have the same refinement structure on their shared face  $F$  and hence, if the refinement edge  $E$  is contained in the shared face  $F$ , it is the same from both sides. See Figure 3.3 for an illustration. These neighbours are called reflected, because the refinement structure inside the elements is a reflection of each other. The strong compatibility condition, which relies on reflected neighbours, does not only impose a condition on

the shared face, but on the complete refinement structure of neighbouring elements.

**Example 3.18 (Reflected Neighbours in 2d).** *We consider two direct neighbours  $T_1 = \{z_0, z_1, z_2\}$  and  $T_2 = \{z_1, z_2, z_3\}$ . As types do not matter for  $d = 2$ , we choose  $t = 0$ . Also we only consider one representative of NVB-equivalent simplices. These elements are reflected neighbours in the cases*

$$\begin{aligned} T_1 &= [z_0, z_1, z_2]_0 & T_2 &= [z_3, z_1, z_2]_0, \\ T_1 &= [z_0, z_2, z_1]_0 & T_2 &= [z_3, z_2, z_1]_0, \\ T_1 &= [z_1, z_0, z_2]_0 & T_2 &= [z_1, z_3, z_2]_0. \end{aligned}$$

See Figure 3.3 for illustration. The children are reflected neighbours in the cases

$$\begin{aligned} T_1 &= [z_0, z_1, z_2]_0 & T_2 &= [z_1, z_2, z_3]_0, \\ T_1 &= [z_0, z_2, z_1]_0 & T_2 &= [z_2, z_1, z_3]_0. \end{aligned}$$

This is depicted in Figure 3.4.

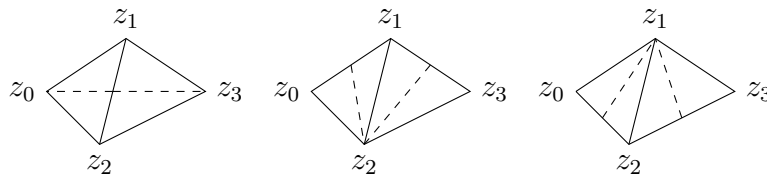


Figure 3.3: Refinement behaviour of reflected neighbours in 2d

The following lemma shows, that reflected neighbours and direct neighbours whose children are reflected neighbours appear naturally during NVB refinement. It states that the two “middle” grandchildren are reflected neighbours.



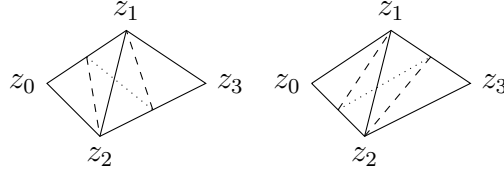


Figure 3.4: Children of neighbouring elements that are reflected neighbours, dotted line represents refinement of neighbouring children

**Lemma 3.19 (Grandchildren are Reflected Neighbours).** *Let  $T = [z_0, z_1, \dots, z_d]_{t_T}$ . The two children*

$$T_0 = [z_0, z_{0d}, z_1, \dots, z_{d-1}]_{t_T+1 \bmod d}$$

$$T_1 = [z_d, z_{0d}, z_1, \dots, z_{t_T}, z_{d-1}, \dots, z_{t_T+1}]_{t_T+1 \bmod d}$$

*are reflected neighbours for type  $t_T = d - 1$ . For type  $0 \leq t_T \leq d - 2$  they are not reflected neighbours, but their children that share the face  $F = T_0 \cap T_1$  are.*

*Proof.* The first statement follows directly by inserting  $t_T = d - 1$  into the definition of  $T_1$ .

To prove the second statement we consider the grandchildren

$$T_{00} = [z_0, z_{0d-1}, z_{0d}, z_1, \dots, z_{d-2}]_{t_T+2 \bmod d}$$

$$T_{01} = [z_{d-1}, z_{0d-1}, z_{0d}, z_1, \dots, z_{t_T}, z_{d-2}, \dots, z_{t_T+1}]_{t_T+2 \bmod d}$$

$$T_{10} = [z_d, z_{dt_T+1}, z_{0d}, z_1, \dots, z_{t_T}, z_{d-1}, \dots, z_{t_T+2}]_{t_T+2 \bmod d}$$

$$T_{11} = [z_{t_T+1}, z_{dt_T+1}, z_{0d}, z_1, \dots, z_{t_T}, z_{t_T+2}, \dots, z_{d-1}]_{t_T+2 \bmod d}.$$

The grandchild  $T_{11}$  is NVB-equivalent (compare Remark 2.32) to

$$T_{11} = [z_{d-1}, z_{dt_T+1}, z_{0d}, z_1, \dots, z_{t_T}, z_{d-2}, \dots, z_{t_T+1}]_{t_T+2 \bmod d}$$

which is a reflected neighbour of  $T_{01}$  at the face  $F = T_0 \cap T_1$ . □

The concept of reflected neighbours inherits to the children. Once two direct neighbours are reflected, all descendants at the shared face, that are direct neighbours, will be reflected.

**Lemma 3.20 (Children of Reflected Neighbours).** *Let  $T, T'$  be reflected neighbours with a shared face  $F$ . Then their children  $T_1 \subset T$  and  $T'_1 \subset T'$  with  $T_1 \cap F = T'_1 \cap F \neq \emptyset$  are reflected neighbours.*

*Proof.* Let  $0 \leq t < d$ ,  $0 < i < d$  and (up to NVB-equivalence)

$$\begin{aligned} T &= [z_0, \dots, z_{i-1}, u, z_i, \dots, z_{d-1}]_t \\ T' &= [z_0, \dots, z_{i-1}, v, z_i, \dots, z_{d-1}]_t. \end{aligned}$$

Then the shared face  $F = \{z_0, \dots, z_{d-1}\}$  contains the refinement edge  $\overline{z_0 z_{d-1}}$  and all children contain half of the face. These are

$$\begin{aligned} T_0 &= [z_0, z_{0d-1}, z_1, \dots, z_{i-1}, u, z_i, \dots, z_{d-2}]_{t+1 \bmod d} \\ T'_0 &= [z_0, z_{0d-1}, z_1, \dots, z_{i-1}, v, z_i, \dots, z_{d-2}]_{t+1 \bmod d} \end{aligned}$$

and if  $t \leq i - 1$

$$\begin{aligned} T_1 &= [z_{d-1}, z_{0d-1}, z_1, \dots, z_t, z_{d-2}, \dots, z_i, u, z_{i-1}, \dots, z_{t+1}]_{t+1 \bmod d} \\ T'_1 &= [z_{d-1}, z_{0d-1}, z_1, \dots, z_t, z_{d-2}, \dots, z_i, v, z_{i-1}, \dots, z_{t+1}]_{t+1 \bmod d} \end{aligned}$$

or if  $t \geq i$

$$\begin{aligned} T_1 &= [z_{d-1}, z_{0d-1}, z_1, \dots, z_{i-1}, u, z_i, \dots, z_t, z_{d-2}, \dots, z_{t+1}]_{t+1 \bmod d} \\ T'_1 &= [z_{d-1}, z_{0d-1}, z_1, \dots, z_{i-1}, v, z_i, \dots, z_t, z_{d-2}, \dots, z_{t+1}]_{t+1 \bmod d} \end{aligned}$$

and are reflected, because  $u$  is at the same position in children of  $T$  as  $v$  in children of  $T'$ .

Now let  $0 \leq t < d$ , and (up to NVB-equivalence)

$$T = [z_0, \dots, z_{d-1}, u]_t$$

$$T' = [z_0, \dots, z_{d-1}, v]_t.$$

Then the refinement edges  $\overline{uz_0}$  and  $\overline{vz_0}$  are not contained in the shared face  $F = \{z_0, \dots, z_{d-1}\}$  and the children that contain  $F$  are

$$T_1 = [z_0, \frac{u+z_0}{2}, z_1, \dots, z_{d-1}]_{t+1 \bmod d}$$

$$T'_1 = [z_0, \frac{v+z_0}{2}, z_1, \dots, z_{d-1}]_{t+1 \bmod d}$$

which are also reflected neighbours. □

In addition to reflected neighbours (Figure 3.3), we include elements in the definition of compatibility, whose children are reflected neighbours (Figure 3.4). This definition is called local as it allows for any pair of direct neighbours to decide whether they are strongly compatible.

**Definition 3.21 (Local Strong Compatibility).** *A face  $F \in \mathcal{T}^{d-1}$  is called strongly compatible if and only if the two adjacent elements are reflected neighbours, or their direct children, that are adjacent to  $F$  are reflected neighbours.*

The allowed combinations in 2d under Condition 3.21 are depicted in Figures 3.3 and 3.4. All other combinations are forbidden, see Figure 3.5.

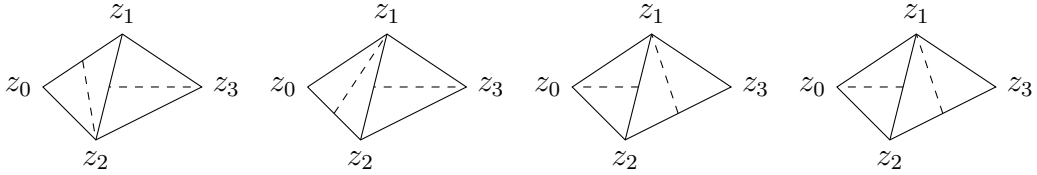


Figure 3.5: Forbidden orientations by Condition 3.21

We define a grid to be strongly compatible, if all pairs of direct neighbours in the grid are strongly compatible.

**Definition 3.22 (Strong Compatibility Condition).** *A conforming triangulation  $\mathcal{T}$  is strongly compatible, if all faces  $F \in \mathcal{T}^{d-1}$  are strongly compatible.*

**Remark 3.23.** *Condition 3.22 implies that all elements in the grid are of the same type, because two elements can only be reflected neighbours if they have the same type. Direct neighbours, whose children are reflected neighbours, also need to have the same type as their children need to coincide in type.*

Mitchell has shown that a distribution of refinement edges, such that Definition 3.22 holds, can be found for any initial triangulation  $\mathcal{T}_0$  [Mit88, Theorem 2.9] in 2 dimensions; compare also with [BDD04, Lemma 2.1]. Traxler has shown in [Tra97, Section 6] that in higher dimensions a necessary and sufficient condition is “all inner hyperedges of that domain have even index”. A hyperedge in his work corresponds to an element of  $\mathcal{T}^{d-2}$  and the index is the number of simplices containing this hyperedge. This condition is usually achieved in practice by filling the domain with Kuhn-cubes [SS05].

We show that the strong compatibility condition actually includes Condition 2.26, because all elements that refine an edge have the same generation.

**Lemma 3.24 (Edge Refinement).** *For any edge  $E \in \{\mathcal{E}_k\}_{k \in \mathbb{N}_0}$  and two elements  $T, T' \in \mathfrak{F}(\mathcal{T})$  with refinement edge  $E$  the generations of these elements coincide  $\text{gen}(T) = \text{gen}(T')$  under Condition 3.22.*

*Proof.* We know that a pair of direct neighbours  $T, T' \in \mathfrak{F}(\mathcal{T})$  with  $\text{gen}(T) = \text{gen}(T')$  and shared face  $F = T \cap T'$  is reflected, or the direct children that contain  $F$  are reflected. If they are not reflected, the refinement edge cannot be contained in  $F$ , as otherwise no child could contain  $F$ .

For any  $E \in \{\mathcal{E}_k\}_{k \in \mathbb{N}_0}$  and any element  $T \in \mathfrak{F}(\mathcal{T}_0)$  with  $E \in T$  as refinement edge the edge  $E$  is contained in  $d - 1$  faces of  $T$  with  $d - 1$  different reflected neighbours  $T' \in \mathfrak{F}(\mathcal{T}_0)$  that have  $E$  as refinement edge and  $\text{gen}(T) = \text{gen}(T')$ . As the grid is simply connected we can iterate over neighbours of neighbours and find all elements that have  $E$  as refinement edge.  $\square$

As all elements refining an edge need to be of the same generation, and refining from  $\mathcal{T}_i$  to  $\mathcal{T}_{i+1}$  refines all elements of generation  $i$ , we get the following corollary.

**Corollary 3.25 (Every uniform refinement is conforming).** *Let  $\mathcal{T}$  be conforming and fulfil Condition 3.22. Then every uniform NVB refinement is conforming. So for any  $k \in \mathbb{N}_0$  the grid  $\mathcal{T}_k$  is conforming.*

Lemma 3.24 is the key to show the following property of NVB; compare with [Ste08, Corollary 4.6].

**Lemma 3.26 (Characteristics of NVB).** *Suppose that the initial triangulation  $\mathcal{T}_0$  satisfies Condition 3.22. Let  $\mathcal{T} \in \mathfrak{F}(\mathcal{T}_0)$  conforming be given and suppose that  $T, T' \in \mathcal{T}$  are neighbours such that the common edge  $E \subset T \cap T'$  is the refinement edge of  $T$ . Then we either have  $\text{gen}(T') = \text{gen}(T)$  and  $E$  is also the refinement edge of  $T'$ , or  $\text{gen}(T') = \text{gen}(T) - i$  with  $1 \leq i < d$ .*

*Proof.* We have proven the first claim for  $\text{gen}(T) = \text{gen}(T')$  in the proof of Lemma 3.24.

The second claim for  $\text{gen}(T) \neq \text{gen}(T')$  is a consequence of all initial elements being of the same type and Lemma 3.4 that after  $d$  uniform refinements all edges of a type 0 element have been refined, so every  $d$ -th uniform refinement consists of elements of type 0.  $\square$

A simple consequence is  $|\text{gen}(T) - \text{gen}(T')| \leq d - 1$  for neighbours  $T, T' \in \mathcal{T}$ . We will exploit these facts in the analysis in Chapter 4.

Finally we extend the local strong compatibility to include direct neighbours that differ by one in type. This will be useful, when analysing the algorithm we design to reach a weakly compatible situation.

**Definition 3.27 (Local Quasi-strong Compatibility).** *Let  $F \in \mathcal{T}^{d-1}$  have the two adjacent elements  $T, T'$  that differ by one in type with  $t_{T'} = (t_T + 1) \bmod d$ . Then  $F$  is called quasi-strongly compatible if and only if the child of  $T$  that contains  $F$  is a reflected neighbour of  $t_{T'}$ .*

This definition mimics the behaviour within adaptively refined strongly compatible grids  $\mathcal{T}_0$  with conforming closure. All initial elements  $T \in \mathcal{T}_0$  in these grids are of the same type. In a conforming triangulation  $\mathcal{T} \in \mathfrak{F}(\mathcal{T}_0)$  any pair of direct neighbours can only differ by one in generation and hence also only differ by one in type (modulo  $d$ ). Refining a strongly compatible initial grid  $\mathcal{T}_0$  adaptively with conforming closure leads to elements fulfilling Definition 3.27.

### 3.2.3 Sufficiency of Weakest Compatibility Condition

We show that for NVB the Weakest Compatibility Condition 2.26 is sufficient for termination of Algorithm 2.1. Up to now we only know that the condition is necessary but it does not guarantee that there always is a finer mesh which then yields termination. So we need to ensure that refinement does not propagate back into the original element and refines an edge twice, which could lead to an infinite refinement loop, by always looping back into the original element and refining again.

First we show that we have a bound on the type difference.

**Lemma 3.28 (Type Difference of Direct Neighbours).** *Let  $T_1, T_2 \in \mathcal{T}$  with  $T_1 \cap T_2 = F \in \mathcal{F}$ . Then under Condition 2.26 the types  $t_{T_1}, t_{T_2}$  can at most differ by one (modulo  $d$ ):*

$$|t_{T_1} - t_{T_2}| \in \{d - 1, 0, 1\}$$

*Proof.* For  $d \leq 3$ , this is trivial. For  $d > 3$ , this is a direct consequence of Lemmas 3.4, 3.6, 3.8 and the fact that the elements only differ by one vertex.  $\square$

**Remark 3.29.** *The above theorem basically exploits the fact that the shared face between two elements has a unique type for  $d > 3$  (face dimension  $> 2$ ) and it can either be the type of the neighbour or one less (mod  $d$ ). For  $d = 3$  (face dimension 2) this does not work, as the refinement trees of 2-simplices may coincide for type 0 and type 1.*

We define refinement infliction, which will also be of interest when considering distributed refinement in Chapter 4, as it is similar to the concept of a refinement propagation graph.

The situation is the following: We have refined an edge  $E$  and now there are non-conforming adjacent elements. At these elements we request the refinement of this edge  $E$  for the mesh to be conforming.

Looking at the refinement tree from bottom to top we observe the necessary refinement that is required to refine a requested edge  $E$ . I.e., all edges in paths within the tree leading to  $E$  have to be refined both within the element itself and within all elements that share the edge.

**Definition 3.30 (Refinement Infliction).** *Inflicted Refinement  $\mathcal{IR}(E, T)$  of an edge  $E$  within the element  $T$  is the set of all  $E \in \mathcal{E}(T)$  that have to be refined before  $E$  is refined, i.e. the nodes of all paths in  $\mathcal{RT}(T)$  leading to  $E$ .*

**Lemma 3.31 (Refinement Infliction within Type 0 Elements).** *Let  $T = [z_0, \dots, z_d]_0$ . Then the refinement infliction of the edge  $E_{ij} = \overline{z_i z_j}$  inside  $T$  is*

$$\mathcal{IR}(E_{ij}, T) = \{E_{lk}, l \leq i, j \leq k\}.$$

Considering NVB-equivalence we know that for type 0 the reflected ordering is the equivalent simplex and the edges within the refinement infliction are symmetric. So this holds also for the simplex  $[z_d, \dots, z_0]_0$ , which has the same refinement tree.

*Proof.* We consider the refinement tree of a type 0 element (Figure 3.1) as in the proof of Lemma 3.4. We observe that there are no connections between nodes on the same level. Nodes that are connected directly to an edge below increase the last index by 1 or decrease the first index by 1. This proves the claim.  $\square$

This allows us to prove the following lemma by exploiting we can refine to a type 0 descendant and use the knowledge of its refinement infliction.

**Lemma 3.32 (No Double Refinement of Edges).** *For any element*

$$T = [z_0, z_1^*, \dots, z_t^*, z_1, \dots, z_{d-t}]_t, t \in \{0, \dots, d-1\}$$

*the inflicted refinement  $\mathcal{IR}(E, T)$  of any edge  $E \in \mathcal{E}(T)$  does not contain an edge that has already been refined and only contains edges that have at least a vertex of  $T$ .*

*Proof.* For type  $t = 0, 1$  this directly follows from Lemmas 3.4, 3.6.

For type  $t \geq 2$  we refine the element  $d - t$  times to reach the descendants  $T'$  (cf. proof of Lemma 3.8) that are of type 0 and are of the form

$$T' = [\bar{z}_0, \bar{z}_1, \dots, \bar{z}_{d-t}, z_1^*, \dots, z_t^*]_0.$$

Here  $\bar{z}_0$  is one of the vertices  $z_i$  and  $\bar{z}_i, i \in \{1, \dots, d-t\}$  is a center of an edge  $\overline{z_k z_j}$  of the initial element  $T$ . While refining to those descendants every edge in  $\text{conv}\{z_0, \dots, z_{d-t}\}$  has been refined once and no other edge of the edges of  $T'$  has been refined.

We now know that there will be no double refinement, because we have a type 0 element. So we have to prove that requesting refinement of the edges that are contained in both  $T$  and  $T'$  does not lead to refinement of any edge that has been refined to get to type 0, i.e., every edge in  $\text{conv}\{z_0, \dots, z_{d-t}\}$ .

So we have to consider the inflicted refinement of the edges of  $\overline{z_i^* z_j}$  and  $\overline{z_i^* z_j^*}$ . We know from Lemma 3.31 that this only contains edges that have at least one vertex  $z_k^*$ . This proves the claim.  $\square$

This leads to the main result of this section which states that for  $d \neq 3$  the weakest compatibility condition is actually sufficient for termination of the iterative refinement algorithm.

**Theorem 3.33 (Sufficiency of Condition 2.26).** *Condition 2.26 is sufficient for Algorithm 2.1 to terminate for NVB for dimension  $d \neq 3$ .*

*Proof.* In the 2d case all elements are of type 0,1 by definition and refining each element uniformly twice leads to all edges being refined once and so the grid is finer and conforming.



For dimension  $d > 3$  we will prove the theorem as follows: We request refinement of any edge  $E \in \mathcal{E}$  of the conforming initial triangulation and prove that the inflicted refinement in direct neighbours may create edges, but does not create edges that lead to double refinement of the initial edge  $E$ . This is done by exploiting that direct neighbours only differ by one in type together with Lemma 3.32.

Then we request refinement of all edges of the grid and prove that this leads to a finer conforming grid, which is sufficient for termination of the algorithm.

Let  $E_{req} = \overline{z_i^* z_j}$  or  $E_{req} = \overline{z_i^* z_j^*}$  be requested for refinement in

$$T = [z_0, z_1^*, \dots, z_t^*, z_1, \dots, z_{d-t}]_t, t \in \{0, \dots, d-1\}$$

and its descendants containing  $E_{req}$  as in the proof of Lemma 3.32 (with  $\bar{z}_0 = z_j$ )

$$T' = [z_j, \bar{z}_1, \dots, \bar{z}_{d-t}, z_1^*, \dots, z_t^*]_0.$$

Also we have a direct neighbour  $T_2$  that contains one different vertex, and carries the same induced refinement tree on the shared face to  $T$ .

Lemma 3.32 shows that requesting refinement of  $E_{req}$  may lead to request edges of the form  $E = \overline{z_i^* \bar{z}}$  for refinement at the direct neighbour  $T_2$ . We need to know whether these requests may lead to double edge refinement within  $T_2$ .

The edge  $E$  is not contained in the current edges  $\mathcal{E}(T_2)$  but in their descendants  $\mathcal{E}_1(T_2)$ , so it first has to be created by refinement before it can be refined. We know from Lemma 3.28 that the types of  $T$  and  $T_2$  differ at most by one and the refinement structure on the shared face  $F$  matches.

As  $d > 3$  the type of the shared face is unique (cf. Remark 3.29), so we can use the fact that vertices that are contained in  $\mathcal{V}_1(T)$  and  $\mathcal{V}(F)$  are also in  $\mathcal{V}_1(F)$ . For all cases except  $t_T = d-1, t_{T_2} = 0$ , this implies that vertices  $z^*$  are in the  $\mathcal{V}_1$  set of  $T_2$  (Remark 2.31). Hence, as in the proof of Lemma

3.32, the requested edge  $E$  only inflicts refinement of the edges containing a vertex  $z^*$  or on edges that were present before.

In the other case we have  $T = [z_0, z_1^*, \dots, z_{d-2}^*, z_1]_{d-1}$  and the requested non-included edge can only be  $E = \overline{z_{01}z_i^*}$  which is not contained in the face of type 0 shared with  $T_2 = [z_0, z_1^*, \dots, z_{d-1}^*]_0$ , so there can be no refinement propagation by this edge.

Requesting all edges  $E \in \mathcal{E}$  for refinement results in refining all  $E \in \mathcal{E}$  and some of the edges that were newly created and contain original vertices  $z \in \mathcal{V}$ . From Lemma 3.32 and the statement above we get that no edge is refined twice. This means there is a finer grid for each triangulation and hence we are done.  $\square$

Our strong intuition is that the weakest condition is also sufficient for  $d = 3$ .

### 3.3 Renumbering Algorithm

The algorithm we design to actually reach a weakly compatible state works in any dimension. It uses a global consistent enumeration of two sets of vertices and exploits the identification of NVB refinement with two locally sorted sets (cf. Remark 2.31). It is designed to be applied directly after grid generation to reach a compatible state of the initial triangulation. The construction of the sets  $\mathcal{V}_0, \mathcal{V}_1$  and the choice of their respective orderings are free parameters that can be adjusted.

Algorithm 3.1 works as follows. We split the vertices of the initial triangulation into two disjoint sets  $\mathcal{V}_0, \mathcal{V}_1$  and sort each of them. Then we use the identification of Remark 2.31 for each element  $T \in \mathcal{T}_0$  and set  $T : \mathcal{V}_0 \cap \mathcal{V}(T), \mathcal{V}_1 \cap \mathcal{V}(T)$  in the respective ordering of  $\mathcal{V}_0, \mathcal{V}_1$ . After proving that this yields a weakly compatible situation for any choice of  $\mathcal{V}_0, \mathcal{V}_1$  and any possible ordering we discuss what may be a good choice for the sets and the orderings.

---

**Algorithm 3.1:** A renumbering algorithm to satisfy Condition 3.12

---

- 1 Let  $\mathcal{V} = \mathcal{T}^0$  be the vertices of the triangulation  $\mathcal{T}$ . Then we divide  $\mathcal{V}$  into disjoint subsets  $\mathcal{V}_i \subset \mathcal{V}, i \in \{0, 1\}, \mathcal{V}_0 \cup \mathcal{V}_1 = \mathcal{V}$  and provide an ordering  $>_i$  for each of them.
  - 2 For all  $T = [z_0, \dots, z_n]_{t_T} \in \mathcal{T}$  we set  $[z_1, \dots, z_{t_T}] = T \cap \mathcal{V}_1$  in order of  $>_1$ ,  $[z_0, z_{t_T+1}, \dots, z_n] = T \cap \mathcal{V}_0$  in order of  $>_0$  and  $t_T = \#(T \cap \mathcal{V}_1) \bmod d$ .
  - 3 If  $T \cap \mathcal{V}_0 = \emptyset$ , we set  $t_T = 0$  and all vertices are sorted ascendingly from  $z_0$  to  $z_n$  with respect to  $>_1$ .
- 

The lemma which states weakest compatibility of the resulting mesh follows directly from Lemmas 3.4, 3.6, 3.8 and the fact that NVB inside an element is strongly compatible.

**Lemma 3.34 (Weakest Compatibility of Resulting Mesh).** *The resulting triangulation of Algorithm 3.1 fulfils the weakest compatibility condition 2.26.*

Even though there are still a lot of free parameters to choose, the meshes generated by Algorithm 3.1 are a strict subset of all possible weakly compatible grid and have additional properties. We can define a level concept for edges and vertices as follows.

**Definition 3.35 (Level Concept for Edges and Vertices).** *The level  $l(z)$  of a vertex  $z$  of the initial grid generated by Algorithm 3.1 is defined by  $l(z) = 0$  for  $z \in \mathcal{V}_0$ ,  $l(z) = 1$  for  $z \in \mathcal{V}_1$ . Newly created vertices always are created on an edge and we set  $l(v) = l(E) + 1$  for  $v$  center of edge  $E = \overline{z_0, z_1}$ .*

*The level of an edge  $E = \overline{z_0, z_1}$  is defined as  $l(E) = \max(l(z_0), l(z_1))$ .*

We now construct the finer mesh  $\mathcal{T}_{fine}$  (compare Lemma 2.39) needed for the termination of Algorithm 2.1.

**Lemma 3.36 (Conforming Descendant).** *Refining all edges of level 0, leads to a conforming grid with all elements being of type 0.*

*Proof.* We refine every simplex individually until it is type 0 without conforming closure. This bisects all edges with both vertices contained in  $\mathcal{V}_0$ ,

i.e., all edges  $E$  with  $l(E) = 0$  and no other edge for every simplex. Hence the resulting triangulation is conforming and only contains elements of type 0.  $\square$

**Remark 3.37.** *Note that the descendants of all faces  $F$  that are quasi-strongly compatible in  $\mathcal{T}$  will be strongly compatible in the constructed descendant of Lemma 3.36. This is because descendants at the face whose types coincide are reflected neighbours and in the constructed descendant all elements are of type 0.*

So from Lemma 3.36 we get the conforming descendant and from Lemma 3.34 we obtain the weakest compatibility condition which are the two ingredients needed to attain weak compatibility. This leads to the following corollary.

**Corollary 3.38.** *The resulting triangulation of Algorithm 3.1 is weakly compatible.*

We briefly investigate the conforming descendant  $\mathcal{T}_{fine}$  where all simplices have type 0. Executing  $d$  uniform refinements on  $\mathcal{T}_{fine}$  leads to every edge in the mesh being bisected exactly once (Lemma 3.4) and a conforming grid, where all elements are of type 0.

**Lemma 3.39.** *Refining all edges of level  $l$  leads to a conforming grid where all edges are of level  $l + 1$ .*

*Proof.* Refining all edges of level 0 leads to a conforming triangulation where all elements are of type 0. All edges in this triangulation are of level 1.

Refining this descendant  $d$  times uniformly coincides with refining all edges of level 1 and we know that the resulting mesh is conforming. Also all edges are now of level 2.

Iterating this argument over the level proves the claim.  $\square$

### 3.3.1 Vertex Sets

Now we address the free parameters in Algorithm 3.1. Important in the course of this discussion is that the vertices in  $\mathcal{V}_1$  are "guarded" (cf. Remark

2.31) from refinement. So edges that have both vertices in  $\mathcal{V}_0$  will be refined before any edge is refined, that has a vertex in  $\mathcal{V}_1$ . We discuss the following variants to choose the sets  $\mathcal{V}_0$  and  $\mathcal{V}_1$  for an initial triangulation  $\mathcal{T}_0$ .

**OT0** (Only Type 0) All elements  $T \in \mathcal{T}_0$  are type 0:

$$\mathcal{V}_0 = \mathcal{V}, \mathcal{V}_1 = \emptyset$$

**ILE** (Initial Longest Edge) Let  $C_{ILE} \in \mathbb{N}_0$  be a constant threshold. Then we define

$$\mathcal{V}_0 = \{v \in \mathcal{V} : v \text{ is in at least } C_{ILE} \text{ longest edges } \}$$

and  $\mathcal{V}_1 = \mathcal{V} \setminus \mathcal{V}_0$ .

**LAE** (Least Adjacent Elements) Let  $C_{LAE} \in \mathbb{N}_0$  be a constant threshold. Then we collect vertices  $v \in \mathcal{V}$  with small environments  $\#\mathcal{T}_v$  by defining

$$\mathcal{V}_1 = \{v \in \mathcal{V} : \#\mathcal{T}_v \leq C_{LAE}\}$$

and  $\mathcal{V}_0 = \mathcal{V} \setminus \mathcal{V}_1$ .

The simplest choice OT0 is the only possibility to attain strong compatibility by definition of strong compatibility on an arbitrary triangulation. If both sets are non-empty, it is highly probable that there are elements, that differ in type, which cannot lead to a strongly compatible situation.

The idea behind ILE is to try mimicking the initial behavior of Longest Edge Bisection, which is known to improve elements that are not quasi-equilateral [RI96, PSP<sup>+</sup>04]. We try to collect the vertices of longest edges into  $\mathcal{V}_0$ , so that all edges that have two vertices in  $\mathcal{V}_0$  are longest edges of most adjacent elements and will be refined first.

LAE is motivated by the idea of refining opposite of large volume angles first, which corresponds to refinement edges on faces that are relatively large.

We collect all vertices that have a small environment, i.e., few adjacent elements and hence a big average volume angle into  $\mathcal{V}_1$ . Then this angle is divided first by bisecting the opposite face. For vertices at the boundary of the grid we divide the constant  $C_{LAE}$  by two, as the neighbourhood is naturally smaller.

These three set choices are in no way a thorough classification of the possible choices but a possible selection of straightforward implementations.

Another possibility would be, for example, to collect all vertices belonging to longest edges that are longer than average into  $\mathcal{V}_0$  or collect the vertices with the maximum volume angle into  $\mathcal{V}_1$ . It is also possible to combine strategies.

### 3.3.2 Vertex Orderings

Algorithm 3.1 has a second degree of freedom, i.e., for each choice of sets we are still free to choose the ordering in each set. As the two sets  $\mathcal{V}_0, \mathcal{V}_1$  are disjoint, we simply construct a global ordering of all vertices  $\mathcal{V}$ , which also sorts each of the sets. We propose two strategies to construct an ordering, the second one being an improved version of the first one. The aim of these orderings is to match neighbouring simplices and to actually reach local strong compatibility on as many faces as possible. We aim to be as strongly compatible as possible, because for strongly compatible grids, we can control the size of the conforming closure [Ste08].

**Definition 3.40 (Successive Reflected Neighbours (SRN) ).** *Let  $O = [ ]$  be an empty ordered set. Choose any element  $T = [z_0, \dots, z_n]_{t_T}$  and set  $O = [z_0, \dots, z_n]$ .*

*For each direct neighbour  $T'$  of  $T$  and the new vertex  $v \in \mathcal{V}(T'), v \notin \mathcal{V}(T)$  we check whether  $v \in O$ . There is an index  $i \in \{0, \dots, n\}$ , such that  $z_i \in \mathcal{V}(T)$  and  $z_i \notin \mathcal{V}(T')$ . If  $v \notin O$  insert  $v$  into  $O$  after  $z_i$ . Then we loop over the direct neighbours of the direct neighbours until each element has been considered.*

**Definition 3.41 (SRN with Announced Edge (SRN2) ).** *We assume that every simplex  $T \in \mathcal{T}_0$  announces an edge  $E_{ann}(T)$ , which it wants to bisect.*

*We execute the SRN strategy with the following addition: If the index  $i = 0$  or  $i = n$  we choose one of two options: Either insert the new vertex  $v$  before the first vertex  $z_0$  or after the last vertex  $z_n$ . If one of the insertions yields  $E_{ann}$  as refinement edge, we choose this one. Otherwise it is not possible to respect the announcement and we simply follow SRN.*

SRN and SRN2 aim at making neighbours (quasi-)strongly compatible. In SRN all direct neighbours of the same type that were used for vertex insertion are reflected. In SRN2 we additionally construct neighbours, who are not reflected, but whose neighbouring children are reflected.

The initial grid  $\mathcal{T}_0$  contains much more faces than vertices. So in most cases in the iteration no vertex insertion takes place, but instead all vertices of the element are already sorted and we cannot guarantee strong compatibility on all faces. Additionally the elements that introduce new vertices are only guaranteed to be strongly compatible to one of their  $d + 1$  direct neighbours. So we can only guarantee a small proportion of faces to be strongly compatible both by SRN and SRN2.

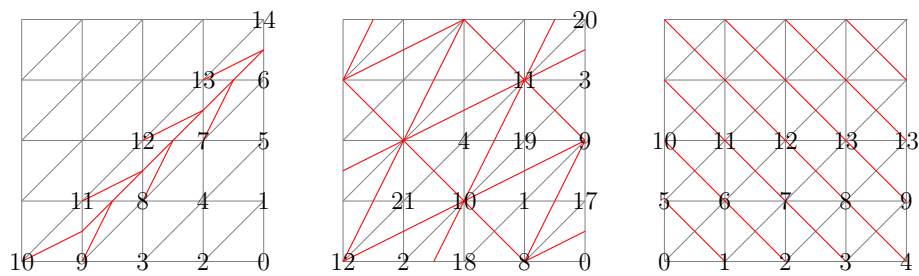
The following example shows that the choice of the ordering directly relates to the size of the conforming closure. Using a structured cube grid filled with Kuhn-cubes, where every simplex announces the diagonal initially, Algorithm 3.1 leads to a strongly compatible grid, if the initial element is sorted correctly.

**Example 3.42 (Different Orderings).** *We choose four different ordering for a 2d structured grid  $\mathcal{T}_0$  of the unit square with  $\mathcal{V}_0 = \mathcal{V}$ ,  $\mathcal{V}_1 = \emptyset$ :*

- *A row-wise ordering (see Figure 3.6c),*
- *SRN,*
- *SRN2 announcing the diagonal edge for each simplex,*

- a diagonally alternating ordering (see Figure 3.6a).

The resulting refinement behaviour is depicted in Figure 3.6. The orderings SRN, SRN2 and the row-wise ordering manage to reach a strongly compatible situation. The diagonally alternating ordering has been hand-crafted to display a worst case, where the refinement propagation of a single refinement in the lower left corner traverses the full grid.



(a) Diagonally alternating ordering leading to an expensive closure  
 (b) SRN ordering leading to strong compatibility  
 (c) Row-wise and SRN2 ordering leading to strong compatibility

Figure 3.6: The conforming closure to different orderings for a 2d Kuhn-triangulation.

A possible implementation of OT0 and SRN2 is shown in Algorithm 3.2.

### 3.4 Numerical Experiments

In this section we study the behaviour of the presented options of the re-ordering algorithm for a variety of different tetrahedral grids. In the implementation first a check is performed whether the provided grids is compatible or not. If the grid is already compatible then no reordering is performed. In general the reordering has to be performed only once on the initial grid. The implementation of the presented algorithm is contained in the open-source DUNE module DUNE-ALUGRID [ADKN16].



---

**Algorithm 3.2:** Possible implementation of methods OT0 and SRN2

---

**Data:** List of active faces  $\mathcal{F}$ , List of vertices  $\mathcal{V}$ , Mesh  $\mathcal{T}$

- 1  $\mathcal{F} = \mathcal{V} = \emptyset$
- 2 Choose initial  $T \in \mathcal{T}$
- 3 Order according to announced refinement edge  $E(T)$
- 4  $\mathcal{V} = \mathcal{V}(T)$
- 5 add all non-boundary  $F \in \mathcal{T}^{d-1}(T)$  with  $E(T) \in F$  at the beginning of  $\mathcal{F}$
- 6 add the other non-boundary  $F \in \mathcal{T}^{d-1}(T)$  at the end of  $\mathcal{F}$  with the flag noRefEdge
- 7 mark  $T$  as treated
- 8 **while**  $\exists$  untreated  $T \in \mathcal{T}$  ( $\Leftrightarrow \mathcal{F} \neq \emptyset$ ) **do**
- 9     Get untreated neighbour  $T'$  and treated element  $T$  of first  $F \in \mathcal{F}$
- 10      $v = \mathcal{T}^0(T) \setminus \mathcal{T}^0(F)$      $v' = \mathcal{T}^0(T') \setminus \mathcal{T}^0(F)$
- 11     **if**  $v' \in \mathcal{V}$  **then**
- 12         | */\* do nothing* *\*/*
- 13     **else**
- 14         | **if** noRefEdge set **then**
- 15             | */\* In this case  $v$  is the first or last vertex of*
- 16             |  $T$  *\*/*
- 17             | insert  $v'$  after last or before first vertex of  $T$ , depending on
- 18             | the announced refinement edge
- 19         | **else**
- 20             | Insert  $v'$  directly after  $v$  into  $\mathcal{V}$
- 21     sort  $T'$  according to  $\mathcal{V}$
- 22     mark  $T'$  treated
- 23     **for** non-boundary  $F' \in \mathcal{T}^{d-1}(T)$  **do**
- 24         | **if**  $F' \in \mathcal{F}$  **then**
- 25             | */\* possibly check for strong compatibility* *\*/*
- 26             | remove  $F'$  from  $\mathcal{F}$
- 27         | **else**
- 28             | **if**  $E(T') \subset F'$  **then**
- 29                 | add  $F'$  at beginning of  $\mathcal{F}$
- 30             | **else**
- 31                 | add  $F'$  at the end of  $\mathcal{F}$

---

### 3.4.1 Mesh Quality

We are interested in two significant mesh quality goals, namely, the *mesh topological quality* goal that tries to minimize the effort of the conforming closure, and the *geometric quality* goal that tries to maximize the shape regularity of the elements.

Strongly compatible grids fulfil the *mesh topological quality* goal by design. So a distance to a strongly compatible situation is a way of measuring this goal. To this aim we define the following two distances.

**Definition 3.43 (Distance 1).** *Let  $\mathcal{T}$  be a weakly compatible grid sorted by one of the above methods and let  $\mathcal{F}_{SC} \subset \mathcal{T}^{d-1}$  be the set of faces that are strongly or quasi-strongly compatible. Then we define the distance*

$$d_{\mathcal{T}}^1 = \#(\mathcal{T}^{d-1}) - \#(\mathcal{F}_{SC}).$$

This is a simple, easily computable and straightforward metric that directly tells us whether the grid fulfils the strong compatibility condition. Unfortunately, it does not directly yield information on the effort of the conforming closure, so to estimate this we define a second distance:

**Definition 3.44 (Distance 2).** *Let  $\mathcal{T}$  be a weakly compatible grid sorted by one of the above variants and let  $\mathcal{C}_{\mathcal{T}}(T)$  the conforming closure of refining  $T$  on the initial grid  $\mathcal{T}$ . Then we define the distance*

$$d_{\mathcal{T}}^2 = \max_{T \in \mathcal{T}} \#(\mathcal{C}_{\mathcal{T}}(T)).$$

Refinement propagation becomes more local (compare Chapter 4) once the whole grid has been refined  $d$  times uniformly. So as a third distance in the numerical experiments we also investigate  $d_{\mathcal{T}_d}^2$  on the  $d$  times uniformly refined grid  $\mathcal{T}_d$ .

The geometric quality of meshes using bisection (both LEB and NVB) is always an issue. One of the angles may be divided  $2^{d-1}$  times under  $d$  uniform bisections. LEB performs a bit better by design, but there is a significant

drop in quality for any bisection-based refinement, if the initial grid contains elements that are close to equilateral, which we expect from mesh-generators such as `TetGen`.

In principle it is advisable to start with elements that are similar to (non-equilateral) Kuhn-simplices. For NVB we know that  $d$  uniform bisections cover all possible similarity classes and hence will only measure our quality indicators on the initial grid and on the  $d$  times uniformly refined grid.

The set of indicators we use are the  $d$ -dimensional sine, aspect ratios of volumes, faces and edges (min, max and average) and also the maximum number of adjacent elements of vertices and edges [Eri78].

### 3.4.2 Threshold Study

The first set of experiments we conduct is a threshold study, where we measure the behaviour of LAE and ILE with threshold values  $C_{ILE}, C_{LAE} \in \{0, \dots, 35\}$ . If the threshold value is 0 we obtain in both cases  $\mathcal{V}_0 = \mathcal{V}$ , so the case OT0 is implicitly included. We use both suggested orderings SRN and SRN2. As all grids in this experiments are 3d, we consider the quality indicators of the 3 times uniformly refined grid. If present, a threshold value of  $-1$  indicates the unrefined grid.

First we test on a sequence of triangulations of the unit cube with decreasing average volume, that have been used in the *3D Benchmark on Discretization Schemes for Anisotropic Diffusion Problems on General Grids* [EHH<sup>+</sup>11]. We will call these grids the "Unitcube Sequence". They are enumerated from coarsest to finest from one to six.

Additionally we will show the same measures for a set of grids representing more complex geometrical structures. We will call these grids "Realistic Grids" and a depiction is presented in Figure 3.25 at the end of the chapter. The tube grid has been previously used for numerical simulation of atherosclerotic plaque formation [GKO14]. The grid representing the head of a human has been used in [JDB<sup>+</sup>15].

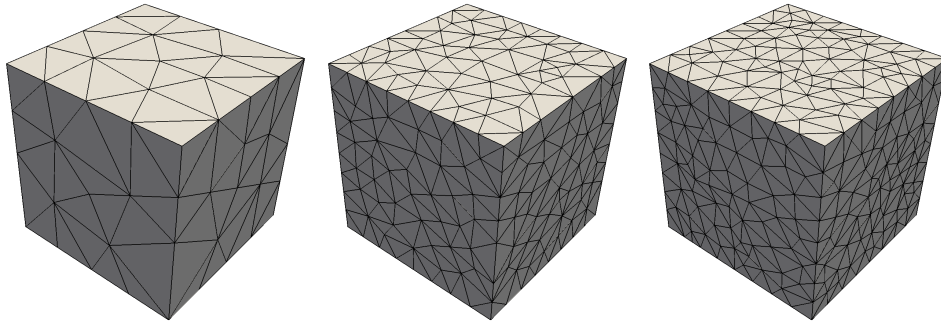


Figure 3.7: Series of tetrahedral grids discretizing the unit cube used in the *3D Benchmark on Discretization Schemes for Anisotropic Diffusion Problems on General Grids* [EHH<sup>+</sup>11].

First we consider the distribution of  $\mathcal{V}_0, \mathcal{V}_1$  by the different strategies SRN and SRN2.

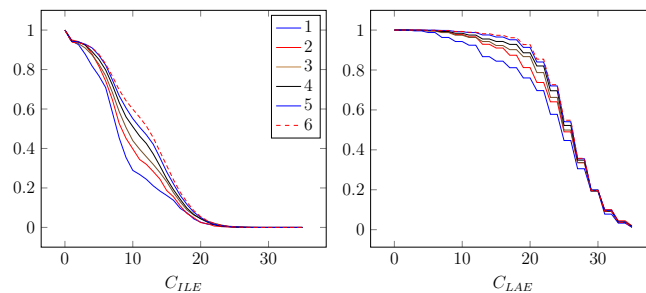


Figure 3.8: Proportion of vertices sorted into  $\mathcal{V}_0$  over different thresholds for the unitcube sequence.

Figures 3.8 and 3.9 show the proportion of vertices that have been sorted into  $\mathcal{V}_0$  for different grids. As expected LAE and ILE have different “active” regions, i.e. regions, where a change of threshold changes the distribution a lot. For ILE this is  $C_{ILE} \in \{1, \dots, 20\}$  and for LAE this is  $C_{LAE} \in \{10, \dots, 30\}$ . These active regions are of interest in all following figures. The distribution of  $\mathcal{V}$  into  $\mathcal{V}_0$  and  $\mathcal{V}_1$  does not depend on the choice of ordering. So the active regions do not change from SRN to SRN2. Once the parameter becomes fully inactive all vertices have either been sorted into  $\mathcal{V}_0$  or into  $\mathcal{V}_1$  completely, so that the behaviour of these cases coincides with choosing the

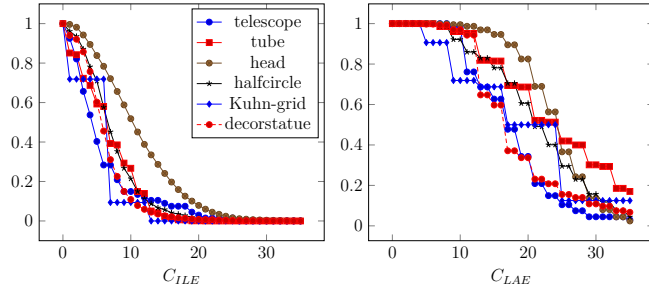


Figure 3.9: Proportion of vertices sorted into  $\mathcal{V}_0$  over different thresholds for the realistic grids.

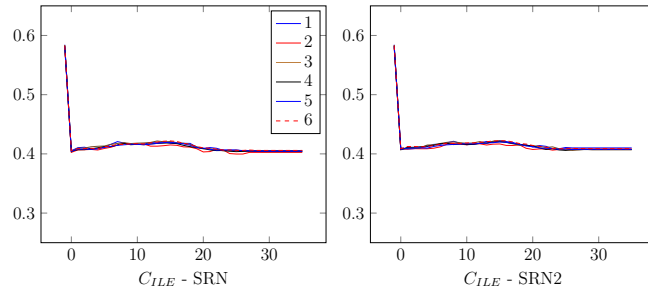
strategy OT0, which is why it is not depicted explicitly in the figures.

Using the abbreviations Volume (V), Longest Edge (LE), Shortest Edge (SE), Largest Face (LF), Smallest Face (SF), Face Volume (F), we examine the following geometric metrics of the grid:  $V/LE^3$ ,  $V/SE^3$ ,  $V/LF^{3/2}$ ,  $V/SF^{3/2}$ ,  $F/LE^2$ ,  $F/SE^2$  and the d-sine [Eri78].

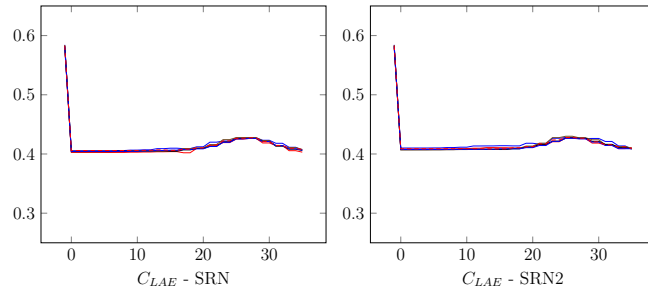
We observed that  $(V/LE^3) \approx (V/LF^{2/3}) \times (F/LE^2)$  and additionally  $(V/LF) \approx$  d-sine. Qualitatively the metrics behave similarly, so we choose to display only the d-sine here.

To be able to compare to the initial grid  $\mathcal{T}_0$  the first value in these graphs with threshold value  $C_{ILE/LAE} = -1$  denotes the unrefined grid  $\mathcal{T}_0$ , which is the same for all thresholds. The study then is executed on the grid  $\mathcal{T}_3$ , so that each of the similarity classes is achieved.

Figures 3.10 and 3.11 display the average d-sine for the investigated grids. For almost all grids we see the expected significant drop of geometric quality from the initial grid to the 3 times uniformly refined mesh and then some improvement in the active region. The only exception is the Kuhn-grid, which actually may get worse in the active region (cf. Figure 3.11, combination ILE/SRN2), where it is not strongly compatible anymore. Also there is no initial drop. For other grids LAE seems to perform a bit better than ILE, and SRN versus SRN2 does not seem to have much impact.

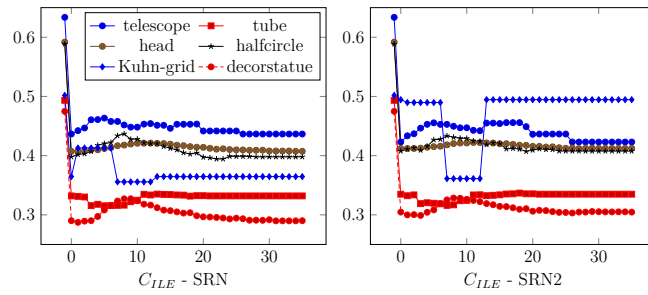


(a) Initial Longest Edge

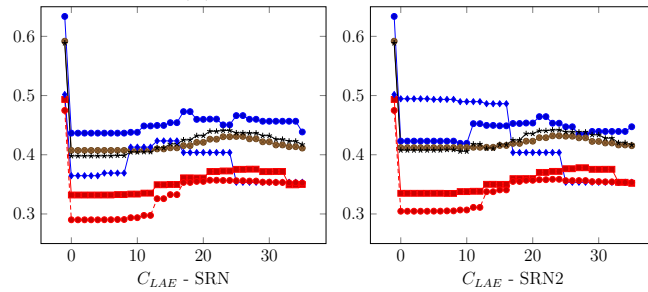


(b) Least Adjacent Elements

Figure 3.10: Average d-sine after uniform refinement for the unitcube sequence.



(a) Initial Longest Edge



(b) Least Adjacent Elements

Figure 3.11: Average d-sine after uniform refinement for the realistic grids.

A way to investigate the minimum angle is to consider the maximum number of adjacent elements at a vertex. Figures 3.12, 3.13 and 3.14 indicate that there is a big difference between LAE and ILE. In the active region of ILE this gets worse, while in the active region of LAE this value improves.

To get an impression of the size of this value note the following calculation. Equilateral tetrahedrons can fill a 3d space with ikosahedrons, i.e., at every vertex there are 20 adjacent elements. Uniform bisection in 3d can quadruple the number of elements at a vertex, which yields an expected 80 elements as a maximum after 3 uniform refinements for equilateral initial grids. The best values of LAE are not far from that.

The special grid originating from Kuhn-cubes (see Figure 3.25d) performs best under bisection with respect to both measures. The second special case among the realistic grids is the decorstatue grid (cf. Figure 3.25c). It has by far the most elements at a vertex in the initial triangulation and we see that ILE is not improving that, while LAE almost recovers the initial value at a high threshold.

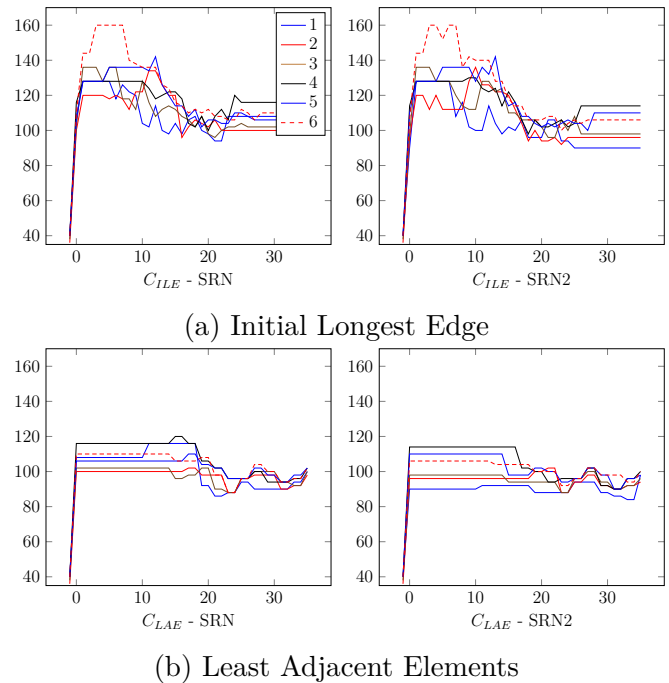


Figure 3.12: Maximum number of elements at a vertex for unitcube sequence

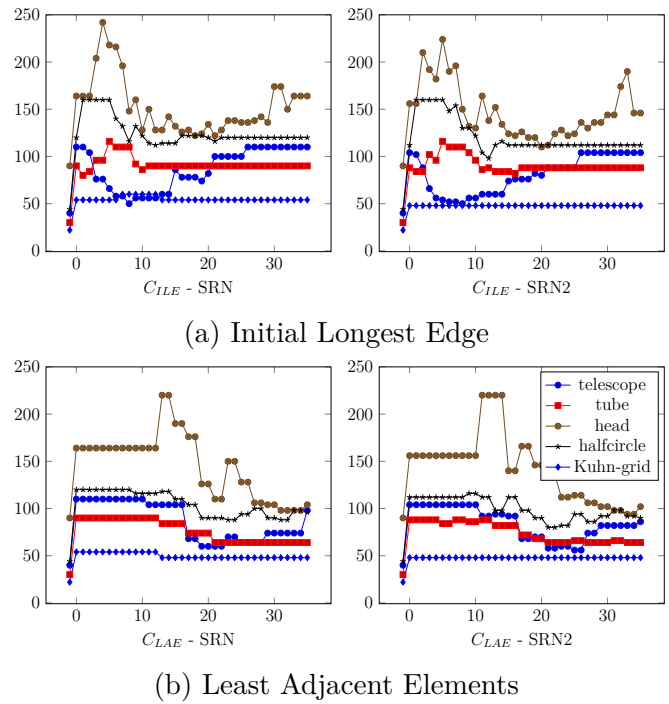


Figure 3.13: Maximum number of elements at a vertex without decorstatue

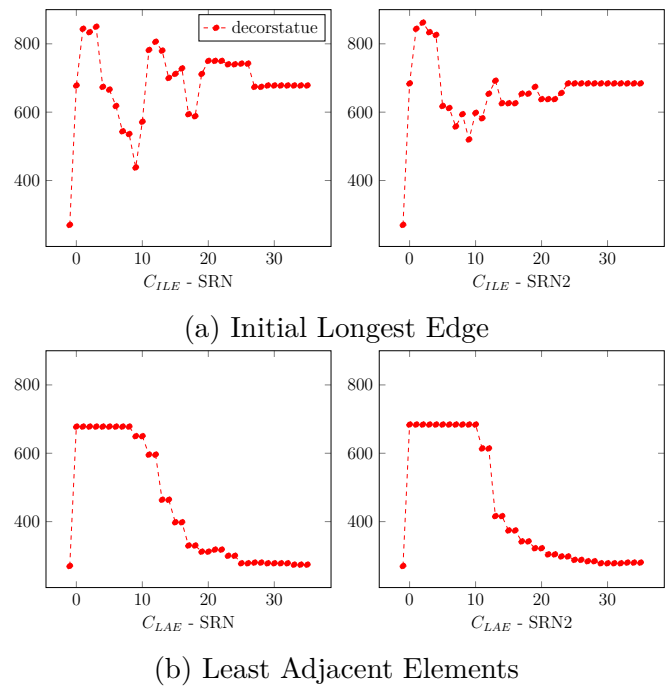


Figure 3.14: Maximum Number of Elements at a vertex for decorstatue



### 3.4.3 Conforming Closure

We discuss the connection between the distances  $d_{\mathcal{T}}^1$ ,  $d_{\mathcal{T}}^2$  and  $d_{\mathcal{T}_3}^2$ . Unfortunately the computation cost of  $d_{\mathcal{T}}^2$  is at least  $O(n^2)$  and  $d_{\mathcal{T}_3}^2$  is at least  $O((8n)^2)$ . Hence we did not compute these measures for all grids and possible parameters but just for a small selection. Also in the implementation we make the simplification, if refinement of an element  $T$  is part of the conforming closure of another element  $T'$  then the conforming closure of  $T$  is included in the conforming closure of  $T'$  and we do not compute the closure of  $T$ . This means, the maximum conforming closure  $(d^2\mathcal{T}, d^2\mathcal{T}_3)$  is computed exactly, but we just get an upper bound for the average conforming closure.

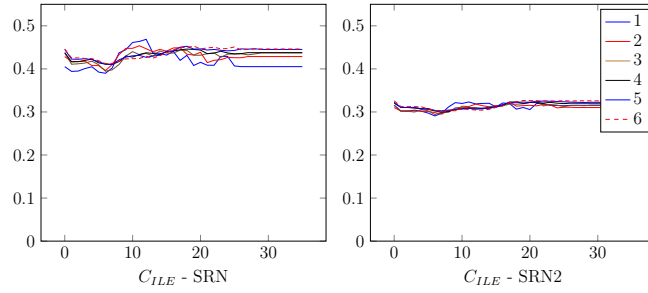
We now investigate distance  $d_{\mathcal{T}}^1$ , i.e., the amount of not strongly compatible faces. To be comparable between grids of different size, we display the share with respect to all inner faces. Boundary faces are excluded, as they do not need to be compatible.

Figures 3.15 and 3.16 display the amount of not strongly compatible faces in the grid. There are a few things to note. In most cases SRN2 outperforms SRN by about 10% of the faces. The Kuhn-grid (Figure 3.16, always bottom) is able to recover the strong compatibility in the inactive region (i.e., OT0) combined with SRN.

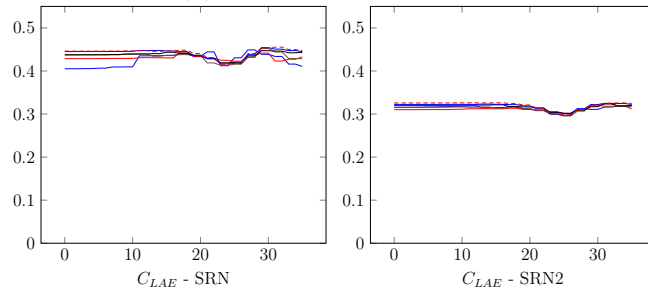
The size of the conforming closure is investigated from Figure 3.17 on, which displays an upper bound for the average conforming closure of each element in the coarsest grid of the unitcube sequence. Figure 3.18 displays the same for the telescope grid and Figure 3.19 for the special Kuhn-grid, which becomes strongly compatible. Figures 3.20, 3.21 and 3.22 display  $d_{\mathcal{T}}^2$  and  $d_{\mathcal{T}_3}^2$ , i.e., the maximum conforming closure for the three grids.

Both for the average and the maximum conforming closure we measure on the initial grid (blue) and on the 3 times uniformly refined grid (red). The closure is almost always smaller in the 3 times uniformly refined case. The conforming closure is more costly in the active region, as we additionally introduce refinement propagation by having neighbours of different type. In

the case of SRN we see for the unitcube mesh that the maximum conforming closure actually diminishes in the active region, but SRN2 performs better in total. The Kuhn-grid performs best in the inactive region, where it actually becomes strongly compatible.

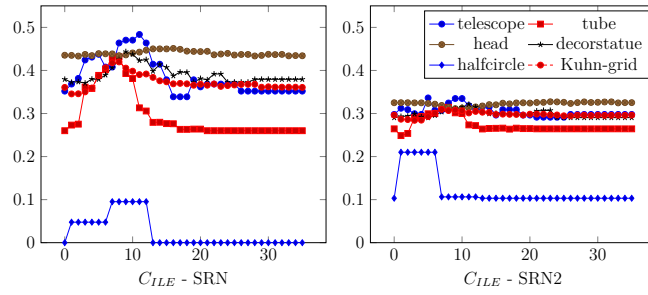


(a) Initial Longest Edge

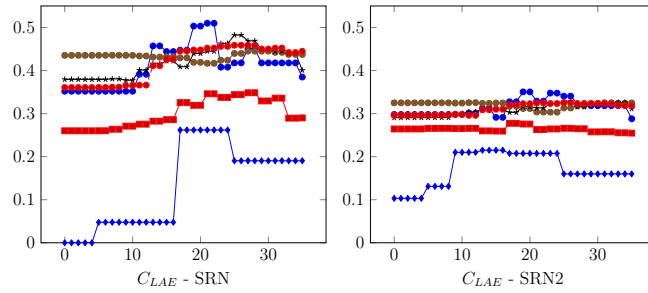


(b) Least Adjacent Elements

Figure 3.15: Share of not strongly compatible faces, unitcube sequence

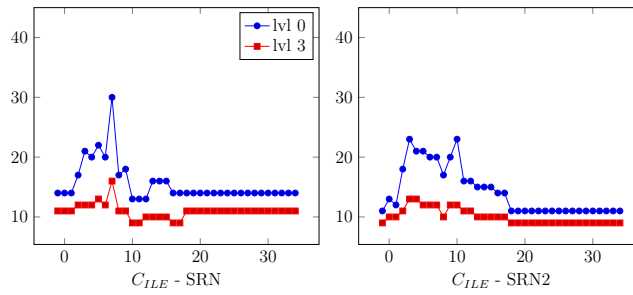


(a) Initial Longest Edge

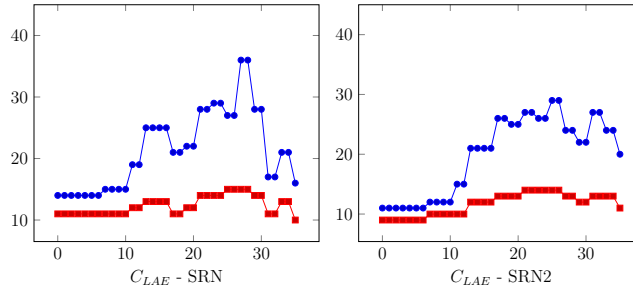


(b) Least Adjacent elements

Figure 3.16: Proportion of not strongly compatible faces, realistic grids.

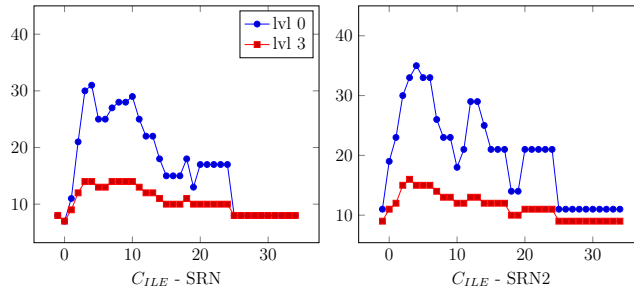


(a) Initial Longest Edge

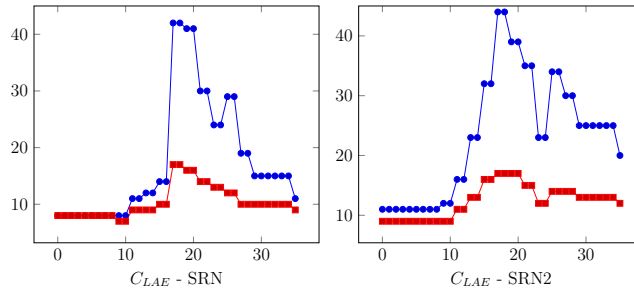


(b) Least Adjacent Elements

Figure 3.17: Average conforming closure for first grid of unitcube sequence

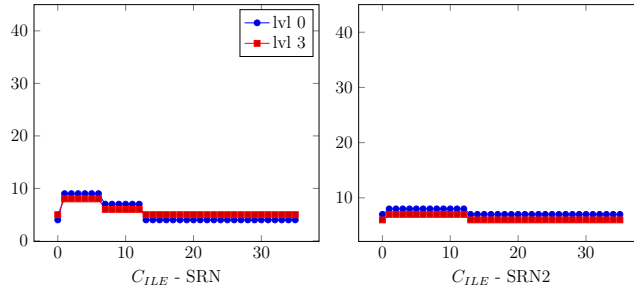


(a) Initial Longest Edge

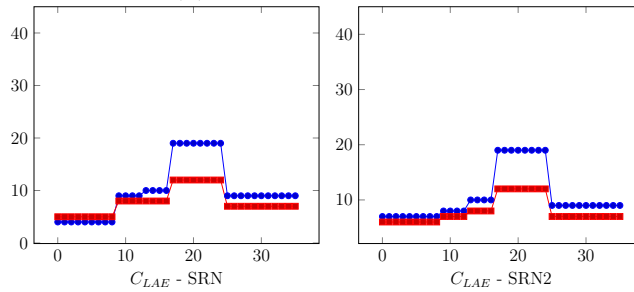


(b) Least Adjacent Elements

Figure 3.18: Average conforming closure for Telescope grid (see Fig. 3.25e)

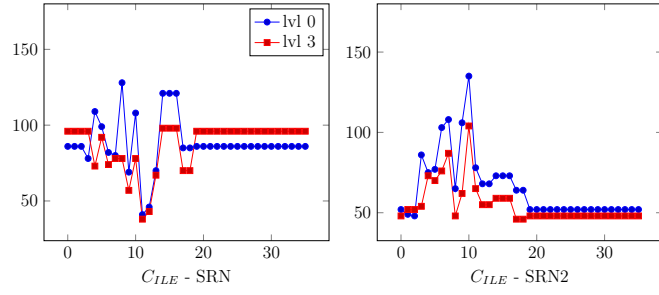


(a) Initial Longest Edge

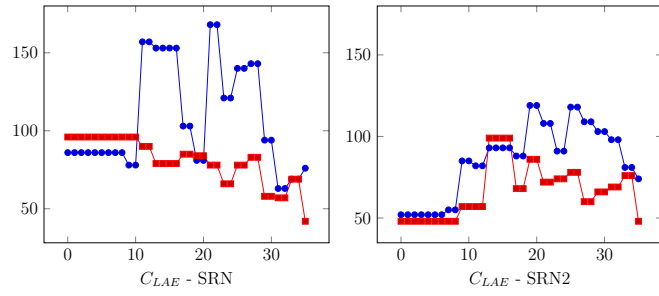


(b) Least Adjacent Elements

Figure 3.19: Average conforming closure for Kuhn-grid

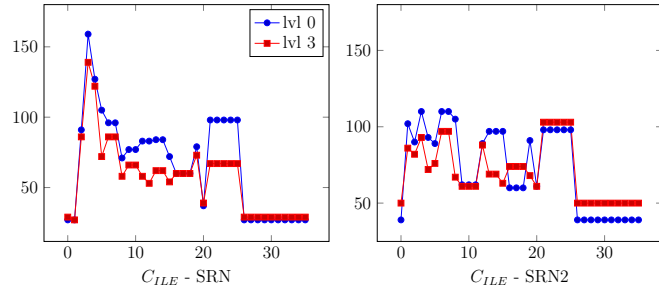


(a) Initial Longest Edge

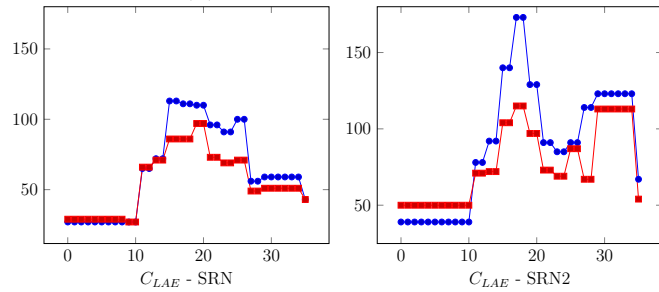


(b) Least Adjacent Elements

Figure 3.20:  $d_{\mathcal{T}}^2$  and  $d_{\mathcal{T}_3}^2$  for first grid of Unitcube Sequence



(a) Initial Longest Edge



(b) Least Adjacent Elements

Figure 3.21:  $d_{\mathcal{T}}^2$  and  $d_{\mathcal{T}_3}^2$  for Telescope grid

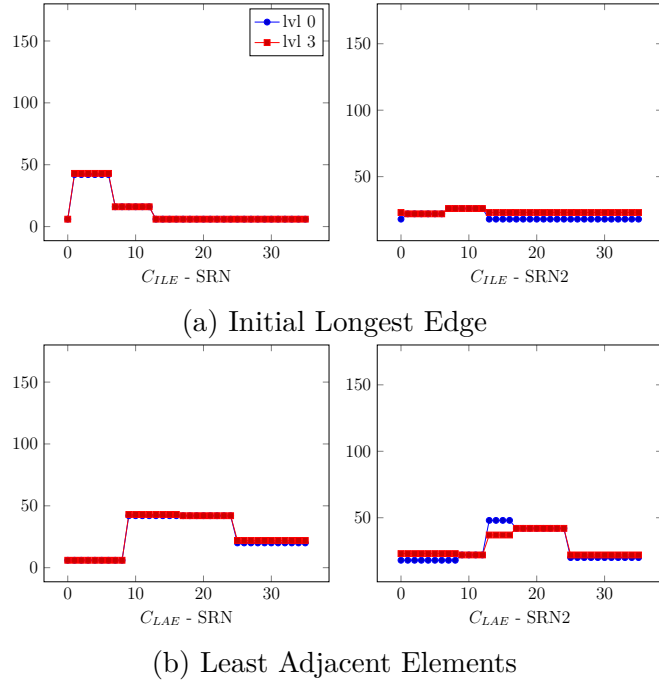


Figure 3.22:  $d_{\mathcal{T}}^2$  and  $d_{\mathcal{T}_3}^2$  for Kuhn-grid

### 3.4.4 Summary and Recommendations

The special behaviour of the grid constructed using Kuhn-cubes shows that when knowing that the current grid is based on Kuhn-cubes, one should choose a method that recovers the strong compatibility, as it outperforms the other options. Kuhn-cubes seem to be a good start for adaptive NVB meshes as they do not drop in geometric quality over refinement.

When given an initial mesh by a standard mesh generator, that has not been triangulated using Kuhn-cubes, we suggest to use SRN2 as it outperforms SRN in size of the conforming closure and share of strongly compatible faces. For ILE we suggest a value of  $C_{ILE} \approx 10$  and for LAE a value of  $C_{LAE} \approx 20$ , as these perform best geometrically. The question whether to use ILE or LAE is a matter of personal taste. ILE performs a bit better when considering the size of the conforming closure, while LAE performs a lot better, when considering the maximum number of elements at a vertex, or the smallest average volume angle at a vertex in particular for the extreme

case of the decorstatue.

### 3.4.5 Algorithm Complexity

We study the runtime of the algorithm over the sequence of unit cube meshes that we used to study the threshold values. Figure 3.23 shows that the runtime is  $O(n \log n)$  with  $n$  being the number of elements.

The measured runtime includes recalculating the grid neighbourhood information, checking whether a grid is compatible initially, setting up  $\mathcal{V}_0, \mathcal{V}_1$ , sorting the mesh, and finally checking how compatible the mesh is. The algorithm can be implemented in  $O(n)$ , if the recalculation of the neighbourhood information is not necessary. In the current implementation the association of neighbouring elements is done via comparison of vertex indices of faces and storage of those in a `std::map` with  $O(\log n)$  member access, where  $n$  is the number of faces. Therefore, the overall algorithm complexity becomes  $O(n \log n)$ . In our tests we have applied the algorithms to meshes with about one million grid cells in under 10 seconds sequentially which we consider sufficiently fast, especially since the algorithm has to be applied only once.

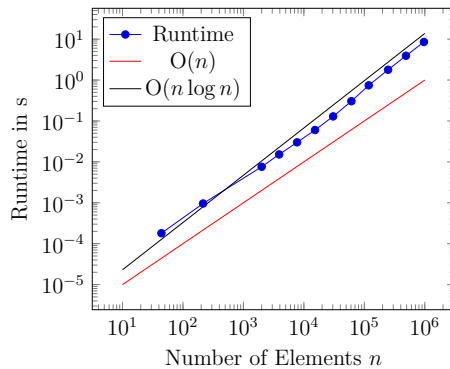


Figure 3.23: Runtime of the algorithm for different grid sizes.

### 3.4.6 Arbitrary Dimensional Grid

As proof of concept we implemented a simple  $d$ -dimensional mesh that can handle bisection. Some implementation details are covered in Section 5.3.

**d-dimensional Experiment** As a numerical example we refine a Kuhn-Cube  $d$  times uniformly and afterwards mark one of the elements for  $d$  further refinements. Then refinement with conforming closure is executed and we display the size of the resulting mesh.

We execute this experiment twice, first with a row-wise enumeration of the vertices leading to strong compatibility. The second time we also use the dimension-wise ordering, but shifted by a permutation depending on the number 7

$$i \rightarrow 7i \bmod d!,$$

which leads to a weakly compatible situation. In both cases we use OT0 as a method, so all vertices belong to  $\mathcal{V}_0$ .

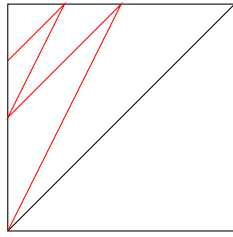
Dimension	2	3	4	5	6	7	8
Initial	2	6	24	120	720	5040	40320
Intermediate	8	48	384	3840	46080	645120	10321920
Final (Strong)	16	109	951	12968	191452	3467190	OOM
Final (Weak)	15	108	1004	11400	152760	2358720	OOM

Table 3.1: Number of Elements of Kuhn-Cube refinement, OOM denotes Out-Of-Memory

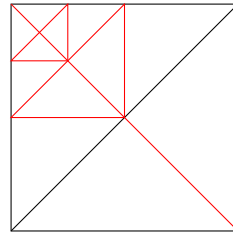
Table 3.1 shows the sizes through the experiment. As expected the initial size is always  $d!$  and the intermediate size after  $d$  uniform refinements is  $2^d d!$  for both the weakly and the strongly compatible grid. Observe that the adaptively refined grid is a lot smaller than  $d$  additional uniform refinements. The 8 GB of RAM of the laptop used to calculate this example was too small to fully execute the example for dimension 8 (OOM = Out-Of-Memory).

In a Kuhn-cube even after  $d$  uniform refinements all elements contain a corner vertex of the original cube (cf. Figure 2.1). The weak setting contains less elements after refinement, because it refines into a corner without needing an expensive conforming closure. See Figure 3.24 for an illustration in 2 dimensions. We do, however, expect the strongly compatible grid to yield better element shapes, as observed in the previous study.





(a) Weakly Compatible



(b) Strongly Compatible

Figure 3.24: Refinement of a Kuhn-cube in 2 dimensions

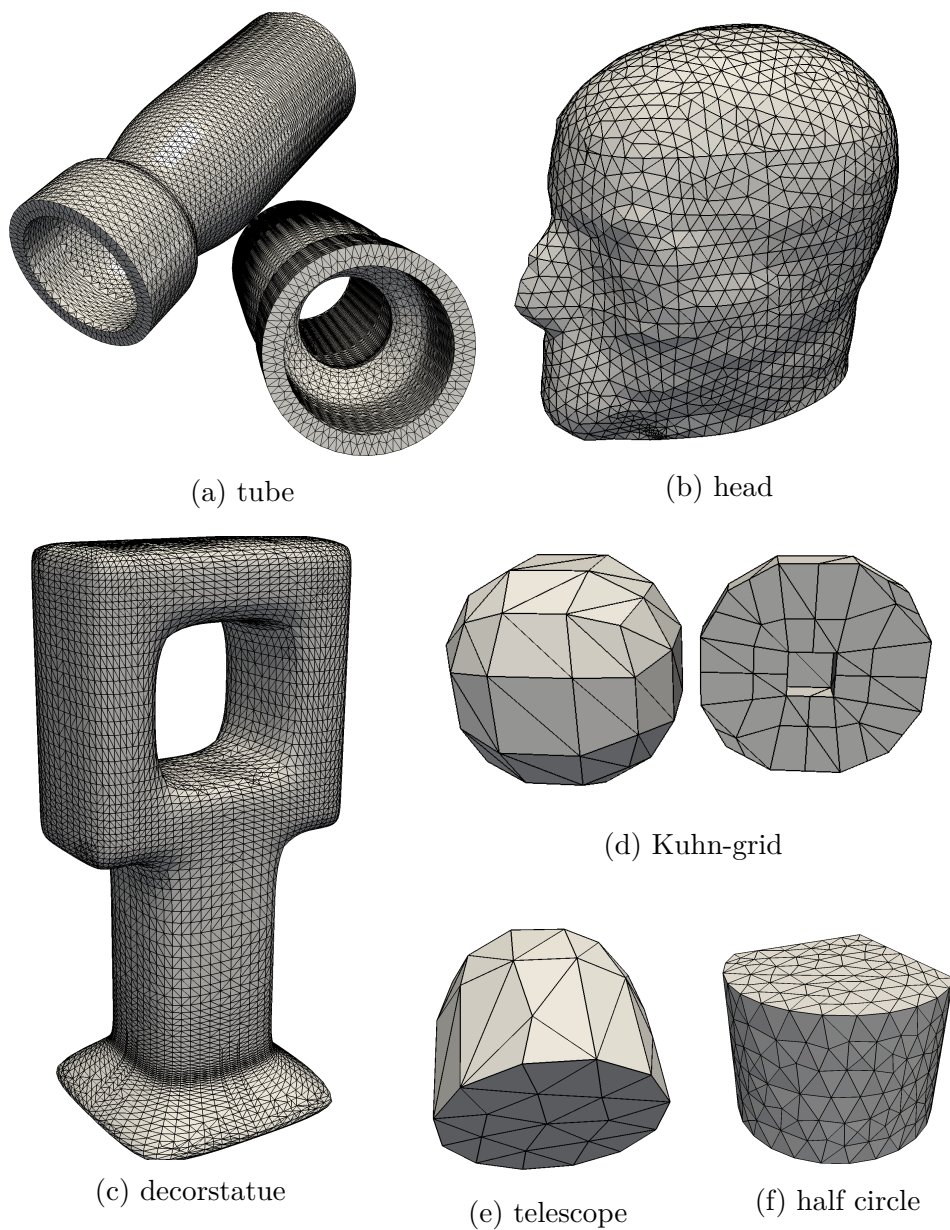


Figure 3.25: Realistic tetrahedral meshes used for studying the proposed algorithms. 3.25a has been previously used for numerical simulation of atherosclerotic plaque formation [GKO14], 3.25b has been used in [JDB<sup>+</sup>15] for electrical impedance tomography simulations, 3.25c has been downloaded from [GAM16] and generated with TETGEN, 3.25e is part of the test grids in the DUNE-GRID module [BBD<sup>+</sup>16], 3.25d stems from a Kuhn-grid and has been modified (vertex renumbering and projection), and 3.25f has been generated with GMSH.

# Chapter 4

## Parallel Refinement

We extend NVB to the spatial domain decomposition case. Section 4.1 briefly introduces the parallel version of NVB refinement, which consists of an outer loop, that requires communication and an inner loop, that executes NVB on each partition. Section 4.2 estimates the number of outer iterations needed and relates it to the amount of global communication needed. The results within this section depend on the strong compatibility condition.

While the first result (Theorem 4.5) holds without additional assumptions, The stronger results (Theorems 4.9 and 4.14) additionally assume that the grid is distributed on a certain element level. With this assumption we can show that the parallel overhead of reaching a conforming state with NVB is bounded, in particular by a constant independent of the number of processors (Theorem 4.9). Section 4.3 extends parts of the results to weakly compatible grids. Theorem 4.18 shows that refinement cannot propagate more than a layer of macro elements around the macro element that the refinement originates from, once the mesh has been refined to a certain level.

Finally, in Section 4.4 we execute experiments to verify the theoretical results and show proper scaling of the algorithm. This chapter constitutes a substantial extension of the previously published work [AK17] in the *Journal of Parallel and Distributed Computing* in 2017.

We make the common assumption that load balancing is done after the

refinement algorithm is finished. Hence we do not consider its effect on the computational cost of the refinement algorithm. However, we will show that it is possible to achieve decent strong scaling on a super computer using our distributed NVB implementation.

Unless stated otherwise throughout this chapter we assume strong compatibility (Condition 3.22). This work was done before the introduction of the weak compatibility condition in our work [AGK18] and its extension is non-trivial. A recent first extension is achieved by Theorem 4.18 and its Corollary 4.20.

## 4.1 Distributed Newest Vertex Bisection

We rely on a grid decomposition approach. This means that we split the initial grid  $\mathcal{T}_0$  into  $P$  partitions for  $P$  processors. Each processor gets exactly one partition and knows the owners of neighbouring partitions. This splitting process is called initial *partitioning*. It often aims at equilibrating the number of elements on each core while minimizing the number of elements at partition boundaries. This is because the number of elements correlates to the workload of the processor, while the amount of faces at the partition boundary correlates to the necessary communication. This goal-oriented form of partitioning is called *loadbalance*.

Loadbalancing shifts elements of the current triangulation  $\mathcal{T}$  from one partition to another. For non-hierarchical grids there are no restrictions on which elements may be shifted. In hierarchical grids, which allow coarsening, the partitions need to know the common ancestors of coarsened elements which may lie across a partition boundary. This is why our implementation in DUNE-ALUGRID only allows for all elements in the tree of a single macro element  $T_0 \in \mathcal{T}_0$  to be moved between partitions. Other solutions to this problem include storing the full macro grid on each partition, communicating the refinement tree of elements split at the partition boundary and always rebalance on coarsening. This restriction can also be avoided by simply disallowing coarsening of elements at partition boundaries.

To close the distributed grid conformingly across partition boundaries during refinement we need to alter the serial refinement algorithm. The basic idea is to execute the serial refinement algorithm on each partition, communicate the refinement status of faces on the partition boundary to the corresponding neighbouring partition, close the neighbouring partition conformingly and iterate.

The parallel refinement algorithm (Algorithm 4.1) expects input on each partition belonging to a processor  $i$  in form of a set  $\mathcal{M}_i$  of elements marked for refinement.

---

**Algorithm 4.1:** Distributed Newest Vertex Bisection

---

- 1 Initialize set of elements marked for refinement on each partition  $\mathcal{M}_i, 0 \leq i < P$ .
  - 2 **while**  $\mathcal{M}_i \neq \emptyset \forall i$  **do in parallel**
  - 3     Refine partition  $P_i$  using NVB until  $\mathcal{M}_i$  is empty
  - 4     Communicate refinement status of partition boundary to corresponding neighbour partition
  - 5     Add nonconforming simplices to  $\mathcal{M}_i$
  - 6     Communicate globally whether  $\mathcal{M}_i$  is empty
- 

Algorithm 4.1 is an improved version of the algorithm introduced in [Zha09] by additionally communicating the refinement status of edges belonging to the partition boundaries. This is slightly more communication expensive in 3 or more dimensions but it reduces the number of outer iterations needed. For 2 dimensions both algorithms coincide as faces are edges in 2d.

The stopping criterion of the while loop requires a global communication (Allreduce,  $O(\log P)$  where  $P$  is the number of partitions), which cannot be expected to scale well onto many cores. On the other hand, communicating the refinement status to the neighbouring partition can be expected to scale quite well as long as the number of neighbouring partitions stays small.

After the refinement has finished it may be that several partitions contain too many elements in comparison to others. In this case we carry out another

loadbalancing step.

## 4.2 Communication Bounds

Bounds for the amount of communication to reach a conforming triangulation are directly related to bounding the number of iterations in Algorithm 4.1. While the first bound from Theorem 4.5 below does not assume more than strong compatibility (Condition 3.22), Theorems 4.9 and 4.14 additionally require a certain form of mesh decomposition.

### 4.2.1 Generic Analysis

The following first lemma helps to understand the direct consequences of a single refinement. It is not to be confused with Lemma 3.28, as Lemma 4.1 treats any kind of neighbours, while Lemma 3.28 considers direct neighbours.

**Lemma 4.1.** *For all neighbours  $T'$  of an element  $T \in \mathcal{T}$  with refinement edge  $E$  with  $E \subset T' \cap T$  one of the following two statements holds:*

- 1 *Refinement of  $E$  in  $T'$  induces no further refinement (it already is the refinement edge).*
- 2  *$E$  is the refinement edge of a child of  $T'$  and refinement of  $E$  induces up to  $d - 1$  refinements of elements  $T_i \subset T'$  with  $\text{gen}(T_i) < \text{gen}(T)$ .*

*Proof.* There are two cases. If the generations coincide  $\text{gen}(T) = \text{gen}(T')$  then there is no further refinement.  $E$  is the refinement edge of both elements (Lemma 3.24).

If the generations differ by  $i$  (i.e.,  $\text{gen}(T) = \text{gen}(T') + i$  with  $1 \leq i \leq d - 1$ ) refinement of  $T$  induces  $i$  further refinements. The refinement edge can only be the same, if both elements have the same generation and neighbours can only differ in generation by less than  $d$  (Lemma 3.26).

The generation is increased by one with each refinement, so there have to be  $i$  refinements of descendants of  $T'$  that are refined, until the  $i$ -th descendant shares the refinement edge with  $T$ . □

Lemma 4.1 can be applied recursively. A single refinement may lead to refinements of neighbours at 1 to  $(d - 1)$  generations older and these may again lead to refinements at even lower generations. All these refinement edges appear while constructing the conforming closure. We aim to collect all these edges into a graph. Therefore we consider the effect of refinement of a single edge onto one adjacent neighbour in the following example.

**Example 4.2** (Direct Refinement Propagation). *Let us assume a simplex  $T$  with generation  $\text{gen}(T) = 0$  and its neighbour  $T'$  with  $\text{gen}(T') = d - 1$  and refinement edge  $E = T \cap T'$  of  $T'$ . Now we refine  $T'$ , so we refine  $E$  and hence  $T$ . Condition 3.22 yields that  $E$  is the refinement edge of a descendant  $T''$  of  $T$  with  $\text{gen}(T'') = d - 1$ . We denote by  $T_E^i$  the descendant of  $T$  with generation  $i$  that contains  $E$  and we denote its refinement edge by  $E_i$ . So  $T'' = T_E^{d-1}$ ,  $T = T_E^0$  and  $E_{d-1} = E$ . Then the refinement propagation can be depicted in the following graph of Figure 4.1.*

*The dashed line denotes the connection to the original edge  $E$ . This does*

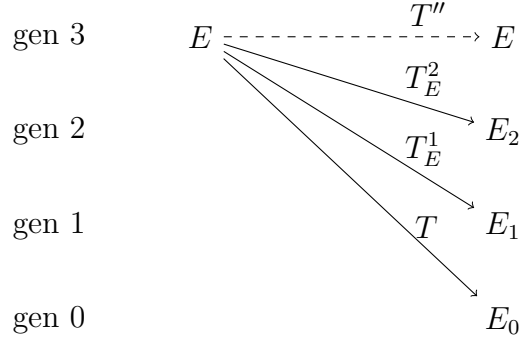


Figure 4.1: The direct refinement propagation of Example 4.2 with  $d = 4$ .

*not induce further refinement and can be neglected.*

More generally we have to analyse the direct refinement propagation for any element that contains  $E$  and additionally for all elements containing any of the edges  $E_i$ , as their refinement may also induce further refinement in the grid. This leads us to the following definition of a *Refinement Propagation Graph* that connects all edges in the conforming closure. Starting point

is the refinement of an edge  $E_0$  and we construct the graph iteratively by considering the direct refinement propagation of any new edge that appears in all adjacent elements of that edge.

**Definition 4.3 (Refinement Propagation Graph).** *Refinement of an element  $T$  with refinement edge  $E_0$  and generation  $\text{gen}(T) = l$  induces refinement propagation in form of a directed graph with root  $E_0$ . This graph is defined recursively: For any element  $T'$  with  $E_0 \subset T'$  and  $\text{gen}(T') < l$  we introduce new nodes in form of the refinement edges  $E'_i$  of  $T'_{E_0}$  for  $\text{gen}(T') < i < l$  and for each node we insert a directed edge from  $E_0$ . We iterate by setting every newly introduced node as a local root  $E_0$ .*

We call this the refinement propagation graph.

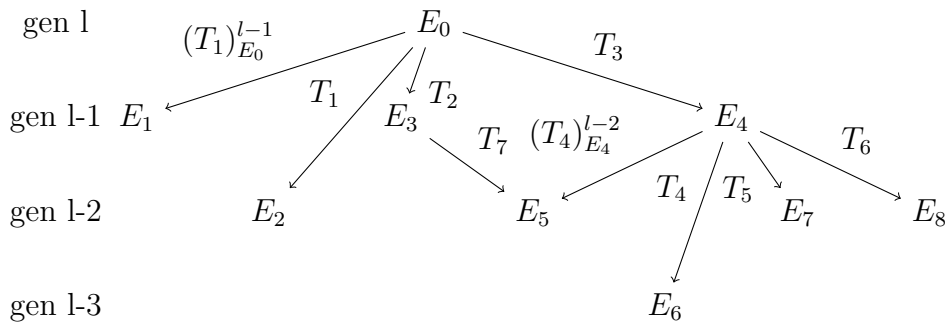


Figure 4.2: Refinement propagation graph example

Figure 4.2 depicts an example of the refinement propagation graph of an initial refinement of an element  $T$  with refinement edge  $E_0$ . For some elements the refinement does not propagate, as  $E_0$  already is the refinement edge. Hence they (and  $T$  itself) are not included in the picture. For three neighbours  $T_1, T_2, T_3$  of  $T$  the refinement does not result in the direct closure. For  $T_1$  it even results in an additional refinement of its child  $(T_1)_{E_0}^{l-1}$  that contains  $E_0$ . Bisection of  $E_4$  to refine  $T_3$  is locally similar to the refinement of  $T$  by  $E_0$ . Note that the graph is not a tree, as refinement edges may be shared ( $E_5$  in our example). All leaves do not induce further refinement, as all adjacent elements are of the same level.



**Remark 4.4.** *In 2 dimensions the refinement propagation graph consists solely of nodes of degree 2 (and the root and the leaf), since in 2 dimensions every edge is shared by exactly two elements and the direct closure is not included. Hence in 2 dimensions we call it the refinement propagation path.*

With these preparations and exploiting strong compatibility we state the following theorem which does not require any conditions on the dimension or the partitioning. It basically uses the fact that the maximum depth of the refinement propagation path is the maximum level difference to bound the number of outer iterations in Algorithm 4.1.

**Theorem 4.5 (Bound by Maximum Level Difference).** *Let  $\mathcal{M} \subset \mathcal{T}$  be the set of elements marked for refinement. Then the number  $N$  of iterations in Algorithm 4.1 to reach a conforming state satisfies*

$$N \leq \max_{T \in \mathcal{M}} \max_{T' \in \mathcal{T}} (\text{gen}(T) - \text{gen}(T')) + 2.$$

*Proof.* Let  $T \in \mathcal{M}$  with  $\text{gen}(T) = l$ . We will bound the maximum depth of the refinement propagation graph of  $T$ , which is an upper bound for the number of iterations as in the worst-case scenario refinement needs to be communicated at every edge.

Due to Lemma 4.1 refinement can be propagated at generation  $l - d < \text{gen}(T') < l$ . It is in particular propagated at generation  $l - 1$  and there is no propagation at generation  $l$ . So the maximum number of propagations  $N_T$  resulting from refining  $T$  is  $l - \min_{T' \in \mathcal{T}} \text{gen}(T')$ . This is clearly also the maximum depth of the refinement propagation graph.

We have to take into account, that the refinement propagation graph does not consider the direct closure, which could need an additional communication.

It follows

$$N_T \leq l - \min_{T' \in \mathcal{T}} \text{gen}(T') + 1$$

If we now take the maximum over all  $T \in \mathcal{M}$  this results in

$$\max_{T \in \mathcal{M}} \max_{T' \in \mathcal{T}} (l(T) - l(T')) + 1.$$

We have to add another 1 as Algorithm 4.1 has to communicate that it has finished.  $\square$

**Remark 4.6.** *Note that Theorem 4.5 in 2d basically bounds the length of the refinement propagation path. For 2 dimensions and Longest Edge bisection without compatibility condition Jones and Plassman [JP97, Theorem 3.6] have shown that a similar term is non-negligible in parallel computing.*

The following example demonstrates that the bound from Theorem 4.5 is sharp. In the worst case we cannot expect anything better even for just 2 partitions. In particular the bound  $O(\log P)$  from [WLPV15, Sec. 2.4] cannot hold without further assumptions on the decomposition.

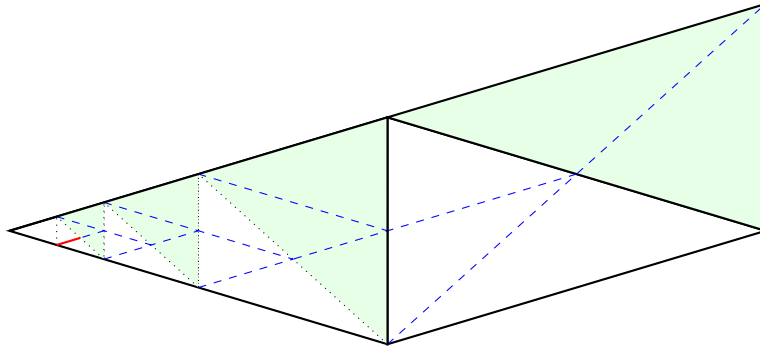


Figure 4.3: A distributed refined mesh to illustrate that the bound of Theorem 4.5 is sharp. Partitions are indicated by the color of the cells. Black lines denote macro element ( $T \in \mathcal{T}_0$ ) borders. Dotted lines denote the initial refinement situation. The refinement request is marked in red. Blue dashed lines denote its refinement propagation.

**Example 4.7.** *Figure 4.3 shows a mesh consisting of three initial triangles fulfilling the compatibility condition 3.22. One triangle has been refined into a corner, so the minimum generation in the mesh is  $\min_{T' \in \mathcal{T}} \text{gen}(T') = 0$  and the generation of the marked element  $T \in \mathcal{M} = \{T\}$  is  $\text{gen}(T) = 6$ , so the bound from Theorem 4.5 is 8 iterations. Every refinement induces additional refinement at exactly one generation lower, so the refinement propagation path traverses 7 edges.*

*The mesh is partitioned in such a way that every edge lies on a processor boundary, which implies that after every refinement Algorithm 4.1 has to stop*

and globally communicate. This means we get 7 iterations, where the marked set is not empty on all processors and one final iteration to communicate, that we are finished. In total this is 8 iterations, which is the bound predicted from Theorem 4.5.

*Distributing the elements into partitions as depicted in Figure 4.3 is not purely artificial. Sorting the leaf elements in vertical direction with respect to their center coordinates leads to these partitions.*

Example 4.7 demonstrates that we have to impose additional assumptions on the decomposition to obtain better bounds on the number of iterations of Algorithm 4.1.

## 4.2.2 Analysis Using a Special Partitioning

We analyse the required number of outer iterations in Algorithm 4.1 under assumptions on the partitioning/loadbalancing. While we describe several ways to partition the domain, we only investigate the refinement behaviour when partitioning the macro triangulation with its respective refinement forest.

A often-used method to create partitions bases on hierarchical space-filling curve (SFC). First the macro elements  $T_0 \in \mathcal{T}_0$  need to be sorted, which can be achieved using a geometric SFC [Bad12]. Then a hierarchic SFC traverses in each macro element  $T_0 \in \mathcal{T}_0$  the refinement tree  $\mathfrak{F}(T_0)$  up to the descendants in the current triangulation (i.e., leaf elements). This induces an ordering on all leaf elements, which is then split into chunks that are distributed to the processors. Common examples of these SFCs are the Sierpinski curve for triangles, the Hilbert curve and the Z curve for cubes. These ensure that the resulting chunk consists of direct neighbours within each macro element. Such a load-balancing method is used in [Bad12, SBB12] for 2-dimensional grids, that can be extended into the third dimension as prisms [MB15]. An extension to tetrahedra is not easily possible, as there is no possible SFC that traverses uniform descendants of a tetrahedron (see

[Bad12]). In this case one can use quasi-space-filling-curves, which do not always traverse over direct neighbours.

Another approach to partitioning is to sort the leaf elements (i.e., the elements of the current triangulation) directly by a geometric SFC, which is usually a good idea, but in the worst case may result in a situation as in Example 4.7. In [Zum02] it is shown, that partitioning the domain using SFCs is quasi-optimal regarding the ratio between the partition boundary and the partition volume.

In this work we assume that the current mesh  $\mathcal{T}$  is partitioned based on elements of a certain generation which are then distributed onto partitions together with their descendants in  $\mathcal{T}$ . Without loss of generality we can assume that we distribute elements on the macro level (i.e., elements  $T$  with  $\text{gen}(T) = 0$ ). If this is not the case we can set the uniformly refined grid as our new initial grid as we do not allow coarsening above this level. Partitioning based on the macro grid is currently implemented in our grid manager DUNE-ALUGRID.

This partitioning method is sub-optimal in cases, where the macro triangulation is small and some macro elements are refined adaptively very often. Then there are not enough macro elements to balance the computation load equally. This does usually not pose a problem, when representing complex geometries since this requires many macro elements. Also it is possible to refine the initial grid (adaptively) and to use the obtained grid as new macro grid with more macro elements.

When analysing the refinement behaviour using this partitioning method, the refinement propagation within one macro element does not have as much impact on the number of outer iterations as the refinement propagation between macro elements, because only refinement propagation across macro edges may have to be communicated. Theorems 4.9 and 4.14 will exploit that refinement tension can be resolved over a single layer of macro element to prove a constant bound on the number of outer iterations.

Before we prove these theorems we need to get an impression of the refinement propagation within the macro grid, when refining a certain macro element  $T_0 \in \mathcal{T}_0$  uniformly to a level  $l$ . This situation is displayed in Figure 4.4 for  $l = 6$ , where black lines show macro edges, the element  $T_0$  is refined uniformly (red) to level 6 and the conforming closure is shown in blue. We see that the refinement propagates into the layer (green background) of macro neighbours  $L_{T_0} = \{T \in \mathcal{T}_0 : \mathcal{V}(T) \cap \mathcal{V}(T_0) \neq \emptyset, T \neq T_0\}$ . Additionally to some outside direct neighbours of elements  $T \in L_{T_0}$  refinement is propagated once, if the initial refinement edge of  $T$  is the shared edge.

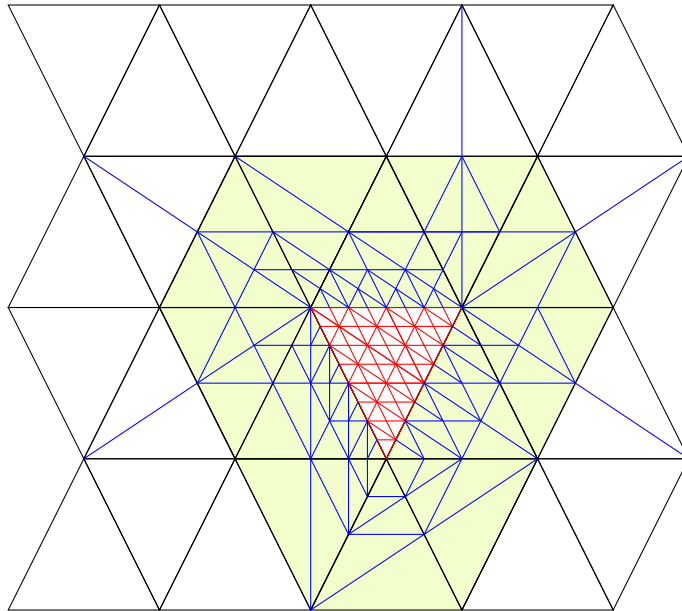


Figure 4.4: Refinement propagation from uniform refinement of a single macro element  $T_0$ . Uniform refinement is red, blue lines denote the closure, black lines belong to the initial grid. The layer of macro neighbours  $L_{T_0}$  has light green background.

Another important observation is that the refinement edge of elements containing a macro vertex  $z$  almost always contains  $z$ . This is formulated in the following lemma for 2 dimensions.

**Lemma 4.8 (Refinement Edge at Macro Vertices).** *Let  $T_0 \in \mathcal{T}_0 \subset \mathbb{R}^2$  be a macro element,  $z \in \mathcal{V}(T_0)$  be a macro vertex and  $T \in \mathfrak{F}(T_0)$  with  $z \in T$ .*

Then the refinement edge  $E$  of  $T$  contains  $z$ , if  $\text{gen}(T) \geq 1$ . Additionally if the initial refinement edge  $E_0$  of  $T_0$  contains  $z$ , this also holds for  $\text{gen}(T) = 0$  (i.e.,  $T = T_0$ ).

*Proof.* We know that, if  $z \in E_0$ , the angle located at  $z$  is split exactly once and, if  $z \notin E_0$ , the angle is never split. This proves the claim.  $\square$

The following results hold for partitioning on any fixed level but only for 2-dimensional grids, as we rely on the fact that we have a refinement propagation path instead of a full graph (cf. Remark 4.4). We basically prove that refinement propagation starting from a descendant  $T$  of  $T_0 \in \mathcal{T}_0$  can only traverse the layer of macro neighbours  $L_{T_0}$ . This is unless an initial refinement edge of a neighbour is on the outside of this layer. A lot of effort in this proof goes into finding a worst-case estimate. Proving that a constant exists is a lot easier.

**Theorem 4.9 (2d Bound by Number of Neighbours).** *Let  $\mathcal{T} \subset \mathbb{R}^2$  be partitioned such that all elements  $T \in \mathcal{T}$  that share an ancestor  $T_0 \in \mathcal{T}_0$  are on the same partition. Let  $z \in \mathcal{V}$  and let  $N_z = \{T \in \mathcal{T}_0 : z \in T\}$  be the set of macro elements containing that vertex. For an element  $T$  in the set of marked elements  $\mathcal{M}$  let  $T_0(T) \supset T$  be the element of the macro grid  $\mathcal{T}_0$  that contains  $T$ . Then the number of global communications  $N$  in Algorithm 4.1 satisfies*

$$N \leq \max_{T \in \mathcal{M}} \max_{z \in \mathcal{V}(T_0(T))} \frac{3}{4} \#N_z + \frac{7}{4} \leq \max_{z \in \mathcal{T}_0} \frac{3}{4} \#N_z + \frac{7}{4}.$$

*Proof.* The second inequality is trivial. The proof of the first inequality splits into three parts.

1. Refinement propagation around a macro vertex.
2. Relate  $\#\mathcal{T}_z$  to  $\#N_z$ .
3. Refinement propagation outside macro neighbour layer  $L_{T_0}$ .

We start with a similar observation as in the proof of Theorem 4.5. By bounding the traversals of edges of  $\mathcal{T}_0$  within the refinement propagation

path we bound the number of global communications. We count edges in the refinement propagation path that are subedges of  $\mathcal{E}_0$ .

Let  $T \in \mathcal{M}$  be the element to be refined. Refinement of  $T$  creates children elements of generation  $l = \text{gen}(T) + 1$ . All refinement propagation across macro edges induced by refinement of  $T$  is contained in the refinement propagation across macro edges of uniformly refining the macro element  $T_0 = T_0(T)$  to this level (cf. Figure 4.4). So by bounding the amount of macro edge traversals of this configuration, we surely bound the macro edge traversals of refinement of  $T$ .

**(1): Refinement propagation around a vertex  $z \in \mathcal{V}(\mathbf{T}_0)$**

From Lemma 4.8 we know that elements around the vertex  $z$  form the refinement propagation path of  $T'$  unless we encounter an element of generation 0. By definition consecutive elements inside a refinement propagation path need to differ by one in level, otherwise there would be no refinement propagation.  $T_0$  has two neighbours at the vertex  $z$ . In both directions around  $z$  the elements in the refinement propagation path decreases by one in generation (cf. Figure 4.5).

So the maximum length of the refinement propagation path around  $z$  is  $(\#\mathcal{T}_z - 1)/2$ , as there have to exist at least two elements with lowest generation.

The number of macro elements  $\#N_z$  that contain  $z$  is smaller than its environment  $\#\mathcal{T}_z$ , as for every element in  $\#N_z$  where the refinement edge is opposite of  $z$ , there are two elements in  $\#\mathcal{T}_z$ . We want to know the macro edge traversals, so we want the bound to be based on  $\#N_z$ .

**(2): Relate  $\#\mathcal{T}_z$  to  $\#N_z$**

Figure 4.5 depicts the worst possible relation between  $\#\mathcal{T}_z$  and  $\#N_z$  when considering macro edge traversals. The element where the refinement originates is marked in red and has level  $l$ . As several iterations of refinement have passed the grid is maximally stressed around the central vertex. So there are two elements of highest generation and from them every element

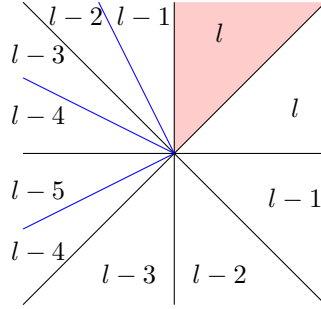


Figure 4.5: Zoom into the refinement situations at a macro vertex. Black lines denote macro edges, blue lines are edges of the first refinement. The original element is marked in red.

the level decreases by one in each direction. We observe that on the left side of the marked element each macro angle (black lines) has been refined once as the initial refinement of the corresponding macro element is opposite of the vertex. On the other side of the red element no macro angle has been refined.

This kind of distribution is the worst as the refinement path in one direction traverses a macro edge every second edge and the path in the other direction traverses a macro edge every edge. The path then may traverse up to  $2/3$  of the macro elements until it encounters the one with lowest level. We consider two cases, depending whether  $\#N_z$  is even or odd.

a.  $\#N_z$  is even:

The worst case is  $\#\mathcal{T}_z = 3/2\#N_z$  and all refined macro angles are neighbouring and we may need  $(3/2\#N_z - 1)/2$  macro edge traversals until we encounter the element with the lowest level.

b.  $\#N_z$  is odd:

The worst case is  $\#\mathcal{T}_z = 3/2\#N_z + 1/2$  and all refined angles are neighbouring and we need  $(3/2(\#N_z) + 1/2 - 1)/2$  macro edge traversals until we encounter the element with the lowest level.

So the number of macro edge traversals  $N_z^E$  within the refinement propaga-



tion path around  $z$  satisfies

$$N_z^E \leq (3/2(\#N_z - 1/2))/2 = 3/4\#N_z - 1/4.$$

**(3): Refinement propagation outside neighbour layer**

First we observe that if we take the maximum over the three vertices of  $T_0$

$$N_z^{T_0} \leq \max_{z \in \mathcal{V}(T_0)} N_z^E$$

this already bounds all macro edge traversals within the layer  $L_{T_0}$ . So we need to discuss, whether refinement can propagate into an element that is not a neighbour.

By Lemma 4.8 we know that refinement propagation that contains a macro vertex  $z \in \mathcal{V}(T_0)$  can only reach an element outside, if the initial refinement edge of a neighbour  $T' \in \mathbb{N}_z$  does not contain  $z$ . So we have to add +1 to the bound to include this case.

In all other cases it is not possible to reach a macro element that is not a neighbour. Any refinement propagation leaving  $T_0$  contains a vertex  $v \in \mathcal{V}(\mathcal{T}_k)$  for some  $k$  and we can use the above argument using  $\mathcal{T}_k$  as “macro” grid.

This yields

$$N^{T_0} \leq N_z^{T_0} + 1 \leq (3/2\#N_z - 1/2)/2 + 1 = 3/4\#N_z + 3/4.$$

Now we finish the proof by adding another +1 for communicating the final status and taking the maximum over all marked elements and their respective macro ancestors.

$$N \leq \max_{T \in \mathcal{M}} N^{T_0(T)} + 1$$

□

There are a couple of remarks to the result of this theorem.

**Remark 4.10.** From part (3) of the proof we observe the following: If each element  $T_0 \in \mathcal{T}_0$  has been refined at least once, then the +1 from this part disappears and we get

$$N \leq \max_{T \in \mathcal{M}} \max_{z \in T_0(T)} 3/4 \#N_z + 3/4 \leq \max_{z \in \mathcal{T}_0} 3/4 \#N_z + 3/4.$$

**Remark 4.11.** Let  $P_z$  be the set of partitions that contain  $z$ . If we do not count all edges of  $\mathcal{T}_0$ , but only those that are actually processor boundaries, we get the following bound that is usually better, as we do not expect each macro element at a vertex to be on a different partition.

$$N \leq \max_{T \in \mathcal{M}} \max_{z \in T_0(T)} \#P_z - 1 + 1 = \max_{T \in \mathcal{M}} \max_{z \in T_0(T)} \#P_z$$

The proof is similar to the proof of Theorem 4.9, but we get  $\#P_z - 1$  as we cannot argue with circumventions in both directions and the +1 is again due to the final communication. Note that if the intersection of a partition and  $N_z$  is not connected, every connected subset has to be counted as a separate partition for the above bound.

**Remark 4.12.** Theorem 4.9 provides a constant that is independent from the number of processors. It just depends on a regular initial mesh (see  $\alpha_0$  in Lemma 3.11).

A similar result holds for 3 dimensional grids. We will not spend as much effort to prove the worst-case but instead just prove a constant. We need the following lemma, which is designed to treat direct neighbours of a refined element (cf. Figure 4.4, direct neighbours of the red element). The lemma expects refinement propagation from a direct neighbour and states that the resulting refinement propagation inside the element stays close to the connecting face.

**Lemma 4.13.** Let  $T = [z_0, z_1, z_2, z_3]_0$  be a 3-simplex and  $F$  be a face of  $T$ . Then any refinement request of an edge contained in  $F$  can only lead to refinement of an edge  $E$  containing the macro vertex  $v$  opposite of  $F$  if  $E = \overline{z^*v}$  with  $z^* \in F$ .

*Proof.* We show that once  $T$  has been refined uniformly once, no edge containing  $z_3$  gets refined anymore. We denote by  $z_{ij}$  the midpoint of  $\overline{z_i z_j}$ . Then one uniform refinement of  $T$  leads to the following simplices:

$$\begin{aligned} A &= [z_0, z_{01}, z_{02}, z_{03}], & B &= [z_1, z_{01}, z_{02}, z_{03}] \\ C &= [z_2, z_{12}, z_{02}, z_{03}], & D &= [z_1, z_{12}, z_{02}, z_{03}] \\ E &= [z_2, z_{12}, z_{13}, z_{03}], & F &= [z_1, z_{12}, z_{13}, z_{03}] \\ G &= [z_2, z_{23}, z_{13}, z_{03}], & H &= [z_3, z_{23}, z_{13}, z_{03}] \end{aligned}$$

As the refinement of  $T$  is equivalent to  $T_R = [z_3, z_2, z_1, z_0]_0$  (see[Ste08]), we can split the proof into two cases:  $v = z_3$  and  $v = z_2$ . We request refinement of all edges contained in  $F$ . The refinement propagation graphs (without writing the elements corresponding to the arrows) look as shown in Figures 4.6 and 4.7. As one can clearly see, in both cases no edge containing  $v$  has

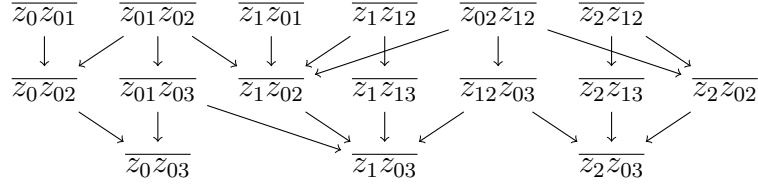


Figure 4.6: Refinement propagation graph for  $v = z_3$

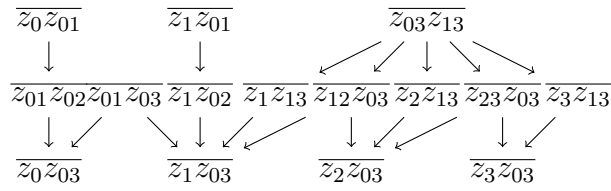


Figure 4.7: Refinement propagation graph for  $v = z_2$

been bisected. □

We use Lemma 4.13 to prove that the refinement propagation created by refinement of a descendant  $T$  of  $T_0 \in \mathcal{T}_0$  mostly stays in the layer of macro

neighbours  $L_{T_0}$ . We bound the number of macro edge traversals by the macro edges that are present at each vertex of  $T_0$ .

**Theorem 4.14 (3d Bound by Number of Neighbours).** *Let the grid  $\mathcal{T} \subset \mathbb{R}^3$  be partitioned as in Theorem 4.9. Let  $z \in \mathcal{V}(\mathcal{T}_0)$  and let  $N_z = \{T \in \mathcal{T}_0 : z \in T\}$  the set of macro elements containing  $z$ . For an element  $T$  in the set of marked elements  $\mathcal{M}$  let  $T_0(T) \supset T$  be the element of the macro grid  $\mathcal{T}_0$  that contains  $T$ . Then the number of global communications  $N$  in Algorithm 4.1 satisfies*

$$N \leq \max_{T \in \mathcal{M}} \max_{z \in \mathcal{V}(T_0(T))} \#N_z/2 + 1.$$

*Proof.* As in the proof of Theorem 4.9 we will investigate the situation where one element  $T \in \mathcal{M}$  has been refined uniformly together with its conforming closure under the assumption that no element had been refined before.

Lemma 4.13 tells us that for a direct neighbour  $T'$  that shares a face  $F$  with  $T$ , all edges that contain the opposite macro vertex  $v = \mathcal{V}(T') \setminus \mathcal{V}(F)$ , are either macro edges, or direct children of macro edges. This directly implies that the leaf element(s) that contain  $v$  can be at most of generation  $d+1(= 4)$ , if the initial refinement edge of  $T'$  was contained in  $F$ , and  $d(= 3)$  else.

We get the same result for all macro neighbours  $T''$  of  $T$ . This means for all elements that are neighbours, but not direct neighbours of  $T$ , the face opposite of the shared vertex may have been refined at most  $d-1(= 2)$  times. So every edge of these faces may have been refined once.

As we communicate over edges, this adds +1 to the proof. We also need a +1 due to the final communication.

So we need to bound the communication inside the layer of elements around  $T_0$ . For each vertex  $z$  in  $\mathcal{V}(T_0)$  we use the number of macro elements  $\#N_z$  that contain  $z$ . The number of macro edge traversals is surely smaller than  $\#N_z/2 - 1$ , as we always communicate to at least two new macro neighbours and we exclude the starting element  $T_0$ .

In total this leads to the proposed result by taking the maximum over all macro vertices  $z \in \mathcal{V}(T_0(T))$  for all marked elements  $T \in \mathcal{M}$ .  $\square$

**Remark 4.15.** *We believe that Theorem 4.14 holds in a similar way for dimensions  $d > 3$ . The proof is very similar, but proving Lemma 4.13 becomes complex.*

The estimate from Remark 4.11 also holds for 3 dimensions. Based on this estimate we propose a new improved algorithm for compatible meshes.

---

**Algorithm 4.2:** Improved Distributed Newest Vertex Bisection

---

- 1 Initialize set  $\mathcal{M}_i$  of elements marked for refinement on each partition  $P_i$ ,  $0 \leq i < P$ .
  - 2 **for**  $i = 0, \dots, \max_{z \in \mathcal{T}_0} \#P_z - 1$  **do in parallel**
  - 3     Refine partition  $P_i$  using NVB until  $\mathcal{M}_i$  is empty
  - 4     Communicate refinement status of partition boundary to corresponding neighbour partition
  - 5     Add nonconforming simplices to  $\mathcal{M}_i$
- 

**Remark 4.16** (Improved Distributed Newest Vertex Bisection). *For strongly compatible meshes we have proven, that this algorithm yields the conforming closure and reaches the final state. So communicating the final state is no longer necessary. Hence we take the better bound  $\max_{z \in \mathcal{T}_0} \#P_z - 1$  instead of  $\max_{z \in \mathcal{T}_0} \#P_z$ . We can directly use the bound from Theorem 4.9 and get an algorithm, which does not need global communication at all. Unfortunately, we cannot expect meshes to be strongly compatible. In this case we propose to use a mixture of both algorithms (Algorithm 4.3), where a fixed number of loops is done like in 4.2 before switching to the Algorithm 4.1 after a global communication to check, whether  $\mathcal{M}_i \neq \emptyset \forall i$ .*

---

**Algorithm 4.3:** Mixed Distributed Newest Vertex Bisection

---

```
1 Initialize set  $\mathcal{M}_i$  of elements marked for refinement on each partition
    $P_i$ ,  $0 \leq i < P$ .
2 for  $i = 0, \dots, \max_{z \in \mathcal{T}_0} \#P_z - 1$  do
3   | Refine partition  $P_i$  using NVB until  $\mathcal{M}_i$  is empty
4   | Communicate refinement status of partition boundary to
   | corresponding neighbour
5   | Add nonconforming simplices to  $\mathcal{M}_i$ 
6 Communicate globally, whether  $\mathcal{M}_i$  is empty.
7 while  $\mathcal{M}_i \neq \emptyset \forall i$  do
8   | Refine partition  $P_i$  using NVB until  $\mathcal{M}_i$  is empty
9   | Communicate refinement status of partition boundary to
   | corresponding neighbour
10  | Add nonconforming simplices to  $\mathcal{M}_i$ 
11  | Communicate globally, whether  $\mathcal{M}_i$  is empty.
```

---

### 4.3 Weakly Compatible Refinement Propagation

In the final stage of this thesis we have found an additional result for weakly compatible grids. We show that once an arbitrary-dimensional weakly compatible mesh has been refined to a certain level refinement propagation remains inside the layer of macro elements around the macro element where it originates.

First, we show that no angle that is adjacent to a refinement edge is ever refined.

**Lemma 4.17 (Descendants at Vertex).** *Let  $T = [z_0, \dots, z_d]_{t_T} \in \mathfrak{F}(\mathcal{T}_0)$  have refinement edge  $E = \overline{z_0 z_d}$ . Then for any descendant  $T' \in \mathfrak{F}(T)$  that contains  $z_0$  the refinement edge of  $T'$  also contains  $z_0$ .*

*Proof.* The child of  $T$  that contains  $z_0$  is

$$T_{child} = [z_0, z_{0d}, z_1, \dots, z_{d-1}]_{t_T \bmod d}$$

and its refinement edge contains  $z_0$ . Now we can iterate the argument to obtain any descendant that contains  $z_0$ .  $\square$

Note that the above result also holds for the vertex  $z_d$  due to NVB-equivalence (cf. Section 3.1). Now we can prove that refinement propagation remains local after  $d$  uniform refinements.

**Theorem 4.18 (Uniform Refinement leads to Locality).** *Let  $\mathcal{T}_0$  consist solely of elements of type  $t_T \in \{0, 1\}$ . Then after refining the full grid to  $\mathcal{T}_d$ , any refinement from any element  $T \in \mathfrak{F}(T_0)$  for a macro element  $T_0 \in \mathcal{T}_0$  does at most propagate into the layer of macro neighbours.*

*Proof.* Let  $T \in \mathfrak{F}(T_0)$  be the marked element. As in the proofs of Theorems 4.9 and 4.14 we first consider an element at the macro vertex  $z \in \mathcal{V}(T_0)$ . Because the grid  $\mathcal{T}_0$  contains only elements of type 0 or 1, all macro edges  $E \in \mathcal{E}_0$  have been refined (Lemmas 3.4 and 3.6). So by Lemma 4.17 every edge in the refinement propagation graph needs to contain  $z$ . This means for these elements the refinement propagation remains in the layer of macro neighbours.

Now we consider any other element  $T \in \mathfrak{F}(T_0)$  which does not contain a macro vertex  $z \in \mathcal{V}(T_0)$ . This element cannot be in  $\mathcal{T}_1$ , as each element in  $\mathcal{T}_1$  contains a macro vertex. So  $T \in \mathcal{T}_k$ , with  $k \in \{2, \dots\}$ . If refinement propagation of this element leads to refining an element  $T_1 \in \mathcal{T}_1$ , then  $T_1$  contains a macro vertex  $z \in \mathcal{V}(T_0)$ . This proves the claim, as we now are in the above case.  $\square$

**Remark 4.19.** *The condition of Theorem 4.18 is fulfilled by a descendant of weakly compatible grids by definition (Definition 3.12). For grids created by Algorithm 3.1 the descendant that fulfils this condition is the descendant that contains only edges of level 1 (Lemma 3.36).*

Theorem 4.18 basically exploits that each tree of a macro element is strongly compatible internally and all additional refinement propagation can only happen at faces, that are contained in weakly compatible macro faces. The strong compatibility inside each element ensures, that although in principle the refinement propagation can traverse the full initial grid, this may only happen on the first  $d$  levels. For finer levels the refinement propagation is caught by the macro neighbours. This yields the following corollary which uses the simple estimate that all we can at most refine descendants of each edge once.

**Corollary 4.20.** *Let  $\mathcal{T}_0$  consist solely of elements of type  $t_T \in \{0, 1\}$ , let  $\mathcal{T}$  be at least as fine as  $\mathcal{T}_d$  and be partitioned as in Theorem 4.9. Let  $\mathcal{E}_z = \{E \in \mathcal{E}_0 : z \in E\}$  the set of macro edges containing  $z$ . For an element  $T$  in the set of marked elements  $\mathcal{M}$  let  $T_0(T) \supset T$  be the element of the macro grid  $\mathcal{T}_0$  that contains  $T$ . Then the number of global communications  $N$  in Algorithm 4.1 satisfies*

$$N \leq \max_{T \in \mathcal{M}} \max_{z \in \mathcal{V}(T_0(T))} \#\mathcal{E}_z + 1.$$

We add +1 because of the final communication.

**Remark 4.21.** *We expect that Theorem 4.18 can be combined with the theorems proving the constant parallel overhead (Theorem 4.9 and 4.14) to show a similar estimate as in [Ste08] that the effort of the conforming closure solely depends on the number of all elements that have been marked including all previous refinement. In particular, we expect that the result holds at least for meshes generated by Algorithm 3.1, which is a subclass of all weakly compatible meshes. Also we expect an additive constant depending on the initial grid which is negligible in the limit.*

## 4.4 Numerical Experiments

In this section we show for various examples that the theoretical results can be reproduced and that very good scalability for the adaptation algorithm is observed on a supercomputer. The NVB algorithm is implemented in DUNE-ALUGRID (compare Section 5.1).



### 4.4.1 Verification of Theoretical Results

The first experiment aims to reflect the theoretical results as we construct a worst-case experiment. A mesh is partitioned such that each process gets exactly one macro element. Then we refine a single element at one of its vertices up to level 20 and we examine the number of iterations necessary to reach the conforming state.

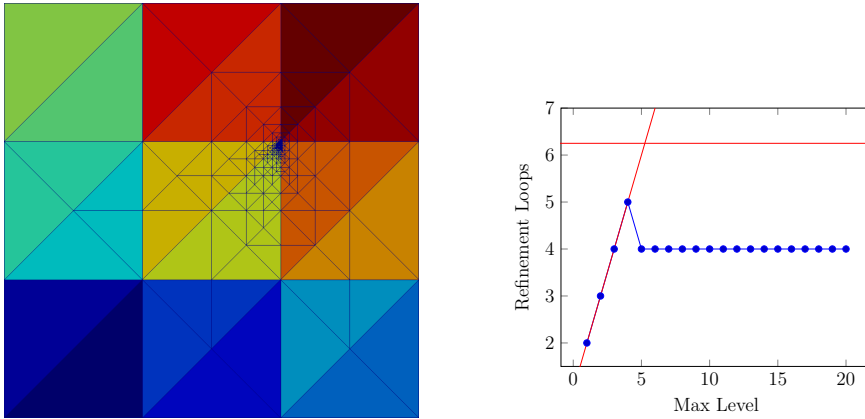


Figure 4.8: Left: A 2d unit square with 18 macro elements on 18 processors. The central yellow element gets refined at the top vertex. The number of macro neighbours  $\#N_z = 6$ . In the right plot red lines denote the two bounds from Theorems 4.9 and 4.5.

From Figure 4.8 we see that for compatible 2d grids both bounds (red lines) are not violated by the current implementation. As this is a worst-case scenario, in the average case we expect the behaviour to be better. Now we perform the same experiment on a weakly compatible grid.

As expected, the plot in Figure 4.9 illustrates that compatibility is a necessary condition for both theorems. This means although the implementation is capable of handling weakly compatible 2d grids, the bounds from the theory do not hold directly. Nevertheless after a few iterations the bound holds also for weakly compatible grids, as the refinement inside each element is strongly compatible, which in combination with Corollary 3.32 guarantees, that the refinement at a weakly compatible face remains local around the macro vertex contained in the face.

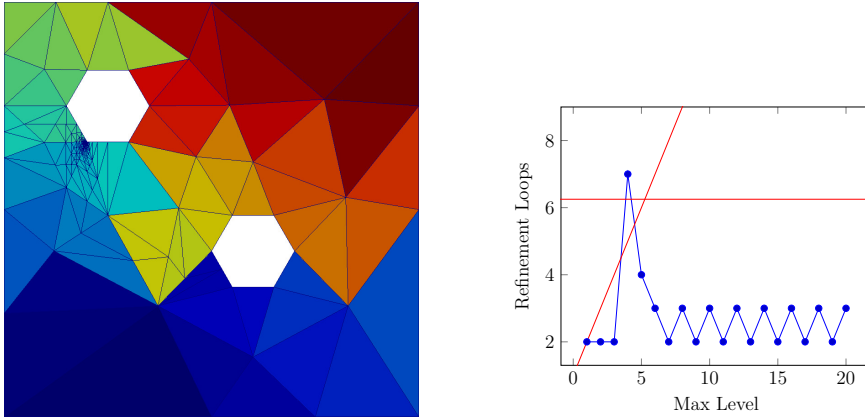


Figure 4.9: Left: A weakly compatible 2d grid with holes containing 60 macro elements on 60 processors. An element (with  $\#N_z = 6$ ) gets refined at a vertex. In the right plot red lines denote the two bounds from Theorems 4.9 and 4.5.

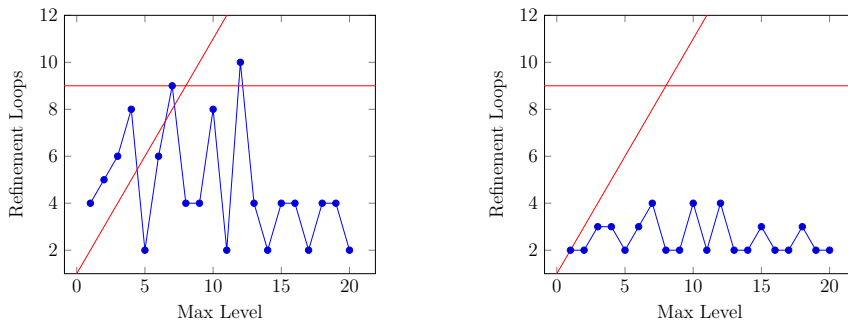


Figure 4.10: Global communication of 3d experiment. On the left refinement status is communicated on faces. On the right refinement status is first communicated over edges and then additionally over faces. Red lines denote the bound from Theorems 4.5 and 4.14

The same setup in 3 dimensions looks as follows. We consider a unit cube with 162 macro elements on 162 processors ( $3 \times 3 \times 3$  Kuhn-cubes) and a simplex of the central Kuhn-cube is refined at one vertex  $z$  ( $\#\mathcal{E}_z = 8$ ). We expect the bounds from Theorem 4.5 and 4.14 to hold in 3d, but the left side of Figure 4.10 seems to be a counterexample. This failure arises from an implementation detail, that refinement state is communicated for faces instead of edges. After implementation of an additional communication of edge refinement state during refinement we see that the theoretical results

hold (Figure 4.10, right side).

#### 4.4.2 Strong Scaling Experiments

In Figure 4.11 we show the refinement in the form of a doughnut that rotates around the center of the unit cube. Instead of using the solution to a partial differential equation to determine the zones for grid refinement and coarsening, a simple boolean function  $E \mapsto \eta_E$  is used. We refine all elements located near the surface of a ball rotating around the center of the 3d unit cube:

$$\begin{aligned} \mathbf{y}(t) &:= \left( \frac{1}{2} + \frac{1}{3} \cos(2\pi t), \frac{1}{2} + \frac{1}{3} \sin(2\pi t), \frac{1}{2} \right)^T, \\ \eta_E &:= \begin{cases} 1 & \text{if } 0.15 < |\mathbf{x}_E - \mathbf{y}(t)| < 0.25, \\ 0 & \text{otherwise,} \end{cases} \end{aligned} \quad (4.1)$$

where  $\mathbf{x}_E$  denotes the barycenter of the element  $E$ . A cell  $E$  is marked for refinement, if  $\eta_E = 1$  and for coarsening otherwise. For the 2d test we simply neglect the third component of  $\mathbf{y}(t)$ .

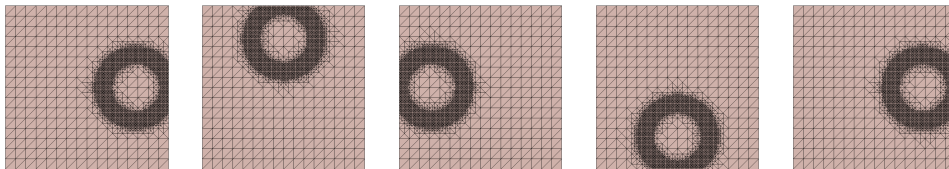


Figure 4.11: Refinement and coarsening of a doughnut like area that rotates around the center. From left to right the simulation time is increased from  $t = 0$  to  $t = 1$  by  $\Delta t = 0.25$ .

This test was introduced in [ADKN16] and demonstrates the distributed NVB since frequent refinement and coarsening typically occurs throughout simulations. In fact, the adaptation and load balancing is performed in each time step. The domain decomposition is based on the Hilbert space filling curve approach implemented in Zoltan [BCCD12] and the balancing of the curve is implemented in DUNE-ALUGRID based on the algorithm presented in [BBHK11]. The implementation of this test case (`main_ball`) is contained

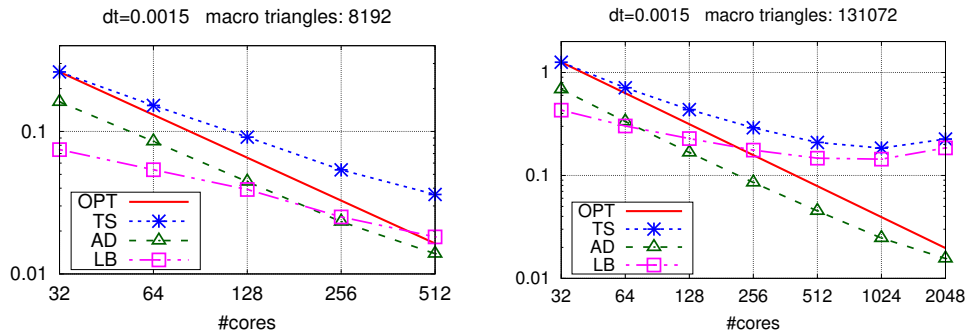


Figure 4.12: 2d results for the doughnut refinement test with  $\Delta t = 0.0015$  on a coarse triangular macro mesh (left) and a finer triangular macro mesh (right). For different number of cores the graph shows average run time per timestep in seconds for the different parts of the algorithm: a full time step (TS), the adaptation loop (AD), the load balancing (LB), and the expected optimal scaling (OPT). The scaling study has been performed on Yellowstone [NCA12]. The test case is part of the DUNE-ALUGRID code and described in [ADKN16]. In both cases we set the maximal refinement level to 12. About 1–2% of new mesh elements are refined during each time step. An according number of elements are coarsened. Overall the number of leaf elements is approximately 1 million (left) and 4.5 million (right).

in the DUNE-ALUGRID package (sub folder examples).

All presented experiments in this section are based on version 2.4 of the DUNE core modules. In the graphs (Figures 4.12 – 4.15) we present three measurements taken from the simulations. TS denotes the average time spent for one timestep, AD denotes the time spent for the complete adaptation loops described in Algorithm 4.1 and 4.2, and LB denotes the average time spent to complete the load balancing for one timestep.

Figures 4.12 and 4.13 provide strong scaling results obtained for the doughnut refinement test with  $\Delta t = 0.0015$  in 2d and 3d, respectively. In both cases about 1–2% elements are newly created every time step and an according number of elements are coarsened. More detailed information is provided in the figure caption.

In Figure 4.14 scaling results for the same experiment with a larger timestep,  $\Delta t = 0.01$ , is presented. The larger timestep leads to about

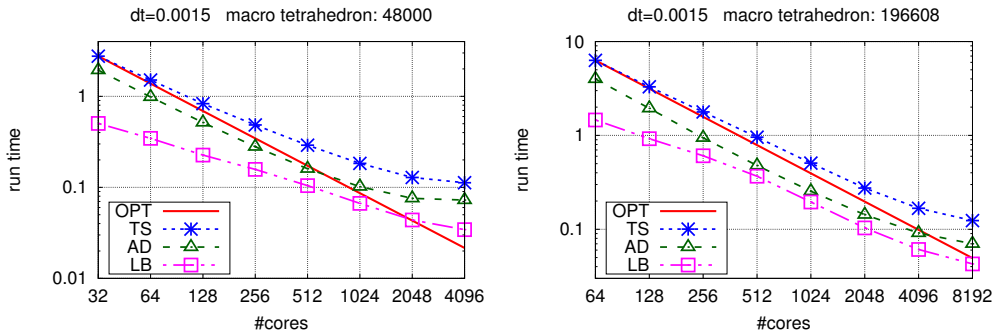


Figure 4.13: 3d results for the doughnut refinement test with  $\Delta t = 0.0015$  on a coarse tetrahedral macro mesh (left) and a finer tetrahedral macro mesh (right). For different number of cores the graph shows average run time per timestep in seconds for the different parts of the algorithm: a full time step (TS), the adaptation loop (AD), the load balancing (LB), and the expected optimal scaling (OPT). The scaling study has been performed on Yellowstone [NCA12]. The test case is part of the DUNE-ALUGRID code and described in [ADKN16]. In both cases we set the maximal refinement level to 12. In addition, we keep a minimal refinement level of 3 in the right case. About 1–2% of new mesh elements are refined during each time step. An according number of elements are coarsened. Overall the number of leaf elements is approximately 11 million (left) and 45 million (right).

10 – 20% of elements being newly created every timestep. In Figure 4.15 we show a comparison between the old implementation of the adaptation algorithm and the improved implementation of the adaptation algorithm. We notice that the number of global synchronizations is reduced by about 50%. As pointed out in Remark 4.16, even though the number of iterations can be computed a priori in some cases it still might be better to use a combination of both, i.e. iterate an average number of times without global synchronization and after that with global synchronization. An example use case for this are weakly compatible macro grids since they cannot fulfil the bounds depending on strong compatibility. Another use case is quasi-global, i.e., refinement that is not varying much over the whole grid. In this case we often just need one or two outer iterations in the parallel refinement algorithm to reach the conforming state.

The scaling experiments have been performed on the supercomputer Yel-

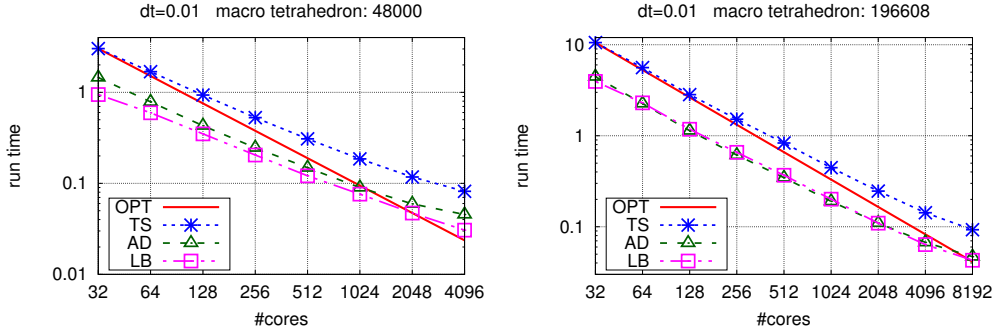


Figure 4.14: 3d results for the doughnut refinement test with  $\Delta t = 0.01$  on a coarse tetrahedral macro mesh (left) and a finer tetrahedral macro mesh (right). For different number of cores the graph shows average run time per timestep in seconds for the different parts of the algorithm: a full time step (TS), the adaptation loop (AD), the load balancing (LB), and the expected optimal scaling (OPT). The scaling study has been performed on Yellowstone [NCA12]. The test case is part of the DUNE-ALUGRID code and described in [ADKN16]. In both cases we set the maximal refinement level to 12. In addition, we keep a minimal refinement level of 3 in the right case. About 10 – 20% of new mesh elements are refined during each time step. An according number of elements are coarsened. Overall the number of leaf elements is approximately 9 million (left) and 33 million (right).

lowstone [NCA12]. The observed strong scaling for the adaptation cycle (green triangular line) is excellent for all experiments and very close to the optimal scaling. The load balancing (pink boxed line) on the other hand at some points fails to scale well because the number of macro simplices per core becomes too small as the number of processes grows which results from the drawback of basing the partitioning upon the macro mesh. In contrast to [WLPV15, Fig. 8], where for a different adaptation experiment the *mesh adaptation loop* yielded non-optimal strong scaling, we conclude from our investigations that the distributed NVB scales well and depending on the macro mesh none or only very few global communications are needed.

For a higher number of cores the strong scaling of the adaptation algorithm deteriorates when the ratio of macro elements per core gets to small, e.g. below 100. In that case, if the mesh refinement is localized, no satisfactory domain decomposition can be computed. This is a weakness of the current

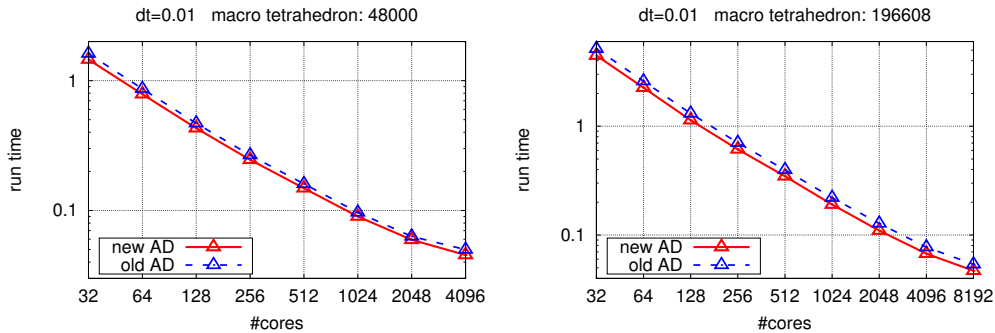


Figure 4.15: 3d results for the doughnut refinement test with  $\Delta t = 0.01$  on a coarse tetrahedral macro mesh (left) and a finer tetrahedral macro mesh (right). For different number of cores the graph shows average run time per timestep in seconds for the improved adaptation loop from Algorithm 4.2 (new AD) and the conventional adaptation loop from Algorithm 4.1 (old AD). The scaling study has been performed on Yellowstone [NCA12]. The test case is part of the DUNE-ALUGRID code and described in [ADKN16]. In both cases we set the maximal refinement level to 12. In addition, we keep a minimal refinement level of 3 in the right case. Both runs show that the new version of the algorithm performs better. In general the runtime of the adaptation loop is reduced by about 10% to 20%.

strategy to base the domain decomposition on the coarsest grid. On the other hand, this allows us to specify the number of global synchronizations needed during an adaptation cycle. As a result, the next undertaking is to implement a version that will allow to select any level in the hierarchy as a basis for the domain decomposition to exploit both, a fixed or small number of global synchronizations and an nearly equal balance of work load for each core.

# Chapter 5

## Some Implementational Aspects

This chapter covers aspects of the software that we developed and extended over the course of this thesis. While this chapter contains only a small scientific contribution, the implementation within represents a major part of the overall work. A considerable amount of time and effort has been spent to implement the presented features.

Most implementation in this work is done within the software framework DUNE [BBD<sup>+</sup>06, BBD<sup>+</sup>08], in particular DUNE-ALUGRID [ADKN16], DUNE-FEM [DKNO10] and DUNE-ACFEM [Hei14]. DUNE-ALUGRID provides an unstructured, adaptive grid manager that can handle simplicial and cubical elements in 2 and 3 dimensions and also conforming and non-conforming refinement. DUNE-FEM provides discrete functions, linear solvers and structures to deal with parallel FEM computations. DUNE-ACFEM is a convenience module on top of DUNE-FEM that allows to simply write mathematical expressions as code, which invokes the corresponding DUNE-FEM based implementation.

We extended the grid manager DUNE-ALUGRID by unifying the 2d and 3d mesh implementations. This has been published in *Archive of Numerical Software* by the author, Andreas Dedner, Robert Klöfkorn and Martin Nolte



[ADKN16]. The idea behind the implementation is not mentioned in the publication and is therefore introduced in Section 5.1. This approach aims at reducing the amount of code and hence increasing its maintainability by reusing part of the 3d grid implementation. The given 2d grid results in a 3d grid construction internally and part of the 3d surface is exported as the managed 2d grid to the user. This is possible as DUNE-ALUGrid consists of two layers: the internal layer has been originally implemented by Bernhard Schupp [Sch99] as part of a PhD thesis and is the actual grid implementation, and a second layer implements the DUNE-Grid interface.

Section 5.2 briefly introduces the DUNE module DUNE-ACFEM which has been developed by Claus-Justus Heine, Birane Kane, Stephan Hilb and the author. DUNE-ACFEM aims to simplify the implementation of Finite-Elements for adaptive, parallel grids using the DUNE infrastructure. DUNE-ACFEM uses expression templates for functions and equation models, which can be added and multiplied. This greatly simplifies the implementation of complex equations as each part of the equation may be implemented separately and can afterwards be assembled by the compiler. This has been optimized such that known zero-expressions are discarded during compile-time. A drawback of this approach are long compilation times.

Finally in Section 5.3 we describe the proof-of-concept implementation of an arbitrary-dimensional simplex grid with NVB refinement under the weak compatibility condition, which is used for the experiments in Section 3.4.6. We choose to store the connectivity information on the edges of the grid. This allows for a convenient implementation of the refinement algorithm, which appears to scale linearly with the number of elements refined in arbitrary dimension.

## 5.1 Dune-ALUGrid

The grid manager and DUNE module DUNE-ALUGRID provides adaptive, distributed, nested grids in 2 and 3 dimensions using either cubes or simplices. This also includes surface meshes, i.e., 2 dimensional grids in a 3

dimensional coordinate system with multilinear elements. The refinement strategy can be chosen to be conforming (NVB) or non-conforming (isometric refinement). The program was originally developed as part of a PhD thesis [Sch99] and provided distributed tetrahedral meshes and cube meshes with isometric refinement. Afterwards conforming refinement in the form of NVB has been added. Once the code a DUNE module, a serial (non-distributed) 2d implementation and a DUNE interface layer have been added.

The grid manager distinguishes between topological and geometric concepts and aims to rely on topological information.

Within this thesis the 2d implementation was reimplemented to reuse the 3d implementation. The goal was to exploit the existing topological structures and parallelization to get a 2d implementation with working parallelism while maintenance. Also the refinement algorithm within DUNE-ALUGRID was reworked to allow weakly compatible NVB grids in 3 dimensions and the relabelling Algorithm 3.1 was implemented.

To extend the 2d grid as a 3d grid we use a projection approach. 2d simplices are modelled as 3d simplices and 2d cubes are modelled as 3d cubes. Afterwards, a part of the surface of the 3d grid is exported as the used 2d grid. This is implemented in the DUNE interface layer which now allows to conveniently read a 2 dimensional grid from the user and import it as an internal 3 dimensional grid and export the 2d grid back to the user.

A benefit of this approach is that the distribution and load-balancing into different partitions do not need much additional implementation effort since they are implemented within the topological part of DUNE-ALUGRID. We can reuse all existing element structures, like hierarchical and neighbourhood iterators, because the above identification keeps the neighbourhood information intact.

A drawback of this approach is the additional memory overhead in the new implementation, since for each 2d element one 3d element is stored. But it turns out that the resulting implementation is actually slightly faster than

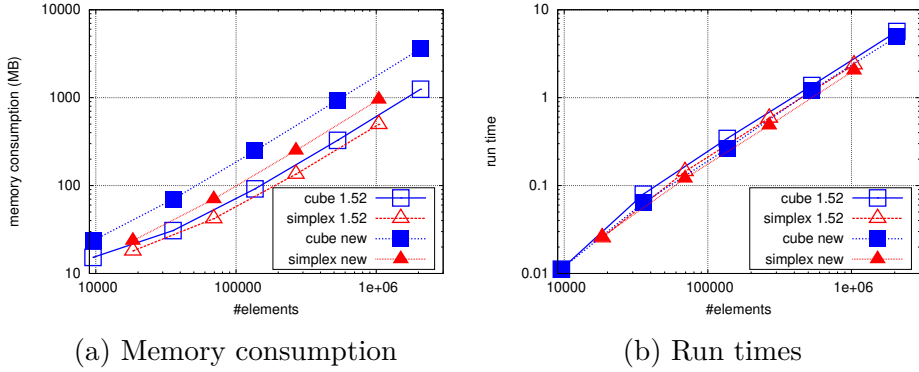


Figure 5.1: Comparison of memory usage and run times for the 2d version in the old and new implementation

the old serial implementation and the number of code lines have been reduced drastically.

Figure 5.1 is taken from [ADKN16, Figure 3] and displays the memory overhead and run times of the new 2d implementation in contrast to the previous serial (ALUGRID Version 1.52) implementation. As one can see in Figure 5.1a the new implementation has the expected memory overhead. Figure 5.1b shows that the new implementation is actually slightly faster than the old one. For more details we refer the reader to [ADKN16].

### 5.1.1 Cubes and Quadrilaterals

First, we discuss the extension of planar quadrilaterals. We choose to extend each quadrilateral as a cube into the third dimension. We use the following notation. We assume to have a 2 dimensional macro grid  $\mathcal{T}_{2d}$  within  $\mathbb{R}^2$  or  $\mathbb{R}^3$ , from which we create a 3 dimensional grid  $\mathcal{T}_{3d}$ .

Consider the planar 2 dimensional cube grid  $\mathcal{T}_{2d}$ . For each vertex  $\mathcal{V}_{2d} \ni v = (x, y)$  we insert a vertex  $v^{real}$  at  $(x, y, 0)$  and a vertex  $v^{fake}$  at  $(x, y, 1)$  into  $\mathcal{V}_{3d}$ . These are connected by an edge  $E_v$ , which is inserted into  $\mathcal{E}_{3d}$ . We identify  $v^{real} = v$ .

For each of the edges  $\mathcal{E}_{2d} \ni E = \overline{v_i v_j}$  the face  $F_E = \text{conv}\{v_i, v_j, v_i^{fake}, v_j^{fake}\}$

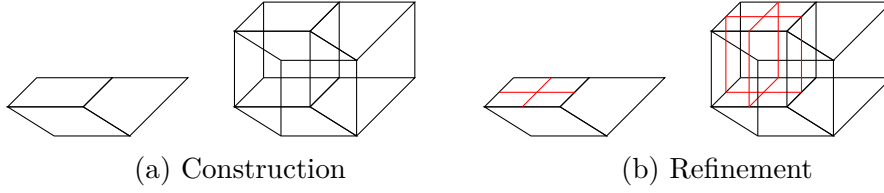


Figure 5.2: Construction and refinement of the 3d cube extension.

is inserted into  $\mathcal{F}_{3d}$ . We identify  $F_E \in \mathcal{T}_{3d}$  with  $E \in \mathcal{T}_{2d}$ . We also insert the two edges  $\mathcal{E}_{3d} \ni E^{real} = E$  and  $E^{fake} = \overline{v_i^{fake}, v_j^{fake}}$  into  $\mathcal{E}_{3d}$ .

For each quadrilateral  $\mathcal{T}_{2d} \ni T_{2d} = \text{conv}\{v_i, v_j, v_k, v_l\}$  we insert the element  $\mathcal{T}_{3d} \ni T_{3d} = \text{conv}\{v_i, v_j, v_k, v_l, v_i^{fake}, v_j^{fake}, v_k^{fake}, v_l^{fake}\}$ . We identify  $T_{2d}$  with  $T_{3d}$ . We also insert the two faces  $F^{real} = T_{2d}$  and  $F^{fake} = T_{2d}^{fake}$ .

Figure 5.2a illustrates this construction. It displays three quadrilaterals within the  $z = 0$  plane, that are extended as cubes into the third dimension by the above description. For each of the 7 vertices the 3d grid contains a bottom vertex, one top vertex and the connecting edge. For each of the 9 edges the 3d grid contains one bottom edge, one top edge and the connecting face. For each of the 3 elements (quadrilaterals) the 3d grid contains one bottom face (quadrilateral), one top face and the connecting element (cube).

The above construction leads to the following cardinality relations:

$$\begin{aligned}
 \#\mathcal{V}_{3d} &= 2\#\mathcal{V}_{2d} \\
 \#\mathcal{E}_{3d} &= 2\#\mathcal{E}_{2d} + \#\mathcal{V}_{2d} \\
 \#\mathcal{F}_{3d} &= 2\#\mathcal{T}_{2d} + \#\mathcal{E}_{2d} \\
 \#\mathcal{T}_{3d} &= \#\mathcal{T}_{2d}
 \end{aligned}$$

The total number of objects within the grid has tripled, which is the expected memory overhead in comparison to a pure 2d implementation. Note that we identify a 2d vertex  $v$  with the 3d vertex  $v^{real}$  instead of identifying it with  $E_v$ . So we identify 0-dimensional objects with 0-dimensional objects,

but for all other dimensions we choose to identify the object with an object of one dimension higher.

To implement isometric 2d refinement we implemented anisotropic refinement in  $z$  direction (cf. Figure 5.2b). The use of standard isotropic 3d refinement introduces too many additional elements. The anisotropic refinement is equivalent to isometric refinement in  $\mathcal{T}_{2d}$  and its extension into the third dimension by the above construction. The resulting sequence of 3d grids is obviously not shape regular. This does not pose a problem, as we solely export the 2d grid, which is refined by standard isometric refinement. The above cardinality relations are kept intact by refinement.

### 5.1.2 Tetrahedrons and Triangles

In the very same fashion the extension of triangles by tetrahedrons is realised.

We consider the planar 2 dimensional simplex grid  $\mathcal{T}_{2d}$ . For each vertex  $\mathcal{V}_{2d} \ni v = (x, y)$  we insert a vertex  $v^{real}$  at  $(x, y, 0)$  into  $\mathcal{V}_{3d}$ . We identify  $v^{real} = v$ . We additionally insert a single vertex  $v^{origin} = (0, 0, 1)$ , so  $\mathcal{V}_{3d} = \mathcal{V}_{2d} \cup \{v^{origin}\}$  and we insert the edge  $E_v = \overline{vv^{origin}}$  into  $\mathcal{E}_{3d}$ .

For each edge  $\mathcal{E}_{2d} \ni E = \overline{v_i v_j}$  we insert the face  $F_E = \text{conv}\{v_i, v_j, v^{origin}\}$  into  $\mathcal{F}_{3d}$ . We identify  $F_E \in \text{grid}_{3d}$  with  $E \in \mathcal{T}_{2d}$ . We also insert the edge  $\mathcal{E}_{3d} \ni E^{real} = E$  into  $\mathcal{E}_{3d}$ .

For each triangle  $\mathcal{T}_{2d} \ni T_{2d} = \text{conv}\{v_i, v_j, v_k\}$  we insert the element  $\mathcal{T}_{3d} \ni T_{3d} = \text{conv}\{v_i, v_j, v_k, v^{origin}\}$ . We identify  $T_{2d}$  with  $T_{3d}$ . We also insert the face  $F^{real} = T_{2d}$  into  $\mathcal{F}_{3d}$ .

Figure 5.3a illustrates this construction. It displays four triangles within the  $z = 0$  plane, that are extended as simplices into the third dimension by the above description. First the extra origin vertex is included. For each of the 6 vertices the 3d grid contains a bottom vertex and the connecting edge to the origin. For each of the 9 edges the 3d grid contains one bottom

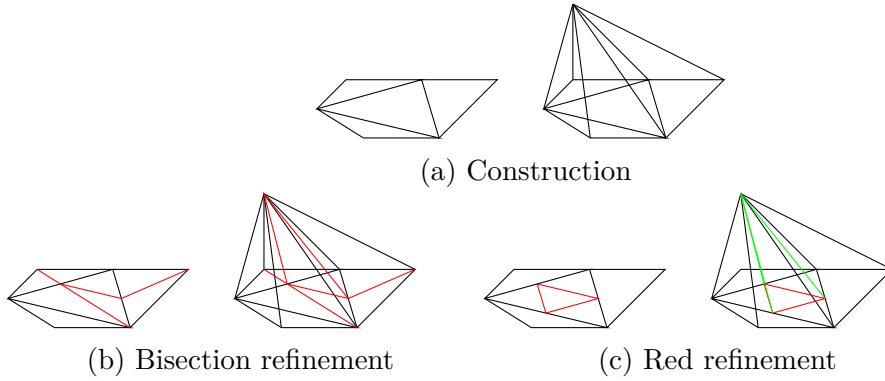


Figure 5.3: Construction and Refinement of the 3d simplex extension.

edge, and the face formed by edge and origin. For each of the 4 elements (2-simplices) the 3d grid contains one bottom face (2-simplex) and the element (3-simplex) formed by origin and face.

The above construction leads to the following cardinality relations:

$$\begin{aligned}
 \#\mathcal{V}_{3d} &= \#\mathcal{V}_{2d} + 1 \\
 \#\mathcal{E}_{3d} &= \#\mathcal{E}_{2d} + \#\mathcal{V}_{2d} \\
 \#\mathcal{F}_{3d} &= \#\mathcal{T}_{2d} + \#\mathcal{E}_{2d} \\
 \#\mathcal{T}_{3d} &= \#\mathcal{T}_{2d}
 \end{aligned}$$

The total number of objects within the grid has doubled, which is the expected memory overhead in comparison to a pure 2d implementation. Note that again we identify a 2d vertex  $v$  with the 3d vertex  $v^{real}$  instead of identifying it with  $E_v$ .

To implement isometric 2d refinement we implemented the green closure in  $z$  direction (cf. Section 2.4). This is depicted in Figure 5.3c. The standard isometric 2d refinement is shown in red and the artificial green closure uses green lines.

NVB on tetrahedrons had already been implemented in DUNE-ALUGRID. We have adjusted the choice of refinement edge such that no edge containing  $v^{origin}$  was refined. Also the choice has been adjusted to actually fit the

2d NVB selection. This new implementation now also allows to implement longest edge bisection by adjusting the choice of refinement edge.

As before the 3d grid is not shape regular, but the 2d grid is for both refinement strategies. As the refinement is equivalent to refine the 2d grid and construct the extension afterwards, the above cardinality relations are kept intact by refinement.

Note that  $v^{origin}$  is inserted first into the grid and never moves, so we can always easily locate it as it has index 0. In the case of communication across vertices in a distributed grid we exclude  $v^{origin}$ , as it connects all partitions and is not part of the actually used 2d grid.

### 5.1.3 Surface 2d Grids

We apply the above ideas to surface grid. This is straightforward for simplices since the location of  $v^{origin}$  is negligible, see Figure 5.5.

In the case of cube meshes for each vertex  $v = (x, y, z)$  we insert the vertex  $v^{real} = v$  and the vertex  $v^{fake} = (x, y, z + 1)$  and  $E_v$  as above. The rest of the grid construction stays as before, see Figure 5.4.

The following problems arise for surfaces:

1. Two elements  $T_i, T_j \in \mathcal{T}_{3d}$  may intersect.
2. A face  $F \in \mathcal{T}_{3d}$  can have two neighbours on one side and no neighbour on the other.
3. Two vertices  $v, \tilde{v}$  can have the same coordinates.
4. The volume of an element  $T \in \mathcal{T}_{3d}$  can be 0.
5. It is currently not possible to construct non-orientable surfaces.

Problems 1-4 arise, because 3 dimensional grid implementations sometimes depend on some of these properties implicitly within their implementation. Fortunately within ALUGRID the topological and geometric concepts are separated so that this extension is possible, as all these problems

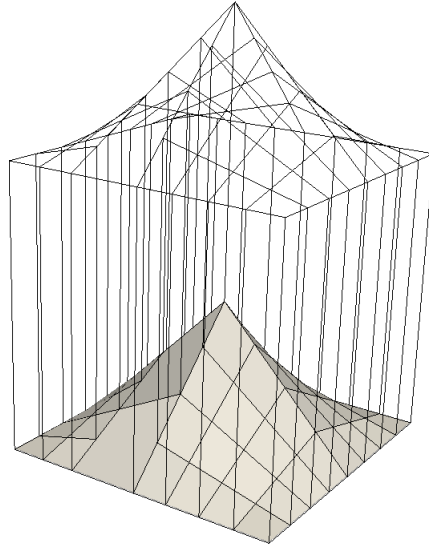


Figure 5.4: 3d surface cube extension of an example grid

are merely geometric. Problem 5 already was a limitation of `ALUGRID` beforehand.

## 5.2 Dune-ACFem - Adaptive Convenient Finite Elements

This introduction into `DUNE-ACFEM` is taken from [AL17]. It explains basic concepts of `DUNE-ACFEM`, that will be used in Section 6.2, where the implementation of our case study is presented.

The `DUNE` extension module `DUNE-ACFEM` [Hei14] is a simulation framework that relies on `DUNE-FEM` [DKNO10] to provide finite-element spaces on generic grids and the corresponding utilities to construct finite-element and other discretization schemes. Examples are given in the `DUNE-FEM-HOWTO`. For more information consult [DKNO10].

The add-on module `DUNE-ACFEM` provides expression templates (and



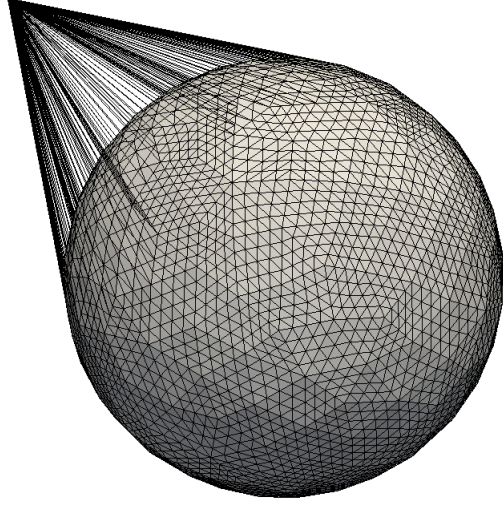


Figure 5.5: 3d surface simplex extension of a sphere

also analytical functions that can be evaluated on the mesh) for the discrete functions of DUNE-FEM and also for models similar to but more elaborate than those in the DUNE-FEM-HOWTO. This aims at simplifying the algorithmic formulation of elliptic and parabolic PDEs.

The expression templates for discrete functions allow for addition, multiplication with scalars, multiplication with scalar functions, scalar products of the components of two functions, and unary expressions like componentwise  $\sin$ ,  $\cos$ ,  $\sqrt{\phantom{x}}$ ,  $\exp$ .

DUNE-ACFEM is designed to solve 2nd order elliptic PDEs of the form

$$\begin{aligned}
 -\nabla \cdot (A(x, u, \nabla u) \nabla u) + \nabla \cdot (b(x, u, \nabla u) u) + c(x, u, \nabla u) u &= f(x) && \text{in } \Omega, \\
 u &= g_D && \text{on } \Gamma_D, \\
 (A(x, u, \nabla u) \nabla u) \cdot \nu + \alpha(x, u) u &= g_N && \text{on } \Gamma_R, \\
 (A(x, u, \nabla u) \nabla u) \cdot \nu &= g_N && \text{on } \Gamma_N,
 \end{aligned}
 \tag{5.1}$$

or, in weak formulation,

$$\begin{aligned} \int_{\Omega} (A \nabla u) \cdot \nabla \phi \, dx + \int_{\Omega} (\nabla \cdot (b u) + c u) \phi \, dx - \int_{\Omega} f(x) \phi \, dx \\ + \int_{\Gamma_R} \alpha u \phi \, do - \int_{\Gamma_N \cup \Gamma_R} g_N \phi \, do = 0, \end{aligned} \quad (5.2)$$

$$\langle \Pi, u \rangle = \langle \Pi, g_D \rangle.$$

Possible Dirichlet data is enforced by standard Lagrange test-functions  $\Pi$ . The multiplication with the test-functions  $\phi$  of the chosen discretization space, the integral and the quadrature are provided by DUNE-FEM. The integrand has to be provided by the model, which is the central concept of DUNE-ACFEM. A model is basically a tuple of the form

$$\mathcal{M} := (\sigma_T, \mu_T, \rho_I, g_N, g_D, w_D, f, \chi_R, \chi_N, \chi_D, \Psi)$$

Using such a model results in the following discrete weak formulation

$$\begin{aligned} \sum_{T \in \mathcal{T}} \left( \int_T (\sigma_T(x, U, \nabla U) : \nabla V + \mu_T(x, U, \nabla U) \cdot V - f \cdot V) \right. \\ \left. + \int_{I \in T \cap \partial \Omega} ((\chi_R(x) \rho_I(x, U) - \chi_N(x) g_N(x)) \cdot V) \right) - \langle \Psi, V \rangle = 0, \\ \langle \Pi, \chi_D U \rangle = \langle \Pi, \chi_D g_D \rangle. \end{aligned} \quad (5.3)$$

with the following constituents:

Flux	$\sigma_T : \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^{m \times d},$
Source	$\mu_T : \mathbb{R}^d \times \mathbb{R}^m \times \mathbb{R}^{m \times d} \rightarrow \mathbb{R}^m,$
Robin-flux	$\rho_I : \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}^m,$
Robin-indicator	$\chi_R : \partial \Omega \rightarrow \{0, 1\},$
Neumann-data	$g_N : \mathbb{R}^d \rightarrow \mathbb{R}^m,$
Neumann-indicator	$\chi_N : \partial \Omega \rightarrow \{0, 1\},$
Dirichlet-data	$g_D : \mathbb{R}^d \rightarrow \mathbb{R}^m,$
Dirichlet-indicator	$\chi_D : \partial \Omega \rightarrow \{0, 1\},$
Bulk-forces	$f : \mathbb{R}^d \rightarrow \mathbb{R}^m,$
Force-functional	$\Psi \in \mathbb{V}^*.$

This is directly reflected in code, as a model is derived from an interface class that requires exactly the implementation of the above constituents (and the linearisation of flux, source and robin-flux for non-linear models). Only non-zero contributions need to be implemented.

DUNE-ACFEM allows forming algebraic expressions from existing models. The generated model-expressions fulfill the model-interface just as "hand-coded" ones. This allows forming of complicated models from a set of convenient pre-built base-models. For a list of predefined models consult [Hei14]. The algebraic expressions include linear combinations of models and multiplying with  $L^\infty$ -functions. The only exception are models containing force-functionals  $\Psi$ . These can only be added and also only to models that do not have a functional themselves.

The `EllipticOperator` is another important concept. It applies and/or assembles the discretization of equation (5.2). The `FemScheme` then solves the given problem and chooses the necessary linear or non-linear solvers, depending on structural information given by the model. DUNE-ACFEM can use any solver (and preconditioner) provided by DUNE-FEM, which in turn includes all solvers and preconditioners provided by DUNE-ISTL. Also, as DUNE-FEM interfaces PETSC, any linear solver and preconditioner from PETSC can be applied.

This introduction has been written for the DUNE-ACFEM release 2.4 version. While most of this text is still valid in the current DUNE-ACFEM version, functionals can no longer be added to models, but now need to be passed to the `FemScheme` directly, as the addition of functionals is not straight-forward and causes a lot of trouble. This has not been updated, because we use the old feature in Section 6.2.

### 5.3 $n$ Dimensional Simplex Grid

As a proof of concept, we have developed a simple  $n$ -dimensional simplex grid implementation to experiment with and confirm the weak compatibility.

It has been used for the experiment in Section 3.4.6. The implementation was written in C++ and consists of three main components:

- A `std::vector` of macro elements,
- a `std::unordered_map` of edges to store the connectivity and
- a `std::vector` of vertices to store the coordinates.

A vertex is just an index with an attached coordinate and a level. A macro element  $T_0$  derives from a simplex class and from a hierarchy class. The simplex class provides the list of sorted vertices and the refinement methods. The hierarchy class provides the binary tree  $\mathfrak{F}(T_0)$  and methods to iterate over it.

An edge is more complex. It stores the indices of its two vertices, optionally a vertex index at its center, a level and the connectivity information, which provides access to all adjacent elements. If there is a center vertex index, then the edge counts as refined.

For refinement purposes there are two algorithms, which have been inspired by the implementation in DUNE-ALUGRID. In these algorithms we use a bucket list, that sorts edges into buckets by their level, which we call refinement queue  $\mathcal{RQ}$ . This idea evolved in discussions with Stephan Hilb. The idea is to refine edges of a lower level first. As Lemma 3.39 asserts that refining all edges of a certain level leads to a conforming situation, this situation needs to be reached before refining further. So the bucket-list guarantees that the adjacency of edges is correct, when it is supposed to be refined in Algorithm 5.2.

First we introduce Algorithm 5.1 to show how the data structures need to be updated during element refinement. The term newly created edges means edges, that are present in children of the element, but were not present in the element itself. Updating only the adjacency of these edges leads to a small connectivity size. The connectivity storage for each edge contains only

---

**Algorithm 5.1:** Refine an Element

---

**Data:** Simplex  $T$ , refinement queue  $\mathcal{RQ}$ , optional edge  $E$

```
1 if  $E$  argument present AND NOT  $E \subset T$  then
2   | Return
3 if  $T$  has been refined then
4   | Recursively call Refine( $T_i, \mathcal{RQ}, E$ ) for each child  $T_0, T_1$ 
5 Get refinement edge  $E$  of  $T$ 
6 if  $E$  has been refined then
7   | get vertex index from edge data structure
8 else
9   | create new vertex, add it to the edge
10  | add  $E$  to  $\mathcal{RQ}$ .
11 refine the element
12 for each newly created edge  $E_{new}$  do
13   | if NOT  $E_{new}$  exists then
14     | Insert  $E_{new}$  into the grid
15   | update list of adjacent elements for  $E_{new}$ 
```

---

the elements of lowest level, which contain the edge, or, in other words, the elements that created this edge.

Algorithm 5.2 is our adaptation of the full refinement algorithm (Algorithm 2.1). It takes as input the set of elements marked for refinement. It is also possible to take a set of edges as marked for refinement. These are then directly added to the refinement queue and the first part (until line 5) is skipped. This allows to implement edge-based indicators (e.g., refine the longest edge in the grid).

The run time of Algorithm 5.2 is depicted in Table 5.1. It displays the time in seconds needed to create the final situation in the experiment from Section 3.4.6 which includes a first global stage of refinement and afterwards local refinement with conforming closure on a Kuhn-cube. The timing includes both the global refinement and the local adaptation step. It does not include grid creation and iterating over the grid to print the size. Also dimension 8, which is present in Section 3.4.6 is excluded since the local

---

**Algorithm 5.2:** Grid refinement

---

**Data:** Marked elements  $\mathcal{M}$

```
1 Create empty refinement queue  $\mathcal{RQ}$ 
2 for  $T \in \mathcal{M}$  do
3   | //Algorithm 5.1
4   | Refine ( $T, \mathcal{RQ}$ )
5 while  $\mathcal{RQ}$  not empty do
6   | Get first  $E$  from  $\mathcal{RQ}$ 
7   | for each element  $T$  adjacent to  $E$  do
8     | //Algorithm 5.1
9     | Refine ( $T, \mathcal{RQ}, E$ )
10  | Remove  $E$  from  $\mathcal{RQ}$ 
```

---

Dimension	2	3	4	5	6	7
Final (Strong)	16	109	951	12968	191452	3467190
Final (Weak)	15	108	1004	11400	152760	2358720
Time (Strong)	0.00076	0.00133	0.01545	0.21723	3.5896	76.5403
Time (Weak)	0.00065	0.00127	0.01480	0.18885	2.8436	52.2202

Table 5.1: #Elements and run times (in s) of the Experiment from Section 3.4.6

adaptation has not finished due to memory problems. We also display the data points of Table 5.1 in Figure 5.6 which indicates that the grid refinement algorithm scales linearly in the final number of elements  $n$ , which is the optimal achievable rate, as each element that is created has to be touched.

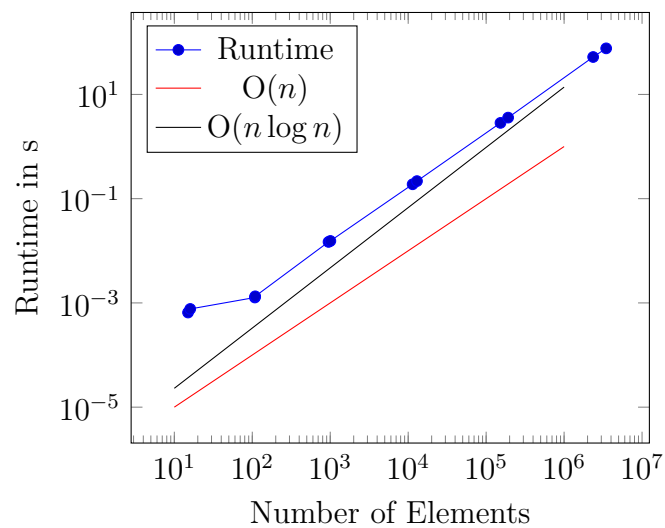


Figure 5.6: Run time of Algorithm 5.2 for different final grid sizes.

## Chapter 6

# Case Study: Using Dune-ACFem for Non-smooth Minimization of Bounded Variation Functions

We demonstrate the ease of implementation with DUNE-ACFEM and the usability of the new 2d grid implementation within DUNE-ALUGRID with a complex image processing application. This is done by minimizing a non-smooth functional consisting of a combined  $L^1/L^2$ -data fidelity term and a total variation term for regularisation. Such an optimization problem has been shown to effectively remove Gaussian and salt-and-pepper noise simultaneously, see [HL13, Lan17a]. In order to compute an approximate solution we use the primal-dual algorithm proposed in [CP11], which requires a saddle point formulation of the problem. For the numerical implementation we discretize using various finite-element spaces defined over locally refined conforming grids. Similar to the works [Bar12, Bar15b, Bar15a, HRC14], where the considered functional is composed solely of an  $L^2$ -data term and a total variation term, we refine the grid adaptively using an a priori criterion.

We use the new 2d grid implementation of DUNE-ALUGRID (cf. Section 5.1) which provides a distributed grid with local refinement capabilities. In



combination with DUNE-ACFEM (cf. Section 5.2) the resulting algorithm is automatically parallelized. The advantage of using DUNE-ACFEM is that it allows us to conveniently express the given algorithm in code and use all of the underlying parallel infrastructure of DUNE effortlessly from the application perspective. This includes (non)-linear solvers, discrete functions (vectors), (linear) operators (system matrices) and interpolation operators.

In Section 6.1 we formulate the continuous and the discrete problem with the respective discrete spaces. In particular for a certain discretization we state that the discrete problem converges to the continuous one as the mesh-size goes to zero. In Section 6.2 we discuss how DUNE-ACFEM is used to implement the primal-dual algorithm. Numerical examples showing the applicability of our proposed implementation and experiments testing different discretizations are presented in Section 6.3. This chapter follows closely the article [AL17] published in *Archive of Numerical Software* by the author and Andreas Langer.

## 6.1 Problem Formulation

**Motivation** The intended application is noise removal on a grey-scale image. We model grey-scale images by scalar-valued  $L^2$ -functions, as they may contain jumps between different objects within the image. To use a coloured image the algorithm would have to be extended to vector-valued functions.

We assume that an original image  $\hat{g}$  has been corrupted by noise. The resulting (noisy) image is called  $g$ , that is

$$g = \hat{g} + \rho$$

where  $\rho$  represents additive noise. The type of additive noise is usually named after the underlying distribution, e.g., Gaussian noise, Poisson noise, Rician noise. In the most general case noise can be applied as a function  $f$ :

$$g = f(\hat{g})$$

An important non-linear example of this type of noise is the so-called salt-and-pepper noise  $S$  with noise level  $s \in [0, 1]$ , which corrupts an original image  $\hat{g}$  by

$$S\hat{g}(x) = \begin{cases} \min \hat{g} & \text{with probability } s/2 \\ \max \hat{g} & \text{with probability } s/2 \\ \hat{g}(x) & \text{with probability } 1 - s \end{cases} .$$

It is known, that minimizing the total variation tends to concentrate the variation at discontinuities of images. This leads to a better resolution of edges within the image. In combination with the total variation fidelity terms of the form  $\|u - g\|$  remove noise. Choosing the  $L^2$  norm removes Gaussian noise quite well [Cha04], while salt-and-pepper noise is effectively removed by an  $L^1$  fidelity term [Nik04]. The combination of both fidelity terms has been shown to remove a combination of these noise types [HL13].

**Problem** We aim to find a denoised approximation  $u$  of the image (or datum)  $g$  by minimizing the following primal problem:

$$\min_{v \in BV(\Omega) \cap L^2(\Omega)} J_{\alpha_1, \alpha_2}(v) := \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|_{L^2(\Omega)}^2 + |Dv|(\Omega) \quad (6.1)$$

There,  $\Omega \subset \mathbb{R}^2$ , is an open, bounded and polygonal (usually rectangular) set with Lipschitz boundary,  $g \in L^2(\Omega)$  is a given datum,  $\alpha_i \geq 0$  for  $i = 1, 2$  with  $\alpha_1 + \alpha_2 > 0$  are two adjustable weighting parameters and  $BV(\Omega) \subset L^1(\Omega)$  denotes the space of functions with bounded variation. That is,  $v \in BV(\Omega)$  if and only if

$$|Dv|(\Omega) := \sup \left\{ \int_{\Omega} v \operatorname{div} \phi dx, \phi \in C_0^\infty(\Omega, \mathbb{R}^2), \|\phi\|_{C^0(\Omega)} \leq 1 \right\} \quad (6.2)$$

is finite, see [AFP00, Giu84]. The space  $BV(\Omega)$  endowed with the norm  $\|v\|_{BV(\Omega)} := \|v\|_{L^1(\Omega)} + |Dv|(\Omega)$  is a Banach space [Giu84]. For  $\alpha_2 > 0$  the minimization problem (6.1) admits a unique solution owing to the strict convexity of the quadratic term [Lan17b].  $BV$  is the natural space for posing the problem, as we expect discontinuities at edges of objects within the image.

Also images typically contain few objects and hence we expect the total variation within the image to be bounded.

By the definition of the total variation (6.2) the equivalent saddle-point (or primal-dual) formulation of (6.1) reads

$$\min_{v \in BV(\Omega) \cap L^2(\Omega)} \max_{\mathbf{p} \in C_0^\infty(\Omega, \mathbb{R}^2), |\mathbf{p}|_2 \leq 1} \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|_{L^2(\Omega)}^2 + \int_{\Omega} v \operatorname{div} \mathbf{p}. \quad (6.3)$$

This primal-dual formulation will be the starting point of our algorithm. It does not have a unique solution in general, even if the primal problem (6.1) is strictly convex ( $\alpha_2 > 0$ ). This is due to the fact, that the dual problem of (6.1), even for  $\alpha_1 = 0$  and  $\alpha_2 > 0$ , is only convex but not strictly convex, see for example [HK04, HS06]. So the solution  $(u^*, \mathbf{p}^*)$  is unique in the primal variable  $u^*$ , but there may be several optimal  $\mathbf{p}^*$  possible.

### 6.1.1 Discretization

As we later intend to exploit the local refinement capabilities of our grid we use finite elements to discretize the problem, which has been shown to work in [Bar15b]. We define the following finite element spaces given a triangulation  $\mathcal{T}$  of a bounded polygonal domain  $\Omega \subset \mathbb{R}^2$ :

$$\begin{aligned} \mathcal{L}^0(\mathcal{T}) &= \{q_h \in L^1(\Omega) : q_h|_T \text{ is constant for each } T \in \mathcal{T}\} \\ \mathcal{S}^1(\mathcal{T}) &= \{v_h \in C(\bar{\Omega}) : v_h|_T \text{ is affine for each } T \in \mathcal{T}\}. \end{aligned}$$

For the vector-valued versions, we write  $\mathcal{L}^0(\mathcal{T})^2$  and  $\mathcal{S}^1(\mathcal{T})^2$ , respectively, and boundary conditions are denoted via a subscript. In particular we use the subscript 0 for zero boundary values and the subscript  $N$  for zero Neumann boundary values in normal direction, e.g.,  $\mathcal{S}_0^1(\mathcal{T}) := \{v_h \in \mathcal{S}^1(\mathcal{T}) : v_h = 0 \text{ on } \partial\Omega\}$  and  $\mathcal{L}_N^0(\mathcal{T})^2 := \{\mathbf{q}_h \in \mathcal{L}^0(\mathcal{T})^2 : \mathbf{q}_h \cdot \mathbf{n} = 0 \text{ on } \partial\Omega\}$  where  $\mathbf{n}$  is the unit vector in normal direction. We will also need the space of piecewise constant functions defined on the skeleton  $\mathcal{T}^1 = \mathcal{F}$ , which can only be

evaluated on the faces:

$$\mathcal{L}^0(\mathcal{F}) := \{q_h \in L^1(\mathcal{T}^1) : q_h|_F \text{ is constant for each } F \in \mathcal{F}\}$$

We need this space  $\mathcal{L}^0(\mathcal{F})$  because the variation of elementwise constant functions is located solely at the faces. We will see, that this is the natural dual space for  $\mathcal{L}^0(\mathcal{T})$  in our setting.

We state the following convergence result for  $u \in \mathcal{S}^1(\mathcal{T})$ .

**Theorem 6.1.** *Let  $u_h^* \in \arg \min_{v \in \mathcal{S}^1(\mathcal{T})} J_{\alpha_1, \alpha_2}(v)$  and  $u^* \in BV(\Omega) \cap L^2(\Omega)$  be a minimizer of the function  $J_{\alpha_1, \alpha_2}$ . Then we have that  $J_{\alpha_1, \alpha_2}(u_h^*) \rightarrow J_{\alpha_1, \alpha_2}(u^*)$  as  $h \rightarrow 0$ . If additionally  $\alpha_2 > 0$ , then  $u_h^* \rightarrow u^*$  in  $L^2(\Omega)$  as  $h \rightarrow 0$ .*

We refer the reader to [AL17] for the proof.

In the following we discretize using finite element spaces  $\mathbb{U}$  (e.g.  $\mathcal{S}^1(\mathcal{T})$  or  $\mathcal{L}^0(\mathcal{T})$ ) and  $\mathbb{P}$  (e.g.  $\mathcal{S}_0^1(\mathcal{T})^2$  or  $\mathcal{L}_N^0(\mathcal{T})^2$ ) for the primal variable  $u$  and the dual variable  $\mathbf{p}$ , respectively.

As  $d = 2$ , we know that  $\mathcal{S}^1(\mathcal{T}) \subset H^1(\Omega)$ , hence the gradient operator  $\nabla : \mathcal{S}^1(\mathcal{T}) \mapsto \mathcal{L}^0(\mathcal{T})^2$  and the divergence operator  $\operatorname{div} : \mathcal{S}^1(\mathcal{T})^2 \mapsto \mathcal{L}^0(\mathcal{T})$  are well-defined. In the case that  $\nabla v \notin L^2(\Omega)$  for  $v \in \mathbb{U}$  or  $\operatorname{div} \mathbf{q} \notin L^2(\Omega)$  for  $\mathbf{q} \in \mathbb{P}$  we use the following identities to represent the action of the gradient operator  $\nabla$  and the divergence operator  $\operatorname{div}$

$$\begin{aligned} \mathbb{U} = \mathcal{L}^0(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{F})^2 : \quad \langle \nabla v, \mathbf{q} \rangle &:= - \sum_{F \in \mathcal{F}} |F| \mathbf{p}|_F [v]_F =: - \langle v, \operatorname{div} \mathbf{q} \rangle, \\ \mathbb{U} = \mathcal{L}^0(\mathcal{T}), \mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2 : \quad \langle \nabla v, \mathbf{q} \rangle &:= - \langle v, \operatorname{div} \mathbf{q} \rangle, \\ \mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2 : \quad - \langle v, \operatorname{div} \mathbf{q} \rangle &:= \langle \nabla v, \mathbf{q} \rangle. \end{aligned} \tag{6.4}$$

A saddle-point formulation also exists for finite element spaces [Bar12]. We show that natural pairings are  $\mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$  and  $\mathbb{U} = \mathcal{L}^0(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{F})^2$ .

For  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  we have that

$$|Dv|(\Omega) = \sup_{\mathbf{p} \in \mathcal{L}_N^0(\mathcal{T})^2, |\mathbf{p}|_2 \leq 1} \int_{\Omega} \nabla v \cdot \mathbf{p} \, dx, \quad (6.5)$$

leading to

$$\begin{aligned} \inf_{v \in \mathcal{S}^1(\mathcal{T})} J_{\alpha_1, \alpha_2}(v) &= \inf_{v \in \mathcal{S}^1(\mathcal{T})} \sup_{\mathbf{p} \in \mathcal{L}_N^0(\mathcal{T})^2} \int_{\Omega} \nabla v \cdot \mathbf{p} \, dx \\ &\quad + \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|_{L^2(\Omega)}^2 - I_{\mathcal{K}}(\mathbf{p}), \end{aligned}$$

where

$$I_{\mathcal{K}}(\mathbf{p}) := \begin{cases} 0 & \text{if } \mathbf{p} \in \mathcal{K} \\ \infty & \text{else} \end{cases}$$

is the indicator function of  $\mathcal{K} := \{\mathbf{p} \in L^1(\Omega)^2 : |\mathbf{p}|_2 \leq 1\}$ .

For  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$  we obtain

$$|Dv|(\Omega) = \sup_{\mathbf{p} \in \mathcal{L}_N^0(\mathcal{F})^2, |\mathbf{p}_F|_2 \leq 1} \sum_{F \in \mathcal{F}} |F| \mathbf{p}_F [v]_F, \quad (6.6)$$

where  $[v]_F \in \mathbb{R}$  defines the jump across  $F \in \mathcal{F}$  in normal direction. Hence in this case the discrete problem reads

$$\begin{aligned} \inf_{v \in \mathcal{L}^0(\mathcal{T})} J_{\alpha_1, \alpha_2}(v) &= \inf_{v \in \mathcal{L}^0(\mathcal{T})} \sup_{\mathbf{p} \in \mathcal{L}_N^0(\mathcal{F})^2} \sum_{F \in \mathcal{F}} |F| \mathbf{p}_F [v]_F \\ &\quad + \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|_{L^2(\Omega)}^2 - I_{\mathcal{K}}(\mathbf{p}). \end{aligned}$$

While this formulation is equivalent to the primal formulation (6.1), in general we cannot expect convergence to the continuous solution, unless all jumps of the continuous solution are correctly represented by faces within  $\{\mathcal{F}_k\}_{k \in \mathbb{N}_0}$ . For more details we refer the reader to [Bar12, Section 4].

## 6.1.2 Primal-Dual Algorithm

The following Algorithm 6.1 is taken from Chambolle and Pock [CP11, Algorithm 1] and solves the general saddle point problem

$$\min_{u \in \mathbb{U}} \max_{\mathbf{p} \in \mathbb{P}} \langle K u, \mathbf{p} \rangle + G(u) - F^*(\mathbf{p})$$

given two proper, lower semi-continuous functions  $F^* : \mathbb{P} \rightarrow \mathbb{R} \cup \{+\infty\}$ ,  $G : \mathbb{U} \rightarrow \mathbb{R} \cup \{+\infty\}$ , and an operator  $K : \mathbb{U} \rightarrow \mathbb{P}^*$ . The algorithm additionally needs two step sizes  $\theta, \sigma > 0$  and an overrelaxation parameter  $\theta \in [0, 1]$ .

---

**Algorithm 6.1:** Algorithm 1 from [CP11]

---

- 1 Initialization:  $\tau, \sigma > 0, \theta \in [0, 1], (u_0, \mathbf{p}_0) \in \mathbb{U} \times \mathbb{P}$  and  $\bar{u}_0 = u_0$
  - 2  $\mathbf{p}_{k+1} = (I + \sigma \partial F^*)^{-1}(\mathbf{p}_k + \sigma K \bar{u}_k)$
  - 3  $u_{k+1} = (I + \tau \partial G)^{-1}(u_k - \tau K^* \mathbf{p}_{k+1})$
  - 4  $\bar{u}_{k+1} = u_{k+1} + \theta(u_{k+1} - u_k)$
- 

Theorem 1 of [CP11] guarantees convergence of this algorithm to a saddle point  $(u^*, \mathbf{p}^*)$ , if the according discretization of the saddle-point problem (6.3) has a saddle point and additionally  $\theta = 1$  and

$$\sigma \tau \|K\|^2 < 1, \tag{6.7}$$

with the operator norm  $\|K\|$  of  $K$ .

Now we formulate our realisation of this primal-dual algorithm by identifying  $F^* : \mathbb{P} \rightarrow \mathbb{R} \cup \{+\infty\}$ ,  $G : \mathbb{U} \rightarrow \mathbb{R} \cup \{+\infty\}$ , and  $K : \mathbb{U} \rightarrow \mathbb{P}^*$  with

$$F^*(\mathbf{p}) = I_{\mathcal{K}}(\mathbf{p}), \quad G(v) = \alpha_1 \|v - g\|_{L^1(\Omega)} + \alpha_2 \|v - g\|_{L^2(\Omega)}^2,$$

and

$$\langle K v, \mathbf{p} \rangle = \langle \nabla v, \mathbf{p} \rangle.$$

We assume, that the datum  $g \in L^2(\Omega)$  and the cost parameters  $\alpha_i$  are given. We initialize  $u_0 = \bar{u}_0 \in \mathbb{U}$ , e.g.,  $u_0 \equiv 0$  or  $u_0 = P g$ , where

$P : L^2(\Omega) \rightarrow \mathbb{U}$  denotes the  $L^2$ -projection onto the discrete space  $\mathbb{U}$ , and  $\mathbf{p}_0 \in \mathbb{P}$  to be the zero function. Parameters of our algorithm are the step sizes  $\sigma, \tau > 0$ , the coefficient  $\beta = \frac{\tau\alpha_1}{1+2\tau\alpha_2}$ , and the overrelaxation parameter  $\theta \in [0, 1]$ .

Then our primal-dual algorithm iterates starting with  $k = 0$  as follows:

1. The update of  $\mathbf{p}_k$  (Algorithm 6.1, line 2) is realised as follows. Set  $\bar{\mathbf{p}} \in \mathbb{P}$  with step size  $\sigma$  as

$$\langle \bar{\mathbf{p}}, \mathbf{q} \rangle = \langle \mathbf{p}_k + \sigma \nabla \bar{u}_k, \mathbf{q} \rangle \quad \forall \mathbf{q} \in \mathbb{P}. \quad (6.8)$$

As the algorithm is derived from formulation (6.3), we need to guarantee that  $|\mathbf{p}_k|_2 \leq 1$  for all  $k$ . This is done by the following update

$$\mathbf{p}_{k+1} = (I + \sigma \partial F^*)^{-1}(\bar{\mathbf{p}}) \quad \Leftrightarrow \quad \mathbf{p}_{k+1}(x) = \frac{\bar{\mathbf{p}}(x)}{\max(|\bar{\mathbf{p}}(x)|_2, 1)}, \quad (6.9)$$

for all  $x \in \Omega$ . Note, that due to the structure of the operator  $(I + \sigma \partial F^*)^{-1}$  (see [CP11] for more details)  $\mathbf{p}_{k+1} \in \mathbb{P} \cap \mathcal{K}$ .

2. The update of  $u_k$  (Algorithm 6.1, line 3) is realised by

$$u_{k+1}(x) = \begin{cases} z(x) - \beta & \text{if } z(x) - \beta \geq Pg(x) \\ z(x) + \beta & \text{if } z(x) + \beta \leq Pg(x) \\ Pg(x) & \text{else} \end{cases} \quad (6.10)$$

for all  $x \in \Omega$ , where  $z \in \mathbb{U}$  is defined via

$$\langle z, v \rangle = \frac{1}{1 + 2\tau\alpha_2} \left( \langle u_k + 2\tau\alpha_2 g, v \rangle + \tau \langle \operatorname{div} \mathbf{p}_{k+1}, v \rangle \right) \quad \forall v \in \mathbb{U}. \quad (6.11)$$

3. We overrelaxate (Algorithm 6.1, line 4):

$$\bar{u}_{k+1} = u_{k+1} + \theta(u_{k+1} - u_k) \quad (6.12)$$

4. If the stopping criterion

$$\frac{\|u_{k+1} - u_k\|_{L^2}}{\tau} + \frac{\|\mathbf{p}_{k+1} - \mathbf{p}_k\|_{L^2}}{\sigma} < TOL \quad (6.13)$$

holds, we terminate the algorithm, otherwise we set  $k \rightarrow k + 1$  and repeat.

Note that in Equation (6.9) with  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  it is a priori not clear that  $\mathbf{p}_{k+1} \in \mathcal{S}_0^1(\mathcal{T})^2$ , as taking the pointwise maximum of two piecewise linear functions results in a piecewise linear function on a finer grid. So in this case we additionally apply the nodal interpolation operator  $\mathcal{I} : C^0(\Omega)^2 \rightarrow \mathbb{P}$ , which then guarantees  $\|\mathbf{p}_{k+1}|_2\|_{L^\infty} \leq 1$  and  $\mathbf{p}_{k+1} \in \mathbb{P}$ . A similar problem arises for  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  and Equation (6.10), which we treat analogously. Further note, that testing with the complete space  $\mathbb{U}$  or  $\mathbb{P}$  (Eqs. (6.8), (6.11)), respectively, is equivalent to an  $L^2$ -projection onto the respective space.

As stated before, Theorem 1 of [CP11] guarantees convergence of this algorithm to a saddle point, if the according discretization of problem (6.3) has a saddle point and additionally  $\theta = 1$  and

$$\sigma\tau\|\nabla\|^2 < 1, \quad (6.14)$$

with the operator norm of the gradient operator on the discrete space  $\mathbb{U}$ . This norm is bounded for discrete functions and of order  $O(1/h_{\min})$ , where  $h_{\min} := \min_{T \in \mathcal{T}} \text{diam}(T)$ , which implies that using finer meshes will always result in more steps of the primal-dual algorithm. Even using an adaptive mesh is not beneficial to the stepsize, as  $\|\nabla\|$  depends on the minimum mesh-width. If there exists a saddle point, i.e., a continuous solution, we can choose the step sizes  $\sigma, \tau$  small enough according to the grid size  $h_{\min}$  that the algorithm converges.



## 6.2 Implementation Details

The algorithm is implemented in the DUNE-project NSM. This project is compatible with the 2.4-release and available on the website <http://www.ians.uni-stuttgart.de/nmh/downloads>. It requires the modules DUNE-ACFEM (compare Section 5.2) and all required modules, available at [gitlab.dune-project.org](http://gitlab.dune-project.org). In this section we show how DUNE-ACFEM is used to tackle this problem.

### 6.2.1 Image Read-In and Noise Simulation

To read-in images we use the CIMG Library - C++ Template Image Processing Toolkit [Tsc]. The single header file library CIMG is an open source project to provide easy handling and processing of images. It is capable of reading-in standard image formats (e.g. `.png`, `.tif`, `.jpg`) and of adding certain types of noise. Each pixel is accessible by its coordinates utilizing the data type `image`, whose constructor gets the filename containing the image location. Moreover, `image` provides a method to apply noise to the data with a given noise level for different noise types. In particular, Gaussian noise with noise level (variance)  $\gamma$  and mean 0 and salt-and-pepper noise  $S$  with noise level  $s \in [0, 1]$  are available.

We define three function classes within the file `src/datafunction.hh` to convert either the data type `image`, a noised version of the `image` or an algebraic expression into a DUNE-FEM discrete function. As domain  $\Omega$  we use the unit square  $[0, 1] \times [0, 1]$  and we scale any rectangular image accordingly. For non-square images this results in an anisotropic representation. The image resolution and aspect ratio relate to the level of initial refinement and to the number of cells in each direction. So  $n$  uniform refinements with  $l \times k$  initial cells lead to an image resolution of size  $l2^n \times k2^n$  square pixels, where every square is composed of two triangles and  $l/k$  corresponds to the aspect ratio. This is done in the gridfile `data/unitsquare2d.dgf` which describes the unit square using the DUNE DGF - Interval [BBD<sup>+</sup>08] nomenclature.

Setting `NSM_USE_NOISE=true` artificially applies noise to the image. We

currently apply two independent types of noise, where every noise implemented by CIMG can be applied. Noise type and noise level are controlled by a set of parameters defined in the `parameter` file. Setting `nsm.noiseType1` to 0 leads to Gaussian noise and setting it to 2 corresponds to salt-and-pepper noise. Setting any noise level parameter to 0 disables the respective noise. For more information consult the documentation of CIMG.

## 6.2.2 Implementation of the Primal-Dual Algorithm

The primal-dual algorithm from Section 6.1.2 is implemented in the file `src/nsm.cc`, where some of its subroutines, which depend on the continuity of the discrete spaces  $\mathbb{U}$  and  $\mathbb{P}$ , are outsourced into the file `src/phc.hh`. The template class `ProjectionHelperClass` uses partial template specialization to correctly define the methods `entitywiseProjection()` and `calculateZ()`.

The method `phc.entitywiseProjection()` realizes equations (6.8) and (6.9). Solving Eq. (6.8) is relatively easy, as all steps are provided by DUNE-ACFEM, e.g., for continuous  $\mathbb{U}$  and  $\mathbb{P}$ :

```

void entitywiseProjection ( const ForwardDiscreteFunctionType & uBar ,
    AdjointDiscreteFunctionType & p )
{
    //the gradient model defines the weak gradient using continuous test-functions
    //from the space of p
    auto gradU = gradientModel(uBar);
    auto Dbc0 = dirichletZeroModel(p);
    auto mass = massModel(p);
    //we solve p = sigma * nabla u + p
    //with dirichlet zero boundary
    auto model = mass - sigma_ * gradU - p + Dbc0 ;
    //Testspace = AdjointDiscreteFunctionSpaceType = space of p
    typedef EllipticFemScheme<AdjointDiscreteFunctionType , decltype(model)>
        SchemeType;
    //p is the returned solution
    SchemeType scheme(p, model);
    scheme.solve();
}

```

In the above case, for given  $\mathbf{p}$  and  $\bar{u}$  the equation

$$0 = \langle \bar{\mathbf{p}}, \boldsymbol{\phi} \rangle - \sigma \langle \bar{u}, -\operatorname{div} \boldsymbol{\phi} \rangle - \langle \mathbf{p}, \boldsymbol{\phi} \rangle \quad \forall \boldsymbol{\phi} \in \mathbb{P}$$

with Dirichlet-zero boundary conditions is solved with respect to  $\bar{\mathbf{p}}$ . The solution is written into the variable  $\mathbf{p}$ .

Projecting  $\mathbf{p}$  to be feasible (Eq. (6.9)) is not in the features of DUNE-ACFEM. So we have to use the features of DUNE-FEM directly. We choose to do the projection element-wise, i.e., we iterate over all elements of the grid. On each element we iterate over the degrees of freedom and if  $|\mathbf{p}|_2 > 1$  holds, we restrict the corresponding value to length 1 in the same direction. This implies the necessary condition  $\|\mathbf{p}\|_{L^\infty} \leq 1$  as  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  or  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$ . For  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  this automatically includes the necessary interpolation.

The calculation of  $z$  (Eq. (6.11)) translates nicely into code. We use the `L2Projection` of DUNE-ACFEM and its ability to linearly combine discrete functions. If  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$ , the weak form of Eq. (6.11) is

$$\langle z_k, \psi \rangle = \frac{1}{1 + 2\tau\alpha_2} \langle u_k + \tau \operatorname{div} \mathbf{p}_{k+1} + 2\tau\alpha_2 g, \psi \rangle \quad \forall \psi \in \mathbb{U}$$

and translates into code in the following way:

```

void calculateZ (const AdjointDiscreteFunctionType &p, const
  ForwardDiscreteFunctionType & uOld, ForwardDiscreteFunctionType& z)
{
  auto divP = divergence(p);
  L2Projection(1./(1.+tau_*lambda_2_) * ( tau_ * divP + uOld + tau_ *
    lambda_2_ * projG_ ), z);
}

```

For  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$  we use the weak divergence to shift the derivative to the test space  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$ . As above we use the basic `Models` of DUNE-ACFEM, in particular the `massModel` and the `weakDivergenceModel` to implement Eq. (6.11). In contrast to the implementation of the method `entitywiseProjection()` we construct the `massModel` from a discrete space

instead of a discrete function. This is explicitly allowed by DUNE-ACFEM as the object simply has to provide the data type for the test-function space. The code looks as follows:

```

void calculateZ (const AdjointDiscreteFunctionType &p, const
    ForwardDiscreteFunctionType & uOld, ForwardDiscreteFunctionType& z)
{
    //get the mass model of the Forward space
    auto U_Phi = massModel(projG_.space());
    //calculate z
    auto weakDiv_P = weakDivergenceModel(p);
    auto projModel = U_Phi -1./(1.+tau_*lambda_2_) * ( tau_ * weakDiv_P +
        uOld + tau_ * lambda_2_ * projG_);
    typedef EllipticFemScheme<ForwardDiscreteFunctionType, decltype(
        projModel)> SchemeType;
    SchemeType scheme(z, projModel);
    scheme.solve();
}

```

The summand `weakDiv_p` is turned into a functional for the case of  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{F})^2$  and  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$ . This is called `EdgeWeakDivergenceFunctional` and is contained in the file `edgeweakdivergencefunctional.hh`. The main implementation is done in the `coefficients()` method, that implements the application of the functional to the basis functions in DUNE-FEM notation. In the resulting code for the method `calculateZ()` the main difference is that `weakDivergenceModel(p)` is replaced by `edgeWeakDivergenceFunctional(p)`. Additionally `weakDiv_p` in the definition of `projModel` has been moved outside the brackets, as functions are not allowed to be added to functionals outside a model:

```

auto weakDiv_P = edgeWeakDivergenceFunctional(p);
auto projModel = U_Phi -1./(1.+tau_*lambda_2_) * ( uOld + tau_ *
    lambda_2_ * projG_ ) - tau_/(1.+tau_*lambda_2_) * weakDiv_p;

```

## 6.3 Numerical Experiments

The main focus of the numerical experiments is the choice of discrete spaces (Section 6.3.2). We compare the two natural choices for  $\mathbb{U}$  and  $\mathbb{P}$  and two additional choices motivated by other works. We use local refinement of the initial grid to reduce computational load, which is explained in Section 6.3.1. Finally we show, that the image actually is capable of denoising (Section 6.3.3) and that we gain intrinsical parallelization by using DUNE-ACFEM in combination with DUNE-ALUGRID and some strong scaling (Section 6.3.4).

Note that parameters in all experiments are not chosen optimal, but to resolve some discontinuities while smoothing others. The chosen parameters in each experiment are the result of a manual search of a parameter space. For an optimal choice of parameters, we refer the reader to [Lan17b], where an expensive algorithm is presented to choose quasi-optimal parameters. Moreover it may be of interest to choose local cost parameters, as in [CDIRS17, DHRC11, Lan17a].

### 6.3.1 Local Refinement

We do only refine on the initial grid and not refine during the iterations of the primal-dual algorithm. So the refinement increases the resolution of discontinuities of the given image and hence drastically improves the projection  $Pg$ . This is implemented in the method `adaptGrid()` in the file `nsm.cc`. We use the grid manager DUNE-ALUGRID [ADKN16], which is capable of handling the needed conforming, parallel, adaptive, triangular grids. As an indicator we use the height of jumps at a given image (i.e., a piecewise constant datum).

This is activated by setting the parameter `nsm.localRefine` to the positive level of additional local refinement. These local refinements are executed in the initialization phase in addition to the uniform refinements. As an example setting `nsm.localRefine` to 4 and `nsm.initialrefinements` to 3 will result in refining up to  $\mathcal{T}_3$  uniformly and executing 4 adaptation runs with

the following marking strategy for the piecewise constant datum  $g \in \mathcal{L}^0(\mathcal{T})$ . Given an entity  $T$  and its neighbour  $T'$ , if

$$|g|_T - g|_{T'}| > ADAPTTOL \quad (6.15)$$

holds, then  $T$  and  $T'$  are marked for refinement. The value of  $ADAPTTOL$  is given by the parameter `nsm.adaptTolerance`.

### 6.3.2 Comparison of Discrete Spaces

We investigate different discrete space pairings of  $\mathbb{U}$  and  $\mathbb{P}$ :

$$\begin{aligned} \mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2, & \quad \mathbb{U} = \mathcal{L}^0(\mathcal{T}), \mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2, \\ \mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2, & \quad \mathbb{U} = \mathcal{L}^0(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{F})^2 \end{aligned}$$

The setups using  $\mathbb{P} = \mathcal{L}_N^0(\cdot)^2$  are motivated by the equivalence of the mixed formulation and the primal formulation for these discrete settings. For the case  $\mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$  we even have guaranteed convergence to the continuous formulation, see Theorem 6.1.

The setting  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  is motivated by [Bar15a], where it is shown that for  $\mathbb{U} = \mathcal{S}^1(\mathcal{T}), \mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  the discrete pre-dual of the functional  $J_{0,\alpha_2}$  converges to the continuous one, which is an alternative way to tackle the given problem. However, setting  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  is not exploited in their proof. It can be for example replaced by choosing  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$ . Therefore we also investigate the case  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$ .

For this set of experiments we choose a similar example as in [Bar15a], i.e., the observed data is 1 on a disk of radius 0.3 and 0 elsewhere. No noise is applied in this experiment. Hence, we choose the discrete datum  $g$  to be given as a piecewise constant approximation of the function

$$f(x) = \begin{cases} 1, & \text{if } |x - (0.5, 0.5)| \leq 0.3 \\ 0, & \text{else.} \end{cases}$$

Note that this datum depends on the chosen grid. We executed the experiment for many different parameters. The best result to depict the difference between the spaces has the following set of parameters. The cost parameters are set to  $\alpha_1 = 10$  and  $\alpha_2 = 20$ , the stopping tolerance to  $10^{-2}$ , and the adaptation tolerance to 0.1. For the  $\mathcal{L}^0/\mathcal{L}_N^0$  case we choose the tolerance  $10^{-4}$  as we use an extension of  $\mathbb{P}$  into  $\Omega$  to calculate the part of  $\mathbf{p}$  in the stopping criterion  $\frac{\|u_{k+1}-u_k\|_{L^2}}{\tau} + \frac{\|\mathbf{p}_{k+1}-\mathbf{p}_k\|_{L^2}}{\sigma} < TOL$ . We do  $a = 3$  uniform refinements and  $b = 5$  additional local refinements. As convergence is guaranteed, if condition (6.14) holds, and since we know that in general  $\|\nabla u\|_{L^2} < Ch\|u\|_{L^2}$  for  $u \in \mathbb{U}$ , the step sizes are automatically set to  $\tau = \sigma = L * 2^{-(a+b)}$  with a suitable constant  $L$ , which depends on the norm of the gradient and the number of elements in the initial grid. For the  $\mathcal{L}^0/\mathcal{L}^0$  case, numerical tests indicate, that it is possible to even set  $\tau = \sigma = L * 2^{-(a+b)/2}$ . Experiments indicate that  $L = 0.19$  for the chosen initial grid, as above the algorithm diverges for higher values. The overrelaxation parameter  $\theta$  is chosen to be 1 to guarantee convergence by [CP11, Theorem 1].

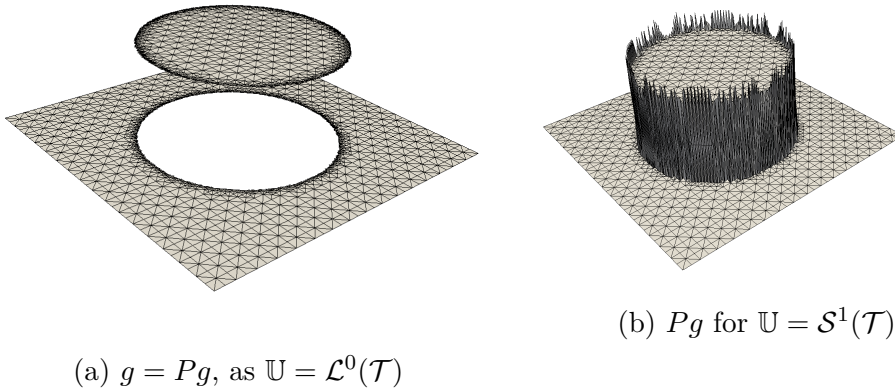


Figure 6.1: Projection of the piecewise constant datum  $g$  onto different spaces  $\mathbb{U}$ .

By imposing all equations weakly the primal-dual algorithm de facto uses the  $L^2$ -projection  $Pg \in \mathbb{U}$  instead of the datum  $g \in \mathcal{L}^0(\mathcal{T})$ . Figure 6.1 shows that in the case of  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  by projecting a discontinuous function onto

a continuous space we introduce an additional error in contrast to the case  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$ . The over- and undershoots of the continuous projected data do not pose a problem, because they are regularized over the course of the algorithm, as we can see in Figure 6.2b.

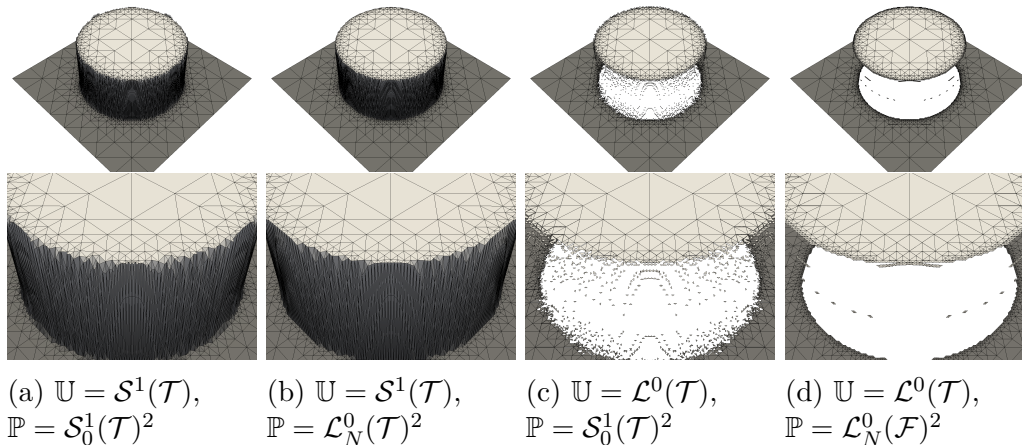


Figure 6.2: Discrete minima  $u^*$  of the functional  $J_{10,20}$

Comparing Figure 6.2a with 6.2b and Figure 6.2c with 6.2d we see that choosing  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$  smears out jumps, i.e., the discontinuities are less accurately approximated. This is due to the fact that  $\mathbb{P}$  is continuous and its basis functions have a larger support than if it were discontinuous. Additionally  $\mathbb{P} = \mathcal{S}^1(\mathcal{T})^2$  is not the space for which the primal and the mixed formulation are equivalent leading to a worse approximation of the total variation. This explains why the value of the functional in Figures 6.3 and 6.4 for these variants is higher than choosing  $\mathbb{P} = \mathcal{L}_N^0(\dots)^2$ . In particular this worsens the quality of the solutions obtained with  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$ ,  $\mathbb{P} = \mathcal{S}_0^1(\mathcal{T})^2$ , as now many elements close to the jump have values clearly distinct from 0 or 1 (Figure 6.2c), which increases the total variation.

Figure 6.3 depicts the evolution of the energy  $J_{10,20}$  during the iterations for the considered space pairings. We observe that  $J_{10,20}$  is not monotonically decreasing but stagnates at a minimal value after a certain number of iterations, note that the scale of the number of iterations in Figure 6.3



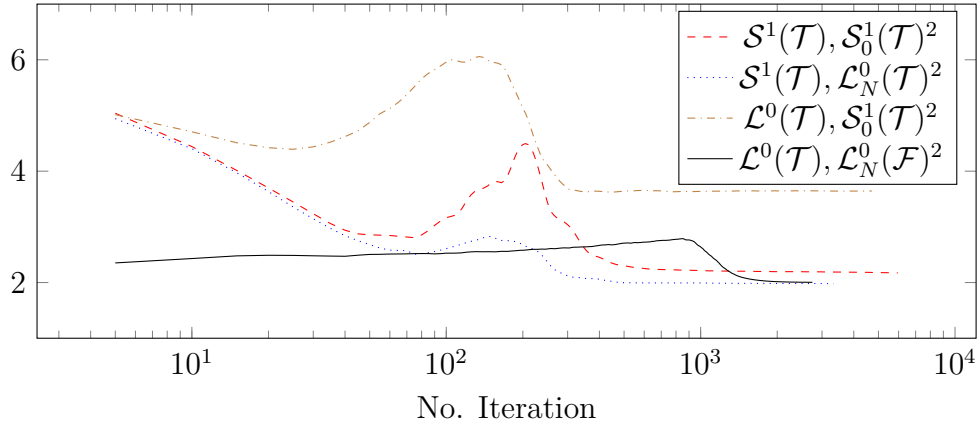


Figure 6.3: Value of the functional  $J_{10,20}$  over the course of the algorithm.

is logarithmic. This demonstrates, that in all these settings the algorithm converges to a saddle point of the respective discrete problem. Due to the different combinations of spaces it is clear that the stationary points in general do not coincide and hence the minimal energy is different. The combination  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$ ,  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$  yields the best result, as its final energy is the smallest among the considered cases.

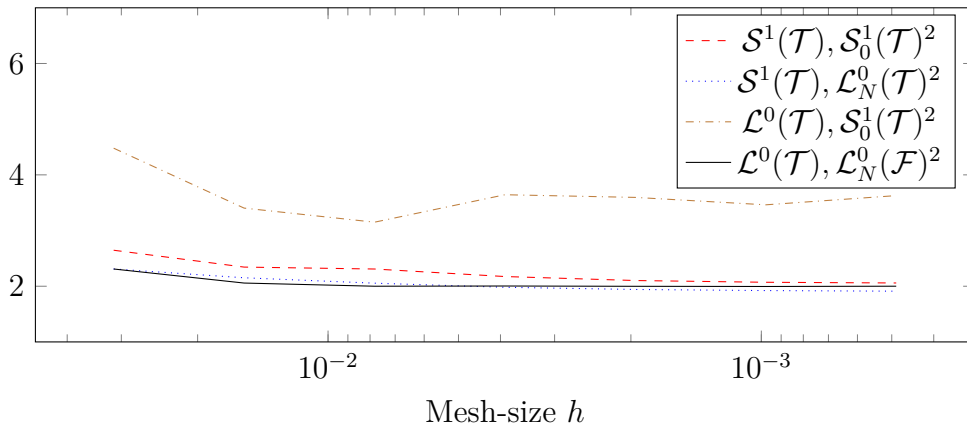


Figure 6.4: Final energy for  $h = 2^{-5}, \dots, 2^{-11}$

We observe that choosing  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  results in a decreasing energy for  $h \rightarrow 0$ , while for  $\mathbb{U} = \mathcal{L}^0(\mathcal{T})$  this is not the case (cf. Figure 6.4). More

precisely for the case  $(\mathcal{L}^0(\mathcal{T}), \mathcal{S}_0^1(\mathcal{T})^2)$  the energy seems to oscillate in a certain sense and for  $(\mathcal{L}^0(\mathcal{T}), \mathcal{L}_N^0(\mathcal{F})^2)$  the energy stays almost constant. This is due to the fact that, while the approximation of the circle gets better, the total variation does not diminish for  $\mathcal{L}^0$ . In general approximating functions of bounded variation is not possible with piecewise constant functions on a triangulation, since the length of discontinuities may not diminish over refinement [Bar12]. This is different for the case  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$ , where discontinuities of functions, that are not representable on the triangulation, can be better approximated.

### 6.3.3 Image Denoising

Here we demonstrate the denoising capability of the algorithm. Motivated by the above experiments we choose the spaces to be  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$ ,  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$ . The considered image with values in  $[0, 1]$  is first corrupted by Gaussian white noise with variance 0.1 and then salt-and-pepper noise with  $s = 0.1$  is added. The obtained image is shown in Figure 6.5b. As the total variation of the image is higher than in the artificial example in Section 6.3.2, we need a different magnitude of parameters. In order to reconstruct the image we choose  $\alpha_1 = 250$ ,  $\alpha_2 = 150$  in (6.1), and perform our primal-dual algorithm with the tolerance set to  $10^{-2}$ ,  $\theta = 1$  and  $\sigma = \tau = 2^{-8}$ . We refine uniformly until the grid contains two triangles for each of the  $256 \times 256$  pixels within the picture. We apply no additional local refinement. In Figure 6.5c we depict the output of our algorithm. The result is reasonably smoothed due to the total variation regularization, while discontinuities are still preserved.

As a second example we run the algorithm on a much finer image, that has a resolution of  $2576 \times 1920$  pixels (Figure 6.6a). The spaces are chosen as above and the cost parameters are  $\alpha_1 = \alpha_2 = 500$  to demonstrate that we lose fine-granular discontinuities we resolve the image on an initial grid of  $161 \times 120$  rectangles, each subdivided in two triangles and initiate 2 initial global refinements and 2 local refinements using the indicator from Equation (6.15). So any pixel is resolved by two triangles at discontinuities with local refinement. A zoom into the mesh is depicted in Figure 6.6b, where one may

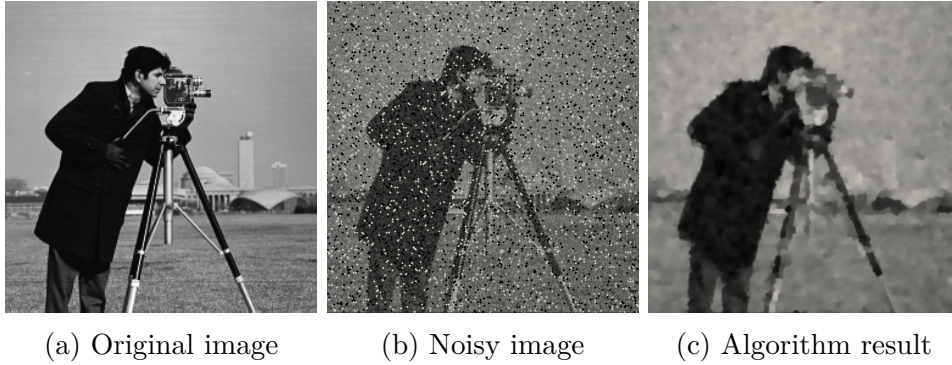


Figure 6.5: A corrupted image before and after applying the algorithm

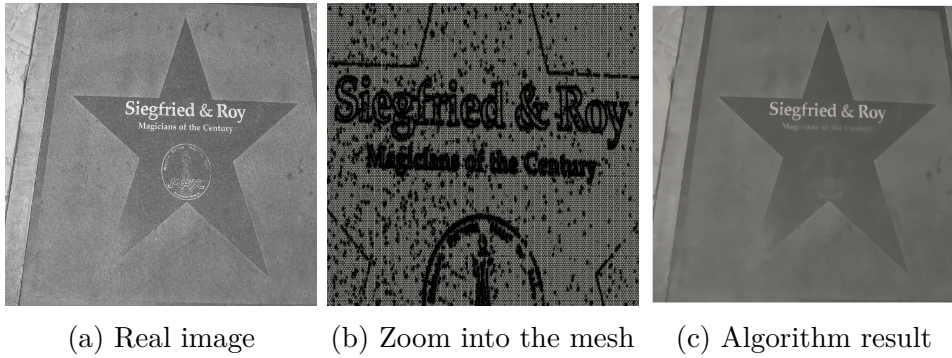


Figure 6.6: A real image before and after applying the algorithm

observe that the discontinuities are captured by the refinement, as intended.

The resulting image (Figure 6.6c) looks as expected, the noisy structure of the stone is reduced to a minimum, but we also lose some details inside the star, that we may have wanted to keep. This may be attributed to the choice of the parameters  $\alpha_1$  and  $\alpha_2$ .

### 6.3.4 Strong Scaling

We do a strong scaling experiment on the same datum as in Section 6.3.2 using the spaces  $\mathbb{U} = \mathcal{S}^1(\mathcal{T})$  and  $\mathbb{P} = \mathcal{L}_N^0(\mathcal{T})^2$  to demonstrate that the combination of DUNE-ACFEM and DUNE-ALUGRID is automatically parallel

# Processors	1	2	4	8	16	30
Runtime in s	970.118	501.147	253.487	162.019	89.4606	60.9888

Table 6.1: Strong scaling of the algorithm

and scales up to a certain extent. The important part about this result is, that we spent almost no effort to parallelize the code. The parallelism is done inside DUNE-ACFEM and the only line of code needed initializes MPI. This is a reexecution of the strong scaling experiment from [AL17, Section 4.3] with more workload.

The computation is executed on a machine with an Intel Xeon Processor E7 with 32 CPUs and 256 GB RAM. The grid manager DUNE-ALUGRID balances the computational load on the initial grid by partitioning it onto the different processors. So we cannot expect the algorithm to scale if the initial mesh is too coarse. E.g. if there are only two initial elements, only two processors can get partitions that are non-empty. Consequently we discretize the unit square by  $256 \times 256$  elements, see `data/finecube_2d.dgf`. As the grid is already fine, we do not need as many uniform refinements to reach a good resolution, so we do  $a = 1$  uniform and another  $b = 3$  local refinements. This results in a different operator norm of the gradient on the initial grid, so we have to set the corresponding parameter `nsm.constantL` to  $L = 0.0007$  for the step sizes  $\tau = \sigma = L * 2^{-(a+b)}$  to be computed correctly. Because we want to investigate the scaling behaviour we just execute the first 100 iterations of the algorithm by setting `nsm.maxIt` to 100 and disable the serial output by setting `nsm.outputStep` to  $-1$ .

Figure 6.7 indicates that the strong scaling up to 30 cores is decent. The values of the graph in Figure 6.7 are shown in Table 6.1. We did not execute the test for 32 cores, which would be the natural next step after 16, as there was a segmentation fault in the `MPIHelper` of DUNE, which is called in the first line that is executed. So either the bug is machine-specific or it is probably fixed in a current DUNE release, so we did not investigate this further.

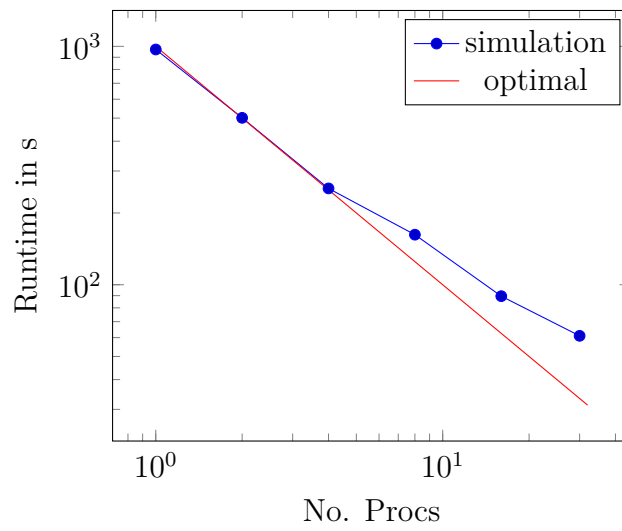


Figure 6.7: Strong scaling of the algorithm

# Acknowledgements

I am grateful for Kunibert Siebert giving me the opportunity to write this dissertation at his chair. I want to thank Claus-Justus Heine, Robert Klöfkorn and Andreas Dedner for their support and time, when I had implementation problems in DUNE and for sharing their expertise on adaptive and parallel grids. A lot of thanks goes to Dominik Göttsche and Alfred Schmidt as referees and in particular to Dominik for proof-reading and improving readability of this work. I would like to thank Andreas Langer, Fernando Gaspoz, Stephan Hilb and Birane Kane for lively discussions that often ended with new insights. Finally, I thank Maria and Simon, who put up with me, when I had trouble progressing. I also would like to thank the German Research Foundation (DFG) for financial support of the project within the Cluster of Excellence in Simulation Technology (EXC 310/1) at the University of Stuttgart.

# List of Figures

2.1	Kuhn-cube in 2 and 3 dimensions. . . . .	21
2.2	How to construct a set of cubes from a simplex in 2 and 3 dimensions. . . . .	22
2.3	Isometric cube refinement in 2 dimensions . . . . .	27
2.4	Isometric simplex refinement in 2 dimensions . . . . .	28
2.5	Arbitrary simplex bisection in 2 dimensions. . . . .	28
2.6	Refinement tree of Figure 2.5 . . . . .	30
2.7	Induced refinement tree of face $\overline{z_0z_1}$ . . . . .	30
2.8	An illustration of Condition 2.26. . . . .	31
2.9	Newest Vertex Bisection in 2d . . . . .	34
2.10	Newest Vertex Bisection in 3d. . . . .	35
2.11	NVB: The four similarity classes for an initial element $T_0$ . . . . .	38
2.12	2:1 Mesh-Balance for Cube Refinement in 3d . . . . .	43
2.13	Construction of Green Closure and Refinement Propagation in 2d. . . . .	44
3.1	Condensed refinement tree of a $d$ -simplex of type 0 under $d$ uniform refinements. . . . .	48
3.2	Condensed refinement tree of a $d$ -simplex of type 1. . . . .	49
3.3	Refinement behaviour of reflected neighbours in 2d . . . . .	55
3.4	Reflected Neighbouring Children . . . . .	56
3.5	Forbidden orientations by Condition 3.21 . . . . .	58
3.6	The conforming closure to different orderings for a 2d Kuhn-triangulation. . . . .	71
3.7	Series of tetrahedral grids discretizing the unit cube . . . . .	75

3.8	Proportion of vertices sorted into $\mathcal{V}_0$ over different thresholds for the unitcube sequence. . . . .	75
3.9	Proportion of vertices sorted into $\mathcal{V}_0$ over different thresholds for the realistic grids. . . . .	76
3.10	Average d-sine after uniform refinement for the unitcube sequence. . . . .	77
3.11	Average d-sine after uniform refinement for the realistic grids.	77
3.12	Maximum number of elements at a vertex for unitcube sequence	78
3.13	Maximum number of elements at a vertex without decorstatue	79
3.14	Maximum Number of Elements at a vertex for decorstatue . . .	79
3.15	Share of not strongly compatible faces, unitcube sequence . . .	81
3.16	Proportion of not strongly compatible faces, realistic grids. . .	82
3.17	Average conforming closure for first grid of unitcube sequence	82
3.18	Average conforming closure for Telescope grid (see Fig. 3.25e)	83
3.19	Average conforming closure for Kuhn-grid . . . . .	83
3.20	$d_{\mathcal{T}}^2$ and $d_{\mathcal{T}_3}^2$ for first grid of Unitcube Sequence . . . . .	84
3.21	$d_{\mathcal{T}}^2$ and $d_{\mathcal{T}_3}^2$ for Telescope grid . . . . .	84
3.22	$d_{\mathcal{T}}^2$ and $d_{\mathcal{T}_3}^2$ for Kuhn-grid . . . . .	85
3.23	Runtime of the algorithm for different grid sizes. . . . .	86
3.24	Refinement of a Kuhn-cube in 2 dimensions . . . . .	88
3.25	Realistic tetrahedral meshes . . . . .	89
4.1	The direct refinement propagation of Example 4.2 with $d = 4$ .	94
4.2	Refinement propagation graph example . . . . .	95
4.3	A distributed refined mesh to illustrate that the bound of theorem 4.5 is sharp. . . . .	97
4.4	Refinement propagation from uniform refinement of a single macro element . . . . .	100
4.5	Refinement situations at a macro vertex . . . . .	103
4.6	Refinement propagation graph for $v = z_3$ . . . . .	106
4.7	Refinement propagation graph for $v = z_2$ . . . . .	106
4.8	A 2d unit square with 18 macro elements on 18 processors. . .	112



4.9	A weakly compatible 2d grid with 60 macro elements on 60 processors. . . . .	113
4.10	Global communication of 3d experiment . . . . .	113
4.11	Refinement and coarsening of a doughnut like area that rotates around the center. . . . .	114
4.12	2d results for the doughnut refinement test . . . . .	115
4.13	3d results for the doughnut refinement test . . . . .	116
4.14	3d results for the doughnut refinement test . . . . .	117
4.15	3d results for the doughnut refinement test . . . . .	118
5.1	Comparison of memory usage and run times for the 2d version in the old and new implementation . . . . .	122
5.2	Construction and refinement of the 3d cube extension. . . . .	123
5.3	Construction and Refinement of the 3d simplex extension. . . . .	125
5.4	3d surface cube extension of an example grid . . . . .	127
5.5	3d surface simplex extension of a sphere . . . . .	128
5.6	Run time of Algorithm 5.2 for different final grid sizes. . . . .	134
6.1	Projection of the piecewise constant datum $g$ onto different spaces $\mathbb{U}$ . . . . .	150
6.2	Discrete minima $u^*$ of the functional $J_{10,20}$ . . . . .	151
6.3	Value of the functional $J_{10,20}$ over the course of the algorithm. . . . .	152
6.4	Final energy for $h = 2^{-5}, \dots, 2^{-11}$ . . . . .	152
6.5	A corrupted image before and after applying the algorithm . . . . .	154
6.6	A real image before and after applying the algorithm . . . . .	154
6.7	Strong scaling of the algorithm . . . . .	156

# Bibliography

- [ACH<sup>+</sup>15] Guillermo Aparicio, Leocadio G. Casado, Eligius M. T. Hendrix, Boglárka G.-Tóth, and Inmaculada Garcia. On the Minimum Number of Simplex Shapes in Longest Edge Bisection Refinement of a Regular  $n$ -Simplex. *Informatica (Vilnius)*, 26(1):17–32, 2015.
- [ADKN16] Martin Alkämper, Andreas Dedner, Robert Klöfkorn, and Martin Nolte. The DUNE-ALUGrid Module. *Archive of Numerical Software*, 4(1):1–28, 2016.
- [AFP00] Luigi Ambrosio, Nicola Fusco, and Diego Pallara. *Functions of Bounded Variation and Free Discontinuity Problems*. Oxford Mathematical Monographs. The Clarendon Press, Oxford University Press, New York, 2000.
- [AGK18] Martin Alkämper, Fernando Gaspoz, and Robert Klöfkorn. A Weak Compatibility Condition for Newest Vertex Bisection in Any Dimension. *SIAM Journal on Scientific Computing*, 40(6):A3853–A3872, 2018.
- [AK17] Martin Alkämper and Robert Klöfkorn. Distributed Newest Vertex Bisection. *Journal of Parallel and Distributed Computing*, 104:1–11, 2017.
- [AL17] Martin Alkämper and Andreas Langer. Using DUNE-ACFem for Non-smooth Minimization of Bounded Variation Functions. *Archive of Numerical Software*, 5(1):3–19, 2017.

- [AMP00] Douglas N. Arnold, Arup Mukherjee, and Luc Pouly. Locally Adapted Tetrahedral Meshes Using Bisection. *SIAM Journal on Scientific Computing*, 22(2):431–448, 2000.
- [Ape99] Thomas Apel. *Anisotropic Finite Elements: Local Estimates and Applications*. Stuttgart: Teubner, 1999.
- [Bad12] Michael Bader. *Space-Filling Curves: An Introduction with Applications in Scientific Computing*. Springer, 2012.
- [Ban83] Randolph E. Bank. The Efficient Implementation of Local Mesh Refinement Algorithms. In *Adaptive computational methods for partial differential equations (College Park, Md., 1983)*, pages 74–81. SIAM, Philadelphia, PA, 1983.
- [Bän91] Eberhard Bänsch. Local Mesh Refinement in 2 and 3 Dimensions. *IMPACT of Computing in Science and Engineering*, 3:181–191, 1991.
- [Bar12] Sören Bartels. Total Variation Minimization with Finite Elements: Convergence and Iterative Solution. *SIAM Journal on Numerical Analysis*, 50(3):1162–1180, 2012.
- [Bar15a] Sören Bartels. Broken Sobolev Space Iteration for Total Variation Regularized Minimization Problems. *IMA Journal of Numerical Analysis*, 36:493–502, 2015.
- [Bar15b] Sören Bartels. Error Control and Adaptivity for a Variational Model Problem Defined on Functions of Bounded Variation. *Mathematics of Computation*, 84(293):1217–1240, 2015.
- [BBD<sup>+</sup>06] Peter Bastian, Markus Blatt, Andreas Dedner, Christian Engwer, Robert Klöfkorn, Sreejith P. Kuttanikkad, Mario Ohlberger, and Oliver Sander. The Distributed and Unified Numerics Environment (DUNE). In *Proceedings of the 19th Symposium on Simulation Technique in Hannover, Sep. 12 - 14, 2006*.

- [BBD<sup>+</sup>08] Peter Bastian, Markus Blatt, Andreas Dedner, Christian Engwer, Robert Klöfkorn, Ralf Kornhuber, Mario Ohlberger, and Oliver Sander. A Generic Grid Interface for Parallel and Adaptive Scientific Computing. II. Implementation and Tests in DUNE. *Computing*, 82(2-3):121–138, 2008.
- [BBD<sup>+</sup>16] Markus Blatt, Ansgar Burchardt, Andreas Dedner, Christian Engwer, Jorrit Fahlke, Bernd Flemisch, Christoph Gersbacher, Carsten Gräser, Felix Gruber, Christoph Grüninger, Dominic Kempf, Robert Klöfkorn, Tobias Malkmus, Steffen Müthing, Martin Nolte, Marius Piatkowski, and Oliver Sander. The Distributed and Unified Numerics Environment, Version 2.4. *Archive of Numerical Software*, 4(100):13–29, 2016.
- [BBHK11] Wolfgang Bangerth, Carsten Burstedde, Timo Heister, and Martin Kronbichler. Algorithms and Data Structures for Massively Parallel Generic Adaptive Finite Element Codes. *ACM Transactions on Mathematical Software*, 38(2):Art. 14, 28, 2011.
- [BBJ<sup>+</sup>97] Peter Bastian, Klaus Birken, Klaus Johannsen, Stefan Lang, Nicolas Neuß, Henrik Rentz-Reichert, and Christian Wieners. UG – A Flexible Software Toolbox for Solving Partial Differential Equations. *Computing and Visualization in Science*, 1(1):27–40, 1997.
- [BCCD12] Erik G. Boman, Umit V. Catalyurek, Cedric Chevalier, and Karen D. Devine. The Zoltan and Isorropia Parallel Toolkits for Combinatorial Scientific Computing: Partitioning, Ordering, and Coloring. *Scientific Programming*, 20(2):129–150, 2012.
- [BDD04] Peter Binev, Wolfgang Dahmen, and Ronald A. DeVore. Adaptive Finite Element Methods with Convergence Rates. *Numerische Mathematik*, 97:219–268, 2004.
- [BDKO06] A. Burri, A. Dedner, R. Klöfkorn, and M. Ohlberger. An efficient implementation of an adaptive and parallel grid in dune. In

- Egon Krause, Yurii Shokin, Michael Resch, and Nina Shokina, editors, *Computational Science and High Performance Computing II*, pages 67–82, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [Bey95] Jürgen Bey. Tetrahedral Grid Refinement. *Computing*, 55(4):355–378, 1995.
- [Bey00] Jürgen Bey. Simplicial Grid Refinement: on Freudenthal’s Algorithm and the Optimal Number of Congruence Classes. *Numerische Mathematik*, 85(1):1–29, 2000.
- [BHK07] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. deal.II – a General Purpose Object Oriented Finite Element Library. *ACM Transactions on Mathematical Software*, 33(4):24/1–24/27, 2007.
- [BKK09] Jan Brandts, Sergey Korotov, and Michal Křížek. On the Equivalence of Ball Conditions for Simplicial Finite Elements in  $\mathbb{R}^d$ . *Applied Mathematics Letters*, 22(8):1210–1212, 2009.
- [BS01] Ivo Babuška and Theofanis Strouboulis. *The Finite Element Method and its Reliability*. Numerical Mathematics and Scientific Computation. The Clarendon Press, Oxford University Press, New York, 2001.
- [BWG11] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. p4est: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133, 2011.
- [CDIRS17] Cao Van Chung, Juan Carlos De los Reyes, and Carola-Bibiane Schönlieb. Learning Optimal Spatially-dependent Regularization Parameters in Total Variation Image Restoration. *Inverse Problems*, 33(7):074005, 2017.

- [CFL28] Richard Courant, Kurt Friedrichs, and Hans Lewy. Über die partiellen Differenzgleichungen der mathematischen Physik. *Mathematische Annalen*, 100(1):32–74, 1928.
- [Cha04] Antonin Chambolle. An algorithm for total variation minimization and applications. *J. Math. Imaging Vision*, 20(1-2):89–97, 2004. Special issue on mathematics and image analysis.
- [CKNS08] J. Manuel Cascon, Christian Kreuzer, Ricardo H. Nochetto, and Kunibert G. Siebert. Quasi-optimal Convergence Rate for an Adaptive Finite Element Method. *SIAM Journal of Numerical Analysis*, 46(5):2524–2550, 2008.
- [CP08] Cedric Chevalier and Francois Pellegrini. PT-Scotch: A Tool for Efficient Parallel Graph Ordering. *Parallel Computing*, 34(6):318 – 331, 2008. Parallel Matrix Algorithms and Applications.
- [CP11] Antonin Chambolle and Thomas Pock. A First-order Primal-dual Algorithm for Convex Problems with Applications to Imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145, 2011.
- [CPE18] Matteo Cicuttin, Daniele A. Di Pietro, and Alexandre Ern. Implementation of Discontinuous Skeletal Methods on Arbitrary-dimensional, Polytopal Meshes Using Generic Programming. *Journal of Computational and Applied Mathematics*, 344:852 – 874, 2018.
- [CRW17] Christophe Chalons, Christian Rohde, and Maria Wiebe. A Finite Volume Method for Undercompressive Shock Waves in Two Space Dimensions. *ESAIM: M2AN*, 51(5):1987–2015, 2017.
- [DHRC11] Yiqiu Dong, Michael Hintermüller, and M. Monserrat Rincon-Camacho. Automated Regularization Parameter Selection in Multi-scale Total Variation Models for Image Restoration. *Journal of Mathematical Imaging and Vision*, 40(1):82–104, 2011.

- [DKNO10] Andreas Dedner, Robert Klöforn, Martin Nolte, and Mario Ohlberger. A Generic Interface for Parallel and Adaptive Discretization Schemes: Abstraction Principles and the DUNE-Fem Module. *Computing*, 90(3):165–196, 2010.
- [Dör96] Willy Dörfler. A Convergent Adaptive Algorithm for Poisson’s Equation. *SIAM Journal of Numerical Analysis*, 33(3):1106–1124, 1996.
- [EHH<sup>+</sup>11] Robert Eymard, Gérard Henry, Raphaële Herbin, Florence Hubert, Robert Klöforn, and Gianmarco Manzini. 3D Benchmark on Discretization Schemes for Anisotropic Diffusion Problems on General Grids. In *Finite Volumes for Complex Applications VI Problems & Perspectives*, volume 4 of *Springer Proceedings in Mathematics*, pages 895–930. Springer Berlin Heidelberg, 2011.
- [Eri78] Folke Eriksson. The Law of Sines for Tetrahedra and  $n$ -Simplices. *Geometriae Dedicata*, 7(1):71–80, 1978.
- [Fre42] Hans Freudenthal. Simplicialzerlegungen von Beschränkter Flachheit. *Annals of Mathematics*, 43(3):580–582, 1942.
- [GAM16] INRIA GAMMA. 3D Meshes Research Database. <https://www.rocq.inria.fr/gamma/gamma/download/download.php>, 2016.
- [GHS15] Fernando D. Gaspoz, Claus-Justus Heine, and Kunibert G. Siebert. Optimal Grading of the Newest Vertex Bisection and  $H^1$ -Stability of the  $L^2$ -Projection. *IMA Journal of Numerical Analysis*, 2015.
- [Giu84] Enrico Giusti. *Minimal Surfaces and Functions of Bounded Variation*, volume 80 of *Monographs in Mathematics*. Birkhäuser Verlag, Basel, 1984.
- [GKO14] Stefan Girke, Robert Klöforn, and Mario Ohlberger. Efficient Parallel Simulation of Atherosclerotic Plaque Formation Using

- Higher Order Discontinuous Galerkin Schemes. In *Finite Volumes for Complex Applications VII*, volume 78 of *Springer Proceedings in Mathematics & Statistics*, pages 617–625. Springer, 2014.
- [GR09] Christophe Geuzaine and Jean-Francois Remacle. Gmsh: A Three-dimensional Finite Element Mesh Generator with Built-in Pre- and Post-processing Facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.
- [Gra19] Jörg Grande. Red-Green Refinement of Simplicial Meshes in  $D$  Dimensions. *Mathematics of Computation*, 88:751–782, 2019.
- [GSS14] Dietmar Gallistl, Mira Schedensack, and Rob P. Stevenson. A Remark on Newest Vertex Bisection in Any Space Dimension. *Computational Methods in Applied Mathematics*, 14(3):317–320, 2014.
- [Hec12] Frederic Hecht. New Development in FreeFem++. *Journal of Numerical Mathematics*, 20(3-4):251–265, 2012.
- [Hei14] Claus-Justus Heine. DUNE-ACFem Documentation. <http://www.ians.uni-stuttgart.de/nmh/stage/documentation/software/dune-acfem/doxygen/>, 2014.
- [HK04] Michael Hintermüller and Karl Kunisch. Total Bounded Variation Regularization as a Bilaterally Constrained Optimization Problem. *SIAM Journal on Applied Mathematics*, 64(4):1311–1333, 2004.
- [HKK10] Antti Hannukainen, Sergey Korotov, and Michal Křížek. On Global and Local Mesh Refinements by a Generalized Conforming Bisection Algorithm. *Journal of Computational and Applied Mathematics*, 235(2):419–436, 2010.
- [HL13] Michael Hintermüller and Andreas Langer. Subspace Correction Methods for a Class of Nonsmooth and Nonadditive Convex



- Variational Problems with Mixed  $L^1/L^2$  Data-fidelity in Image Processing. *SIAM Journal on Imaging Sciences*, 6(4):2134–2173, 2013.
- [Hor97] Reiner Horst. On Generalized Bisection of  $n$ -Simplices. *Mathematics of Computation*, 66(218):691–698, 1997.
- [HRC14] Michael Hintermüller and Monserrat Rincon-Camacho. An Adaptive Finite Element Method in  $L^2$ -TV-based Image Denoising. *Inverse Problems and Imaging*, 8(3):685–711, 2014.
- [HS06] Michael Hintermüller and Georg Stadler. An Infeasible Primal-dual Algorithm for Total Bounded Variation-based Inconvolution-type Image Restoration. *SIAM Journal on Scientific Computing*, 28(1):1–23, 2006.
- [HS18] Susan Hert and Michael Seel.  $dD$  Convex Hulls and Delaunay Triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.13 edition, 2018.
- [Hua06] Weizhang Huang. Mathematical Principles of Anisotropic Mesh Adaptation. *Communications in Computational Physics*, 1:276–310, 04 2006.
- [IBG12] Tobin Isaac, Carsten Burstedde, and Omar Nabih Ghattas. Low-cost parallel algorithms for 2:1 octree balance. In *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012*, Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012, pages 426–437, 10 2012.
- [IBWG15] Tobin Isaac, Carsten Burstedde, Lucas C Wilcox, and Omar Ghattas. Recursive Algorithms for Distributed Forests of Octrees. *SIAM Journal on Scientific Computing*, 37(5):C497–C531, 2015.

- [JDB<sup>+</sup>15] Markus Jehl, Andreas Dedner, Timo Betcke, Kirill Aristovich, Robert Klöfkorn, and David Holder. A Fast Parallel Solver for the Forward Problem in Electrical Impedance Tomography. *IEEE Transactions on Biomedical Engineering*, 62(1):126–137, 2015.
- [JHJ12] Niclas Jansson, Johan Hoffman, and Johan Jansson. Framework for Massively Parallel Adaptive Finite Element Computational Fluid Dynamics on Tetrahedral Meshes. *SIAM Journal on Scientific Computing*, 34(1):C24–C41, 2012.
- [JP97] Mark T Jones and Paul E Plassmann. Parallel Algorithms for Adaptive Mesh Refinement. *SIAM Journal on Scientific Computing*, 18(3):686–708, 1997.
- [KDB02] Shailendra Kumar, Sajal Das, and Rupak Biswas. Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments. In *Conference: 16th International Parallel and Distributed Processing Symposium (IPDPS 2002) Proceedings*, pages 15–19, 2002.
- [KK70] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20, 02 1970.
- [KKK08] Sergey Korotov, Michal Křížek, and A. Kropáč. Strong Regularity of a Family of Face-to-face Partitions Generated by the Longest-edge Bisection Algorithm. *Computational Mathematics and Mathematical Physics*, 48(9):1687–1698, 2008.
- [Kos94] Igor Kossaczky. A Recursive Approach to Local Mesh Refinement in Two and Three Dimensions. *Journal of Computational and Applied Mathematics.*, 55:275–288, 1994.
- [KPS16] Sergey Korotov, Ángel Plaza, and José P. Suárez. Longest-edge  $n$ -Section Algorithms: Properties and Open Problems. *Journal of Computational and Applied Mathematics*, 293:139–146, 2016.

- [Kuh60] Harold W Kuhn. Some Combinatorial Lemmas in Topology. *IBM Journal of research and development*, 4(5):518–524, 1960.
- [Lan17a] Andreas Langer. Automated Parameter Selection for Total Variation Minimization in Image Restoration. *Journal of Mathematical Imaging and Vision*, 57(2):239–268, Feb 2017.
- [Lan17b] Andreas Langer. Automated parameter selection in the  $L^1$ - $L^2$ -TV model for removing Gaussian plus impulse noise. *Inverse Problems*, 33(7):074002, 2017.
- [LMZ08] Q. Liu, Z. Mo, and L. Zhang. A Parallel Adaptive Finite-element Package Based on ALBERTA. *International Journal of Computer Mathematics*, 85(12):1793–1805, 2008.
- [LWH12] Anders Logg, Garth N. Wells, and Johan Hake. *DOLFIN: a C++/Python Finite Element Library*, pages 173–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.
- [Mau95] Joseph M. Maubach. Local Bisection Refinement for  $n$ -Simplicial Grids Generated by Reflection. *SIAM Journal on Scientific Computing*, 16(1):210–227, 1995.
- [MB15] Oliver Meister and Michael Bader. 2D Adaptivity for 3D Problems: Parallel SPE10 Reservoir Simulation on Dynamically Adaptive Prism Grids. *Journal of Computational Science*, 9:101–106, 2015.
- [Mit88] William F. Mitchell. *Unified Multilevel Adaptive Finite Element Methods for Elliptic Problems*. PhD thesis, Department of Computer Science, University of Illinois, Urbana, 1988.
- [Mit89] William F. Mitchell. A Comparison of Adaptive Refinement Techniques for Elliptic Problems. *ACM Transactions in Mathematical Software*, 15:326–347, 1989.
- [Mit16] William F. Mitchell. 30 Years of Newest Vertex Bisection. *AIP Conference Proceedings*, 1738(1), 2016.

- [NCA12] NCAR/CISL. Computational and Information Systems Laboratory. Yellowstone: IBM iDataPlex System (Climate Simulation Laboratory). Boulder, CO: National Center for Atmospheric Research. <http://n2t.net/ark:/85065/d7wd3xhc>, 2012.
- [Nik04] Mila Nikolova. A variational approach to remove outliers and impulse noise. *Journal of Mathematical Imaging and Vision*, 20(1-2):99–120, 2004.
- [POG08] Doug Pagnutti and Carl Ollivier-Gooch. Delaunay-based anisotropic mesh adaptation. In Rao V. Garimella, editor, *Proceedings of the 17th International Meshing Roundtable*, pages 141–157, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.
- [PSP+04] Ángel Plaza, José P. Suárez, Miguel A. Padrón, Sergio Falcón, and Daniel Amieiro. Mesh Quality Improvement and Other Properties in the Four-triangles Longest-edge Partition. *Computer Aided Geometric Design*, 21(4):353 – 369, 2004.
- [RCFC06] Maria-Cecilia Rivara, Carlo Calderon, Andriy Fedorov, and Nikos Chrisochoides. Parallel Decoupled Terminal-edge Bisection Method for 3D Mesh Generation. *Engineering with Computers*, 22(2):111–119, 2006.
- [RI96] Maria-Cecilia Rivara and Gabriel Iribarren. The 4-Triangles Longest-side Partition of Triangles and Linear Refinement Algorithms. *Mathematics of Computation*, 65:1485–1502, 10 1996.
- [Riv97] Maria-Cecilia Rivara. New Longest-edge Algorithms for the Refinement and/or Improvement of Unstructured Triangulations. *International Journal for Numerical Methods in Engineering*, 40(18):3313–3324, 1997.
- [SBB12] Martin Schreiber, Hans-Joachim Bungartz, and Michael Bader. Shared Memory Parallelization of Fully-adaptive Simulations Using a Dynamic Tree-split and -join Approach. In *20th Annual In-*

- ternational Conference on High Performance Computing*, pages 1–10. IEEE Computer Society, 2012.
- [Sch97] Joachim Schoeberl. NETGEN An Advancing Front 2D/3D-Mesh Generator Based on Abstract Rules. *Computing and Visualization in Science*, 1:41–52, 07 1997.
- [Sch99] Bernhard Schupp. *Entwicklung eines effizienten Verfahrens zur Simulation kompressibler Strömungen in 3D auf Parallelrechnern*. PhD thesis, Universität Freiburg, 1999.
- [Sch13] René Schneider. *A Review of Anisotropic Refinement Methods for Triangular Meshes in FEM*, pages 133–152. Springer Berlin Heidelberg, 2013.
- [Sew72] E.G. Sewell. *Automatic Generation of Triangulations for Piecewise Polynomial Approximation*. PhD thesis, 1972.
- [Si15] Hang Si. TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Transactions on Mathematical Software*, 41(2), 2015.
- [SS05] Alfred Schmidt and Kunibert G. Siebert. *Design of Adaptive Finite Element Software*, volume 42. Springer-Verlag, Berlin, 2005.
- [SSB08] Hari Sundar, Rahul S Sampath, and George Biros. Bottom-up Construction and 2:1 Balance Refinement of Linear Octrees in Parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008.
- [SSH<sup>+</sup>] Alfred Schmidt, Kunibert G. Siebert, Claus-Justus Heine, Daniel Köster, and Oliver Kriessl. ALBERTA: An Adaptive Hierarchical Finite Element Toolbox. Version 1.2 and 2.0.
- [Ste08] Rob Stevenson. The Completion of Locally Refined Simplicial Partitions Created by Bisection. *Mathematics of Computation*, 77(261):227–241, 2008.

- [The18] The CGAL Project. CGAL User and Reference Manual. <https://doc.cgal.org/4.13/Manual/packages.html>, 2018.
- [Tra97] Christoph T. Traxler. An Algorithm for Adaptive Mesh Refinement in  $n$  Dimensions. *Computing*, 59:115–137, 1997.
- [Tsc] David Tschumperlé. The CImg Library - C++ Template Image Processing Toolkit. <http://www.cimg.eu/reference/>.
- [WLPV15] Thomas Witkowski, Siqi Ling, Simon Praetorius, and Axel Voigt. Software Concepts and Numerical Algorithms for a Scalable Adaptive Parallel Finite Element Method. *Advances in Computational Mathematics*, pages 1–33, 2015.
- [Zha09] Lin-Bo Zhang. A Parallel Algorithm for Adaptive Local Refinement of Tetrahedral Meshes Using Bisection. *Numerical Mathematics Theory Methods and Applications*, 2(1):65–89, 2009.
- [Zum02] Gerhard Zumbusch. Load Balancing for Adaptively Refined Grids. *Proceedings in Applied Mathematics and Mechanics*, (1):534–537, 2002. also as report 722 SFB 256, University Bonn.