Diplomarbeit

Analysis and Optimization of Nested Meshes for Adaptive Mesh Refinement

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Masterstudium Technische Mathematik

eingereicht von

Florian Pöppl, BSc
Matrikelnummer 01425416

ausgeführt am Institut für Mikroelektronik
eingereicht an der Fakultät für Mathematik und Geoinformation der Technischen Universität Wien

Betreuung
Hauptbetreuer: Privatdoz. Dipl.-Ing. Dr.techn. Josef Weinbub, BSc
Mitbetreuer: Univ.Ass. Dipl.-Ing. Michael Quell, BSc

Wien, 04.05.2020

(Unterschrift Verfasser) (Unterschrift Betreuer)

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Contents

1 Introduction **3**
1.1 Motivation and Objectives 3
1.2 Outline 5

2 Adaptive Mesh Refinement **6**
2.1 Patch-Based AMR 9
2.2 The AMR Algorithm 14
2.2.1 Timestepping 14
2.2.2 Flagging 16
2.2.3 Regridding 17
2.2.4 Initialization 17
2.2.5 Synchronization 18
2.2.6 Averaging 20

3 Mesh Cell Clustering **21**
3.1 Single-Level Clustering 22
3.1.1 Signature-Inflection Clustering 24
3.1.2 Tile Clustering 31
3.2 Multi-Level Clustering 33
3.2.1 Top-Down Clustering 34
3.2.2 Bottom-Up Clustering 37

4 Implementation **39**
4.1 Requirements 39
4.2 Interface 40
4.3 Parallelization 42

5 Numerical Experiments **45**
5.1 Clustering Examples 45
5.1.1 Top-Down Signature Clustering 45
5.1.2 Bottom-Up Tile Clustering 56
5.1.3 Comparison of Signature Clustering and Tile Clustering 58
5.2 Vortex Flow Simulation 60

6 Conclusion **63**

A Code Samples **64**
A.1 Mesh and Grid Interfaces 64
A.2 Vortex Flow Simulation 65

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Abstract

Adaptive mesh refinement is a numerical method which locally increases the mesh resolution within existing meshes in order to reduce the computational effort of numerical simulations. The mesh resolution is only increased where required, for example, in areas with a high estimated error. This enables accurate simulation of problems with highly varying spatial scales, which would be unfeasible with fixed mesh resolutions.

Focusing on adaptive refinement of Cartesian meshes, the nested meshes have to be properly embedded in a mesh hierarchy according to specific nesting criteria. The creation of such a nested mesh hierarchy requires a priori defined points of interest to be suitably clustered. In line with the goal of reducing computational effort of subsequent numerical simulations, the number of meshes as well as the number of individual mesh cells have to be optimally minimized. The challenge in creating an efficient mesh hierarchy is finding a trade-off between these quantities and the computational cost for the adaptation of the hierarchical data structure, including interpolation of the data associated with the mesh cells.

In this thesis, two approaches to mesh clustering and nested mesh generation are evaluated and a novel algorithm for transforming a given Cartesian nested mesh configuration according to given points of interest is discussed. The algorithm produces efficient nested mesh hierarchies that are guaranteed to conform to extended nesting criteria, e.g., each mesh also has a unique parent mesh. An implementation of the adaptive mesh refinement procedure, including the mesh generation algorithms, is presented together with numerical examples. For these examples, the novel algorithm reduces the total number of cells for a single refinement level by up to 10%, depending on the problem geometry.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

1 Introduction

1.1 Motivation and Objectives

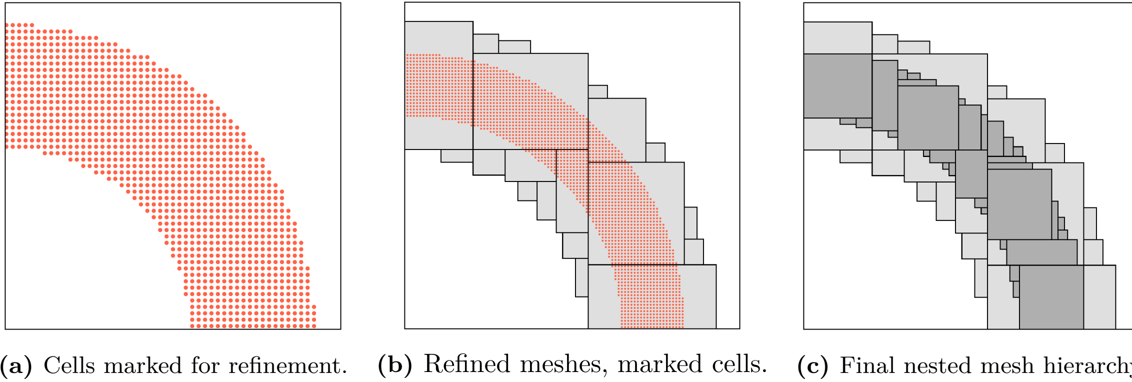
Even though numerical methods for partial differential equations (PDEs) are always advancing in speed and accuracy, certain problems require resolutions that cannot be feasibly simulated in full or feature strongly differing spatial scales. The accuracy of the numerical solution of PDEs, here considered to ultimately represent a numerical simulation, is directly related to the resolution of the approximating discrete computational mesh. If the mesh resolution is too coarse, regions with a highly varying solution are approximated inaccurately leading to incorrect simulations. However, a finer resolution in turn strongly increases the computational effort of considered numerical simulations. One approach to overcome this problem is to focus the computational effort on areas where accuracy is most important, while paying less attention to less interesting areas – such as parts of the domain where the solution is constant. This yields the desired accuracy while keeping the cost of computation low. These areas of interest are usually not known a priori, but have to be determined during a simulation and the discretization has to be adapted accordingly.

The numerical solution of PDEs necessitates a discretization of the problem domain, referred to as the, previously indicated, computational mesh. For such a mesh, function values are stored at specified points in space, called nodes or cells. Meshes are categorized as either structured or unstructured meshes, where structured meshes are characterized by pre-defined connectivity and regular positioning (e.g., a Cartesian mesh), whereas unstructured meshes can have arbitrarily placed nodes (e.g., triangular or tetrahedral meshes). While adaptive methods exist for both types of meshes, this work focuses on structured meshes as they are particularly attractive for certain numerical problems involving finite difference (and related) schemes due to reducing the complexity and memory requirements of the numerical algorithms, as connectivity information is given inherently.

Specifically, adaptive mesh refinement (AMR) as presented in [1] is a technique for solving compressible flow problems, as well as more general hyperbolic differential equations, which dynamically refines a structured Cartesian (i.e., uniform quadrilateral or hexahedral cells) mesh in areas where it is required, based on certain error measurements or solution properties [8]. This creates a hierarchy of nested meshes with increasingly fine mesh spacing (cf. Figure 1.1), where a higher accuracy is achieved by propagating the solutions from the finer meshes to the coarser meshes.

For time-dependent PDEs, such as hyperbolic differential equations, the solution is known at an initial starting time and needs to be advanced forward in time. This is usually done in discrete intervals, called *timesteps*. The timestepping of a nested mesh hierarchy is then not much different from timestepping any single Cartesian mesh. Existing numerical methods for Cartesian meshes, usually finite volume or finite difference methods [16], can straightforwardly be integrated into the AMR procedure. The challenge is to identify the areas which should be refined as well as where to place the refined meshes. For the latter, there are certain constraints as to how meshes may be nested.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



(a) Cells marked for refinement. **(b)** Refined meshes, marked cells. **(c)** Final nested mesh hierarchy.

Figure 1.1: Adaptive mesh refinement identifies and selectively refines areas of interest in the problem domain. Starting from a coarsely-resolved root mesh covering the whole domain, cells are marked for refinement and a new set of meshes with finer resolution is created, covering all marked cells. These new meshes may be marked and refined again. This process can continue recursively up to a maximum depth (in this case three levels), resulting in a hierarchy of nested meshes.

There exist a number of software packages implementing AMR algorithms. The *Clawpack* software suite [32] is an open source library of finite volume methods for linear and nonlinear hyperbolic systems of conservation laws. Clawpack includes a Fortran-based AMR implementation called *AMRClaw* [11] which supports patch-based AMR with multiple strategies for identifying areas of interest. AMRClaw was used in [24] to model the propagation of a transoceanic tsunami, where the behaviour is governed by nonlinear hyperbolic equations. Adaptive mesh refinement allowed for modelling of both the large-scale properties of the oceanic regions as well as the comparatively small-scale behaviour in coastal areas.

Chombo [36] is a C++/Fortran library providing finite difference and finite volume codes and related tools for solving PDEs on block-structured meshes using AMR and has a focus on supporting large-scale parallelization. Chombo was used in [29] to evaluate AMR performance for modeling antarctic ice dynamics. AMR algorithms were employed to dynamically generate new meshes as the modeled ice sheet evolves and reduces the mesh cell count as well as total computational effort by an order of magnitude compared to a single, uniform mesh with the same accuracy requirements.

The *SAMRAI* C++ library [30] contains tools for developing software with AMR functionality. SAMRAI focuses especially on integrability with existing codebases and usage in high-performance massively parallel computing. In [14], the SAMRAI library provided AMR capability for simulation of laser plasma interaction. The use of AMR reduced cell count to approximately one-third compared to a uniform mesh, resulting in significantly faster run-times and lower memory cost.

A more specific use-case are level-set methods, numerical methods for calculating the time evolution of geometric interfaces, which are represented as the 0-level-set of a suitable level-set function. As high accuracy is usually only required near the interface, AMR provides an intuitive way to reduce computational effort while achieving the desired accuracy [34]. A level-set based numerical method together with AMR was presented in [33], allowing for simulation of two-phase flows on the scales of individual bubbles by selectively refining only the areas around the interface between liquid and gas.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

This thesis focuses mainly on the creation of the nested mesh hierarchy, presuming the areas to-be-refined are already known. The goal is to develop an optimized, parallelizable mesh hierarchy generation algorithm and to provide a C++ AMR implementation for use with existing, well-tested and optimized numerical code. While mesh hierarchies for AMR in general have to fulfil certain *nesting criteria*, these numerical solvers pose additional constraints on the mesh hierarchies. Particularly the new requirement for unique parent meshes allows for further performance optimizations which would otherwise not be possible. Current mesh generation methods [6, 25], as well as their concrete implementations [30, 32, 36], are insufficient as they do not generally adhere to the extended nesting criteria.

Based on the widely used Berger-Rigoutsos signature clustering algorithm [6], a new algorithm for creating and transforming nested mesh hierarchies is developed and optimized for mesh quality. This algorithm is evaluated and compared to a tile-based algorithm, adapted from [25]. Unlike the original algorithms, both novel algorithms are guaranteed to fulfill the extended nesting criteria.

The AMR refinement procedure, including mesh management, initialization, synchronization and averaging, as well as the optimized mesh generation algorithms are implemented as a C++11 library, with OpenMP used for shared-memory parallelization. Also, Python 3.7 is used for post-processing the data for evaluation and visualization. The 3D plots are created using ParaView 5.4.1.

1.2 Outline

In Section 2, the different types of AMR are discussed and related terminologies and fundamental concepts are introduced.

In Section 3, the algorithms used for clustering the flagged mesh cells on a single level and on multiple levels are presented.

In Section 4, a brief overview of the AMR implementation including the mesh hierarchy generation algorithms is given.

In Section 5, the results from the clustering algorithms as well as a complete simulation example are examined.

In Section 6, the work is summarized and directions for possible future investigations are discussed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2 Adaptive Mesh Refinement

Adaptive mesh refinement is a numerical technique for optimizing the solution process of PDEs which allows efficiently obtaining higher accuracy by refining the spatial resolution only where needed. Starting from a coarsely resolved root mesh, areas in need of further refinement are identified using error estimation techniques, properties of the solution or intrinsic characteristics of the underlying physical problem. Additional mesh nodes are then placed accordingly. By identifying and selectively refining areas of interest, AMR allows accurate simulation even of features much smaller than the domain size, where a single uniform mesh would require an unfeasible computational effort. AMR can be applied to various geometries and mesh types, depending on the requirements of the numerical methods used as well as desired performance characteristics, especially with respect to parallelization [19].

The term AMR is used for mesh refinement in conjunction with both structured and unstructured meshes; this work focuses on the first, where a distinction is made between three types of AMR for structured meshes: Cell-based AMR (cf. Figure 2.1), patch-based AMR (cf. Figure 2.2) and tree-based AMR (cf. Figure 2.3). The terminology is not completely unambiguous in the literature, as the term block-based AMR is sometimes also used to describe tree-based AMR algorithms [27].

Initially, structured AMR was devised to solve hyperbolic differential equations with finite difference methods [5, 8] but was later extended to work with parabolic [17] and elliptic [33] differential equations and also used in the context of adaptive multigrid methods [13, Ch. 9]. The hyperbolic case is more straightforward, since the explicit methods used only require local mesh data for time integration. To enable the solution of elliptic differential equations, an iterative domain-decomposition approach is used [2, 33].

The basic structured-AMR procedure is:

First, the given differential equation is solved on a single root mesh with specified boundary conditions. Then, areas requiring refinement are determined and marked (by error approximation or other methods, see Section 2.2.2) and new meshes are created that cover the marked areas. This process may continue iteratively until a refinement criterion is satisfied and no new meshes are required.

New mesh data is either interpolated from a less refined (coarser) mesh or copied from previously existing meshes of the same refinement level. The meshes are advanced in time separately, with stepsizes dependent on their refinement level. After each step meshes are synchronized, their boundary values are exchanged with neighboring meshes or interpolated from coarser meshes.

The cost of creating and adapting the nested mesh hierarchy, including initialization and synchronization overheads, is usually small compared to the savings resulting from the greatly reduced amount of mesh cells (cf. Definition 2.1) achieved by refining the spatial resolution only locally.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Cell-based AMR

In cell-based AMR, all cells are refined individually [7, 9]. This is usually realized as a tree-structure of, e.g., quad-trees or oct-trees. This has the benefit of perfect refinement efficiency (since only the cells needed are refined) but has high memory requirements for the hierarchical structure and synchronization of the refined cells is complicated – the main benefit of structured meshes is not having to keep track of the (relative) positions of individual cells, yet cell-based AMR requires some way of describing cell connectivity which poses a significant computational overhead.

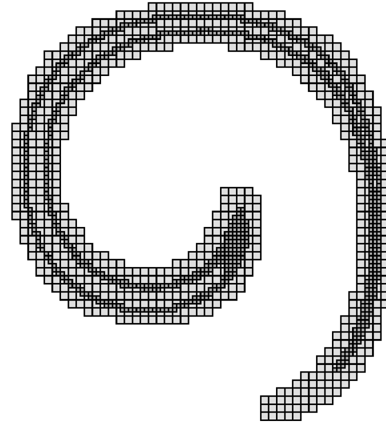


Figure 2.1: Cell-based refinement.

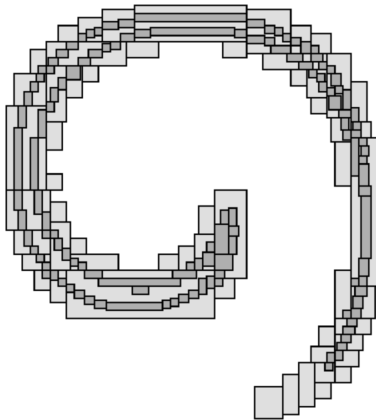


Figure 2.2: Patch-based refinement.

Patch-based AMR

As an alternative to refining cells on an individual level, patch-based AMR (or block-based AMR) [1] groups cells of the same refinement level together. These rectangular patches are then refined and may in turn contain multiple patches. This means that while some cells are refined unnecessarily, the overall structure is simpler since less separate meshes are generated and only connectivity between patches and not between individual cells has to be considered. If rectangular, axis aligned patches are used, the same numerical integrators can be used on every patch, improving performance significantly.

Tree-based AMR

Tree based AMR is very similar to cell-based AMR in that a quad-/oct-tree structure is used for refinement. However, refinement is not done on single cells but on cell-blocks of predefined sizes [26, 28]. This retains the benefits of structured meshes for each block and results in very simple data structures. However, mesh efficiency may be low, since blocks are refined together even if only small parts actually require higher resolutions, resulting in significant over-refinement. Tree-based AMR can be understood as a special case of patch-based AMR with only equally sized patches, but implementations usually differ in how the patches and their connectivities are represented internally.

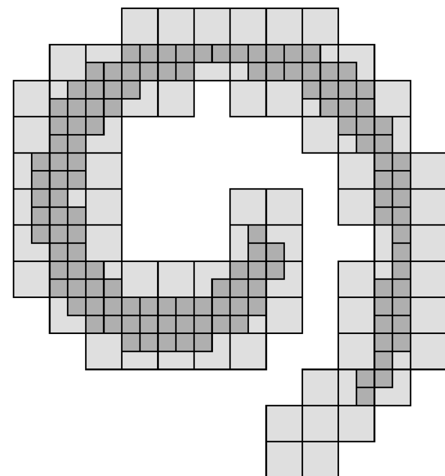


Figure 2.3: Tree-based refinement.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Between the cell-based, patch-based and tree-based methods, a trade-off has to be made with regards to number of nodes (mesh efficiency) and number of mesh patches (for cell-based AMR, each refined cell is treated as a single mesh patch). Patch-based AMR represents a suitable compromise, especially since mesh characteristics are controllable by the clustering algorithm.

However, patch-based AMR has certain drawbacks [23], such as an increased difficulty in parallelization: In block-based methods, all meshes of a refinement level have the same size, whereas the patch-sizes may differ greatly. Also, patch-based AMR does not preserve physical symmetry without additional effort, which may be required for physically-accurate simulation of certain problems.

In this thesis, a hierarchy of increasingly finer resolved, axis-aligned but not necessarily identically-sized Cartesian meshes (called patches) is investigated, where each refinement level is represented by a union of non-intersecting patches of the same spatial resolution. This patch-based approach is analyzed and an optimized AMR algorithm is devised.

The use of rectangular, axis-aligned meshes has the additional benefit of allowing the use of the same integration algorithm for all meshes, regardless of refinement level, resolution or size. Owing to the simple mesh structures, this enables the use of optimized and highly parallelizable algorithms with provable convergence properties.

From an implementation perspective, this also makes it possible to cleanly decouple the AMR algorithm from the numerical solver, allowing for the use of existing, well-tuned integrators and reuse of the same AMR code in different problem domains. The process of creating this mesh hierarchy is conceptually simple, yet poses several challenges when it comes to an optimal and efficient implementation.

TU Bibliothek WIEN Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

2.1 Patch-Based AMR

Patch-based mesh refinement uses rectangular Cartesian meshes where refinement is done on rectangular boxes (called **patches**, as in Definition 2.1) that may contain further refined meshes themselves.

Definition 2.1 (Cartesian Grid, Mesh and Patch)**.**
A domain $\Omega = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$ is discretized by a Cartesian grid with a global indexing scheme $I := \{(i, j, k) \in \mathbb{N}_0^{<n_x} \times \mathbb{N}_0^{<n_y} \times \mathbb{N}_0^{<n_z}\}$ for a given grid size $\mathbf{n} \in \mathbb{N}^3$.
The uniform Cartesian **grid** $\mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n})$ is defined as

$$\mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n}) := \left\{ \mathbf{x}_{ijk} := \mathbf{a} + \begin{pmatrix} \frac{i+1/2}{n_x}(b_x - a_x) \\ \frac{j+1/2}{n_y}(b_y - a_y) \\ \frac{k+1/2}{n_z}(b_z - a_z) \end{pmatrix} \middle| \; 0 \le i < n_x,\; 0 \le j < n_y,\; 0 \le k < n_z \right\}.$$

While the full discretization of the domain is called grid, the parts of a grid actually containing data values will be referred to as mesh to avoid ambiguity.
For a grid $\mathcal{G}_\Omega = \mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n})$, a **patch** is a rectangular sub-region $P \subseteq \Omega$ defined by its first index $\mathbf{f} \in \mathbb{N}_0^3$ and size $\mathbf{s} \in \mathbb{N}^3$

$$\begin{aligned} P(\mathbf{f}, \mathbf{s}) := & \left[a_x + \frac{f_x}{n_x}(b_x - a_x),\; b_x + \frac{f_x + s_x}{n_x}(b_x - a_x) \right] \\ & \times \left[a_y + \frac{f_y}{n_y}(b_y - a_y),\; b_y + \frac{f_y + s_x}{n_y}(b_y - a_y) \right] \\ & \times \left[a_z + \frac{f_z}{n_z}(b_z - a_z),\; b_z + \frac{f_z + s_x}{n_z}(b_z - a_z) \right]. \end{aligned}$$

All index tuples belonging to P are then defined as

$$\begin{aligned} I_x(P) &:= \{f_x,\; f_x + 1,\; \dots,\; f_x + s_x - 2,\; f_x + s_x - 1\}, \\ I_y(P) &:= \{f_y,\; f_y + 1,\; \dots,\; f_y + s_y - 2,\; f_y + s_y - 1\}, \\ I_z(P) &:= \{f_z,\; f_z + 1,\; \dots,\; f_z + s_z - 2,\; f_z + s_z - 1\}, \\ I(P) &:= I_x(P) \times I_y(P) \times I_z(P). \end{aligned}$$

The **mesh** M corresponding to the patch $P(\mathbf{f}, \mathbf{s})$ is given by

$$\mathcal{M}(\mathbf{f}, \mathbf{s}) := P(\mathbf{f}, \mathbf{s}) \cap \mathcal{G}_\Omega.$$

The elements of the mesh, the **mesh nodes** $\mathbf{x}_{ijk} \in M$, are the centers of the respective **mesh cells** $m_{ijk} := P((i, j, k), (1, 1, 1))$, which together make up the patch

$$P(\mathbf{f}, \mathbf{s}) = \bigcup_{(i,j,k) \in I(P)} m_{ijk}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A mesh is comprised of individual mesh cells for which data values are stored at the respective mesh nodes (cf. Figure 2.4). Areas of interest are marked for refinement by flagging the corresponding mesh cells. Note that mesh nodes are defined as the center of their cells and not as the edge points. While AMR can also be used with standard meshes, the cell-based approach is better suited for use with numerical methods for hyperbolic problems, such as finite volume methods.

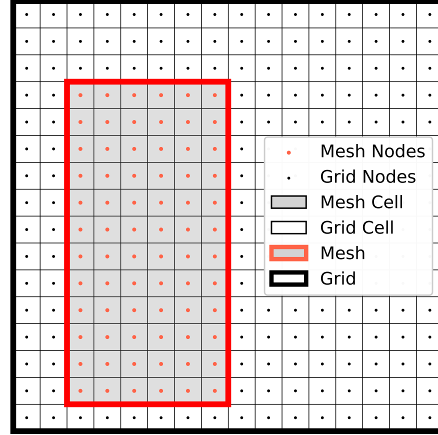


Figure 2.4: Grid nodes are defined as the centers of the corresponding grid cells, covering the whole discrete domain. Mesh cells only cover the parts of the grid for which data is stored.

With respect to the implementation, each mesh not only contains its **inner cells**, as defined above, but is enlarged to include a number of **ghost cells** describing its boundary conditions (cf. Section 2.2.5).

With this in mind, the patch-based hierarchical structure is formally defined.

Definition 2.2 (Patch-based Nested Mesh Hierarchy)**.**
A patch-based nested mesh hierarchy of the physical domain $\Omega := [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z]$ with refinement ratio $r \in \mathbb{N}_{\geq 2}$ is a set of **levels**

$$\mathcal{L}^{(l)} = \left(\mathcal{G}^{(l)}, \mathcal{M}^{(l)}, \mathcal{P}^{(l)}, \mathcal{R}^{(l)}\right), \quad l \in \{0, \ldots, q\},$$

where q is the maximum depth. Each **level** $\mathcal{L}^{(l)}$ consists of a grid $\mathcal{G}^{(l)}$, a set of meshes $\mathcal{M}^{(l)}$, a set of patches $\mathcal{P}^{(l)}$ and a **refinement function** $\mathcal{R}^{(l)}$.

The root grid $\mathcal{G}^{(0)} := \mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n}^{(0)})$ defines each level's grid size $\mathbf{n}^{(l)} := \mathbf{n}^{(0)} r^{(l)}$ as well as its grid spacing

$$\mathbf{h}^{(l)} := \left(\frac{b_x - a_x}{n_x^{(l)}}, \frac{b_y - a_y}{n_y^{(l)}}, \frac{b_z - a_z}{n_z^{(l)}}\right).$$

Starting from the coarsest (or *highest*) level, the root level $\mathcal{L}^{(0)}$, descending the hierarchy gradually decreases the level's grid spacing until the finest (or *lowest*) level $\mathcal{L}^{(q)}$ is reached.

For ease of notation, a global indexing scheme based on the grid hierarchy

$$\{\mathcal{G}^{(l)} = \mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n}^{(l)}) : 0 \leq l \leq q\}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

is used. The index $(i,j,k)^{(l)}$ refers to the cell with center

$$\mathbf{x}_{ijk}^{(l)} := \mathbf{a} + \begin{pmatrix} \frac{i+1/2}{n_x^{(l)}}(b_x - a_x) \\ \frac{j+1/2}{n_y^{(l)}}(b_y - a_y) \\ \frac{k+1/2}{n_z^{(l)}}(b_z - a_z) \end{pmatrix}$$

on level $\mathcal{L}^{(l)}$.

The root level contains only the base domain mesh $\mathcal{M}^{(0)} := \mathcal{G}^{(0)} = \mathcal{G}(\mathbf{a}, \mathbf{b}, \mathbf{n}^{(0)})$, which covers the full computational domain. Every other level $\mathcal{L}^{(l)}$, $1 \le l \le q$ contains $p^{(l)} \in \mathbb{N}^{(0)}$ patches and the corresponding meshes

$$\mathcal{P}^{(l)} := \{P(\mathbf{f}_m^{(l)}, \mathbf{s}_m^{(l)}), \quad 1 \le m \le p^{(l)}\}, \quad \mathcal{M}^{(l)} := \{M(\mathbf{f}_m^{(l)}, \mathbf{s}_m^{(l)}), \quad 1 \le m \le p^{(l)}\}.$$

All levels $\mathcal{L}^{(l)}$ include a refinement function defined on the respective grid cell centers

$$\mathcal{R}^{(l)} : \mathcal{G}^{(l)} \to \{0,1\} \text{ with } \mathcal{R}_{ijk}^{(l)} := \mathcal{R}(\mathbf{x}_{ijk}^{(l)}),$$

that marks which cells need to be refined, i.e., $\forall \mathbf{x} \in \mathcal{G}^{(l)}, \mathcal{R}(\mathbf{x}) = 1 \implies \mathbf{x} \in \mathcal{P}^{(l+1)}$.

An AMR algorithm starts of with a rectangular, uniform Cartesian grid and, based on a predictive error estimation or other indicators, certain cells are marked for refinement (*flagged*). These cells are suitably clustered into patches using a clustering algorithm. The refined meshes can in turn be marked and refined, creating a hierarchical structure according to Definition 2.2.

The method was first used to solve shock hydrodynamics problems in [5] where additional constraints are imposed on the subgrids, greatly simplifying the numerical calculations and allowing the use of specialized, more efficient data structures [3]. Recent research further improves on patch-based AMR in regards to performance and scalability [30].

An important aspect, especially considering modern computer architectures, is parallelization. Various parallel adaptations of the base AMR algorithm exist, both for shared-memory and distributed-memory architectures [18, 27].

An optimized AMR algorithm should produce a mesh hierarchy with a minimal number of meshes and cells while still covering all cells marked for refinement. This necessitates a trade-off between mesh efficiency (ratio of refined cells to marked cells) and number of meshes. What constitutes an optimal mesh hierarchy is problem-dependent and also depends on factors beyond the AMR algorithm, such as the performance characteristics of the numerical solver used on the mesh or whether parallelization is desired on a distributed computing architecture.

Classic block-structured patch-based AMR, as discussed in [5], uses rectangular axis-aligned patches which are required to be aligned to the next coarser level and may not differ in refinement more than 1 level from neighboring patches. In this work, additional conditions are imposed on the mesh hierarchy, stemming from compatibility requirements with existing code as well as to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

allow for more efficient numerical treatment: A valid nested mesh hierarchy has to adhere to the nesting criteria given in Definition 2.3.

While it may be possible to relax these conditions, it would require development of new, more complex and computationally intensive solvers. Investigations in this direction are not in the scope of this thesis, the nesting criteria are taken as given.

Definition 2.3 (Nesting Criteria)**.**
A nested mesh $(\mathcal{L}^{(l)})_{l=0}^{q}$ satisfies the nesting criteria if its patches $(\mathcal{P}^{(l)})_{l=0}^{q}$ satisfy

$$\forall k < l: \quad \mathcal{P}^{(l)} \cap \partial\mathcal{P}^{(k)} = \emptyset, \tag{N1}$$

$$\forall P, \widetilde{P} \in \mathcal{P}^{(l)}: \quad P^{\circ} \cap \widetilde{\mathcal{P}}^{\circ} = \emptyset, \tag{N2}$$

$$\forall P \in \mathcal{P}^{(l)}: \quad P_{\mathbf{f}} \bmod r = \mathbf{0} \;\wedge\; P_{\mathbf{s}} \bmod r = \mathbf{0}, \tag{N3}$$

$$\forall P \in \mathcal{P}^{(l)}: \quad \exists!\; \widetilde{P} \in \mathcal{P}^{(l-1)} : P \subset \widetilde{P}, \tag{N4}$$

$$\forall P \in \mathcal{P}^{(l)}: \quad P_x \geq m_0 \wedge P_y \geq m_0 \wedge P_z \geq m_0 \tag{N5}$$

for all levels $l \in \{1, \ldots, q\}$.
This means that neighboring patches differ by 0 or 1 levels (N1), patches do not overlap on the same hierarchy level (N2), patches are aligned to the parent grid (N3) and every patch is covered by exactly one parent patch of the next coarser level (N4). Additionally, all patches' sides are required to be equal to or larger than a given minimum width m_0 (N5).

Early AMR implementations (e.g., [5]) only require criteria (N2) and (N4). In later clustering approaches [6], (N2) was also added. Still, existing algorithms do not generate mesh hierarchies where each mesh is guaranteed to have one unique parent. The minimum width requirement (N5) is usually not a strict criterion, but rather a termination condition for the clustering algorithm (smaller patches are allowed if necessary to ensure (N1)-(N4)); here, all patches are required to be larger than the minimum size. The algorithms presented in this thesis are modified in a way that guarantees (N1)-(N5) while still producing efficient mesh clusterings. Figure 2.5 shows an example grid with multiple patches, some conforming to and some violating the nesting criteria from Definition 2.3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

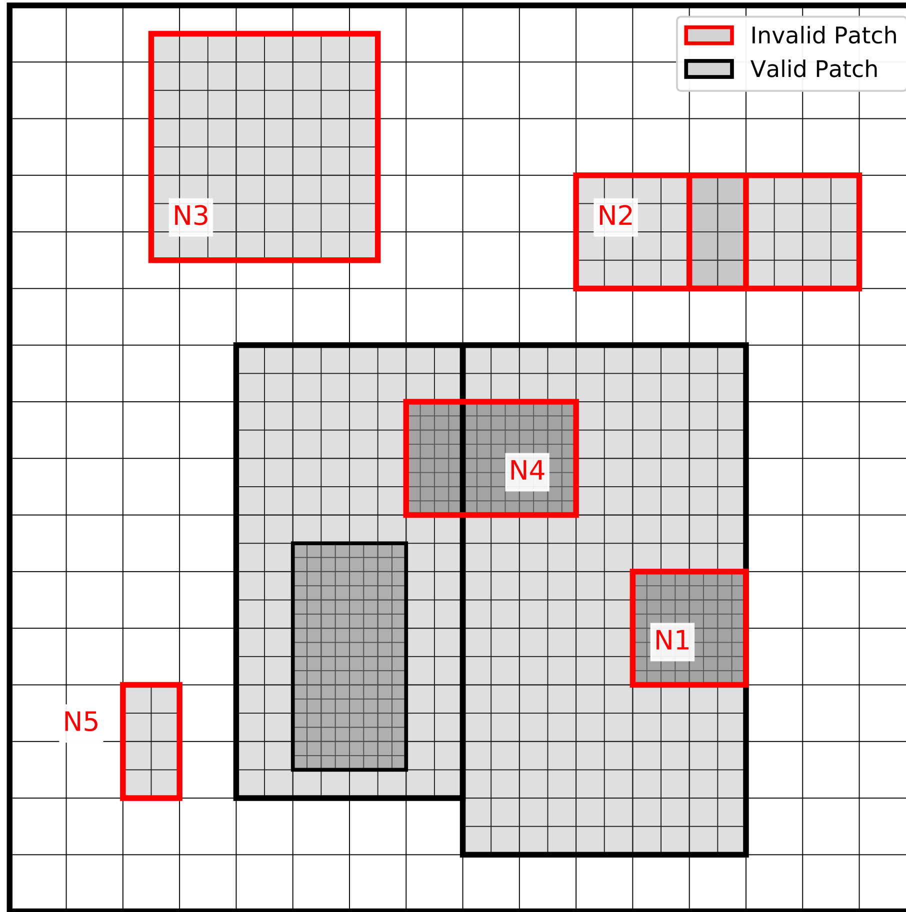


Figure 2.5: The nesting criteria forbid patches that border an under-refined area (N1), overlapping patches (N2), non-aligned patches (N3), patches that are not contained in a single parent patch (N4) and patches that are smaller than the minimum width (N5) (in this case $m_0 = 4$).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2 The AMR Algorithm

In this section the core AMR algorithm, as introduced in [8], is described. The AMR algorithm is used to solve general hyperbolic equations of the form

$$\begin{aligned} u_t + \nabla \cdot f(u) &= 0 \qquad && \text{on } \Omega \times \mathbb{R}^+, \\ u(\,\cdot\,, 0) &= u_0 && \text{on } \Omega, \end{aligned} \tag{1}$$

with suitable boundary conditions on a domain $\Omega = [a_x, b_x] \times [a_y, b_y] \times [a_z, b_z] \subset \mathbb{R}^3$.

The AMR algorithm is split into 6 main parts:

- **Timestepping**: Advance meshes forward in time.
- **Flagging**: Mark cells to be refined.
- **Regridding**: Cluster marked cells into patches.
- **Initialization**: Initialize new meshes, interpolate or copy mesh data.
- **Synchronization**: Exchange mesh boundary data with neighboring meshes.
- **Averaging**: Propagate the solution from fine to coarse by averaging the mesh values.

2.2.1 Timestepping

The AMR algorithm does not itself contain an integrator and can be used with different kinds of numerical methods depending on the specific application. Here, the usage of an explicit, cell-centered numerical scheme (see e.g. [10])

$$u^{n+1} = u^n + \Phi(u^n,\, f,\, k,\, h), \tag{2}$$

with temporal stepsize k and spatial stepsize h, is assumed. Discussion of the numerical methods will be kept short in order to focus on the creation, management and analysis of the mesh hierarchy. A rigorous treatment of numerical methods for solving hyperbolic equations and the intricacies of using these within the AMR algorithm is given in [5, 8, 16].

AMR is also used in conjunction with level-set methods ([12, 15, 34]), which require the solution of the transport equation

$$\begin{aligned} u_t + \mathbf{v} \cdot \nabla u &= 0, \\ u_t(\,\cdot\,, 0) &= u_0, \end{aligned} \tag{3}$$

where $u : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}$ is the level-set function describing an interface which is located at the 0-level-set $\Gamma_t = \{\mathbf{x} \in \Omega : u(\mathbf{x}, t) = 0\}$. Its time evolution is given by a velocity field $\mathbf{v} : \mathbb{R}^3 \times \mathbb{R}^+ \to \mathbb{R}^3$. Usually, the initial level-set function u_0 is taken as the signed distance to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

the initial interface Γ_0

$$u_0(\mathbf{x}) = d_\pm(\mathbf{x}, \Gamma_0), \quad \mathbf{x} \in \Omega.$$

The numerical solution method is independent of the AMR algorithm and not the focus of this thesis. For timestepping the numerical examples of type (3), a dimensionally-split three-dimensional (3D) upwind scheme is used [16, Ch. 9]

$$\begin{aligned}
u_{ijk}^{n+1/3} &= u_{ijk}^{n} - \frac{k}{h}\left(\max(v_x,0)\left(u_{ijk}^{n} - u_{(i-1)jk}^{n}\right) + \min(v_x,0)\left(u_{(i+1)jk}^{n} - u_{ijk}^{n}\right)\right), \\
u_{ijk}^{n+2/3} &= u_{ijk}^{n+1/3} - \frac{k}{h}\left(\max(v_y,0)\left(u_{ijk}^{n+1/3} - u_{i(j-1)k}^{n+1/3}\right) + \min(v_y,0)\left(u_{i(j+1)k}^{n+1/3} - u_{ijk}^{n+1/3}\right)\right), \qquad (4) \\
u_{ijk}^{n+1} &= u_{ijk}^{n+2/3} - \frac{k}{h}\left(\max(v_z,0)\left(u_{ijk}^{n+2/3} - u_{ij(k-1)}^{n+2/3}\right) + \min(v_z,0)\left(u_{(ij(k+1)}^{n+2/3} - u_{ijk}^{n+2/3}\right)\right).
\end{aligned}$$

The AMR algorithm advances the whole hierarchy in time. In order to retain the desired accuracy as well as for stability reasons, the temporal stepsize for each level is dependent on the spatial stepsize, usually in the form of a Courant–Friedrichs–Lewy (CFL) condition dependent on the numerical method used, in our case

$$k^{(l)} \leq \frac{h^{(l)}}{\sup\limits_{\mathbf{x}\in\Omega}\left(\|\mathbf{v}(\mathbf{x})\|_\infty\right)}, \quad 0 \leq l \leq q.$$

This implies that finer grids need to be advanced r-times more often than coarser grids (cf. Figure 2.6). For each level $\mathcal{L}^{(l)}$, timestepping is straightforward: Advance all meshes forward by $k^{(l)}$. The boundary conditions for each non-root mesh are taken from either neighboring meshes or interpolated from the parent mesh (cf. Section 2.2.5).

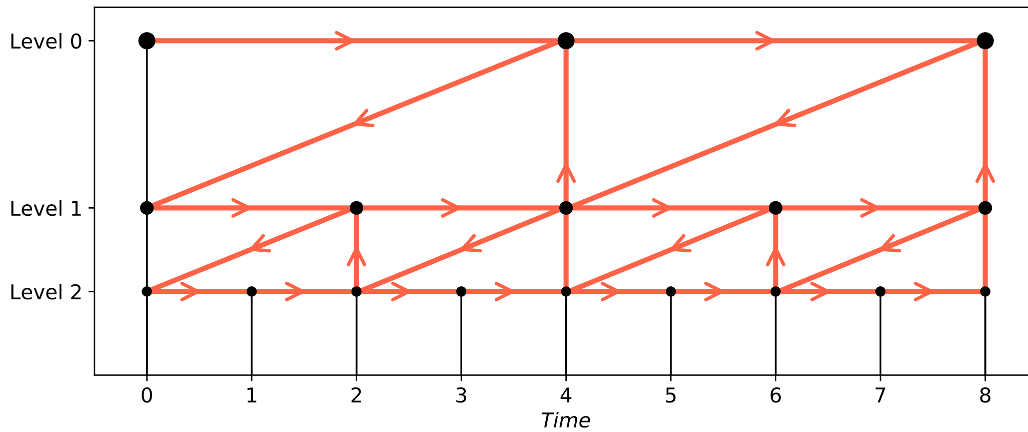


Figure 2.6: Timestep schedule for a hierarchy of 3 levels and refinement factor $r = 2$. First, the coarser level is advanced, followed by r steps of the next finer level. This process is repeated recursively for all levels, followed by an update of the coarser grids with the more accurate values from below.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

2.2.2 Flagging

The AMR algorithm achieves better accuracy with lower computational effort compared to uniform grids by selectively refining areas of interest. This necessitates accurate identification of what areas contribute most to the global error.

Often this is done by exploiting inherent properties of the numerical integration algorithm in order to obtain estimates of the local truncation errors using Richardson-extrapolation on the already existing meshes [8]. Areas exhibiting errors larger than the given error tolerance are then flagged for refinement. Depending on the magnitude of the error, it is possible to require refinement by multiple levels at once.

Other options for marking cells for refinement are based on the flow gradient, characteristics of the solution, e.g., curvature [17] or divergence or, in the case of level-set methods, a given distance from the interface, the 0-level-set.

Here, a simple flagging scheme based on the absolute cell values is used. A cell at level $\mathcal{L}^{(l)}$ is flagged if its absolute value exceeds a predefined, level-dependent threshold $c^{(l)}$

$$\mathcal{R}^{(l)}(x_{ijk}^{(l)}) := \begin{cases} 1, & \text{if } |u_{ijk}^{(l)}| > c^{(l)}, \\ 0, & \text{else.} \end{cases} \tag{5}$$

In the level-set case, this means that all cells $x_{ijk}^{(l)}$ less than $c^{(l)}$ from the interface are marked for refinement (cf. Figure 2.7).

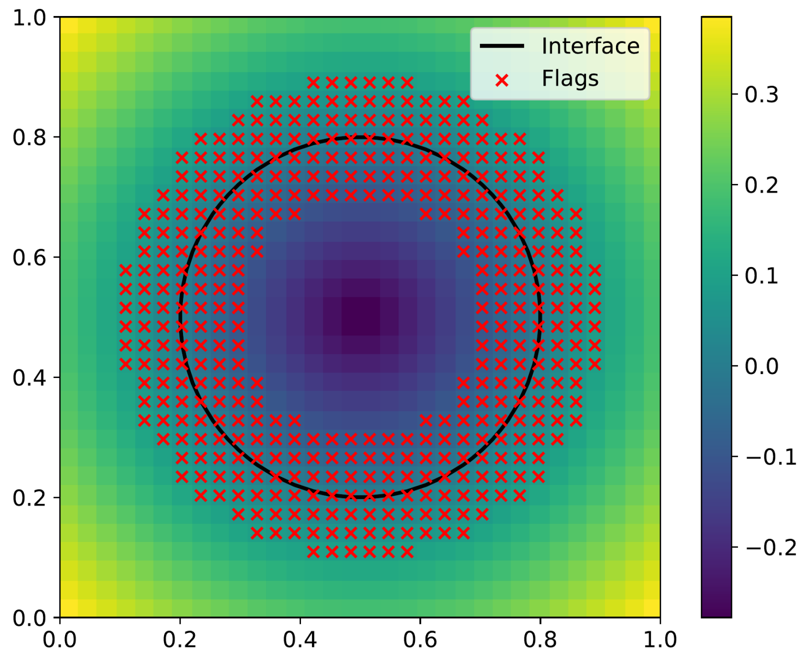


Figure 2.7: Area of interest around a circular interface with center (0.5, 0.5) and radius 0.3. All cells with a distance less than 0.1 from the interface are marked.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

2.2.3 Regridding

Given an existing mesh hierarchy and a set of refinement flags, an adapted mesh hierarchy needs to be created that covers all the newly marked cells and fulfills the nesting criteria, while if possible only changing the lower (i.e., finer) levels of the hierarchy. Methods to achieve this are discussed in detail in Section 3.

It is not obvious when or how often to perform the regridding process. If the error estimation and regridding procedure is not done often enough, the area of interest may move outside of the refined area, resulting in a loss of accuracy.

Here, regridding is performed every s-th timestep, where s is a user-definable parameter. In general, the optimal regridding period is dependent on the speed with which the area of interest moves. As an alternative to regridding more often, it is possible to apply an additional buffer around existing flags in order to preemptively refine the cells around the area of interest.

2.2.4 Initialization

During the regridding process, new meshes are created and need to be initialized with the correct data values. Depending on their location, data for the new meshes $\left(\mathcal{M}^{(l)}\right)_{l=1}^{q}$ will need to be either copied over from previous meshes $\left(\widetilde{\mathcal{M}}^{(l)}\right)_{l=1}^{q}$ or interpolated from the respective parent meshes (cf. Figure 2.8).

Remark 2.4 (Interpolate Mesh Data)**.**
If a mesh (or parts of a mesh) of level l, $0 < l \leq q$ was not previously part of a refined region, i.e., $\mathcal{M}^{(l)} \setminus \widetilde{\mathcal{M}}^{(l)} \neq \emptyset$, all cell data $u_{ijk}^{(l)}$ not covered are trilinearly interpolated from the parent mesh

$$\begin{aligned}
&\forall (i,j,k) : \mathbf{x}_{ijk}^{(l)} \in \mathcal{M}^{(l)} \setminus \widetilde{\mathcal{M}}^{(l)} : \\
&\tilde{\imath} := \left\lfloor \frac{i}{r} \right\rfloor, \ \tilde{\jmath} := \left\lfloor \frac{j}{r} \right\rfloor, \ \tilde{k} := \left\lfloor \frac{k}{r} \right\rfloor, \\
&\bar{\imath} := \frac{i}{r} - \tilde{\imath}, \ \bar{\jmath} := \frac{j}{r} - \tilde{\jmath}, \ \bar{k} := \frac{k}{r} - \tilde{k},
\end{aligned}$$

$$\begin{aligned}
u_{ijk}^{(l)} = I_{ijk}(u^{(l-1)}) := \; & (1-\bar{\imath})\,(1-\bar{\jmath})\,(1-\bar{k})\; u_{\tilde{\imath}\tilde{\jmath}\tilde{k}}^{(l-1)} && + \bar{\imath}\,(1-\bar{\jmath})\,(1-\bar{k})\; u_{(\tilde{\imath}+1)\tilde{\jmath}\tilde{k}}^{(l-1)} \\
& + (1-\bar{\imath})\,\bar{\jmath}\,(1-\bar{k})\; u_{\tilde{\imath}(\tilde{\jmath}+1)\tilde{k}}^{(l-1)} && + (1-\bar{\imath})\,(1-\bar{\jmath})\,\bar{k}\; u_{\tilde{\imath}\tilde{\jmath}(\tilde{k}+1)}^{(l-1)} \\
& + \bar{\imath}\,\bar{\jmath}\,(1-\bar{k})\; u_{(\tilde{\imath}+1)(\tilde{\jmath}+1)\tilde{k}}^{(l-1)} && + \bar{\imath}\,(1-\bar{\jmath})\,\bar{k}\; u_{(\tilde{\imath}+1)\tilde{\jmath}(\tilde{k}+1)}^{(l-1)} \\
& + (1-\bar{\imath})\,\bar{\jmath}\,\bar{k}\; u_{\tilde{\imath}(\tilde{\jmath}+1)(\tilde{k}+1)}^{(l-1)} && + \bar{\imath}\,\bar{\jmath}\,\bar{k}\; u_{(\tilde{\imath}+1)(\tilde{\jmath}+1)(\tilde{k}+1)}^{(l-1)}.
\end{aligned}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Remark 2.5 (Copy Mesh Data)**.**
If a mesh (or parts of a mesh) of level $l,\ 0 < l \leq q$ were previously part of a refined region, i.e., $\mathcal{M}^{(l)} \cap \widetilde{\mathcal{M}}^{(l)} \neq \emptyset$, all cell data $u_{ijk}^{(l)}$ that was previously covered is copied

$$\forall (i,j,k) : x_{ijk}^{(l)} \in \mathcal{M}^{(l)} \cap \widetilde{\mathcal{M}}_l : \quad u_{ijk}^{(l)} := \tilde{u}_{ijk}^{(l)}.$$

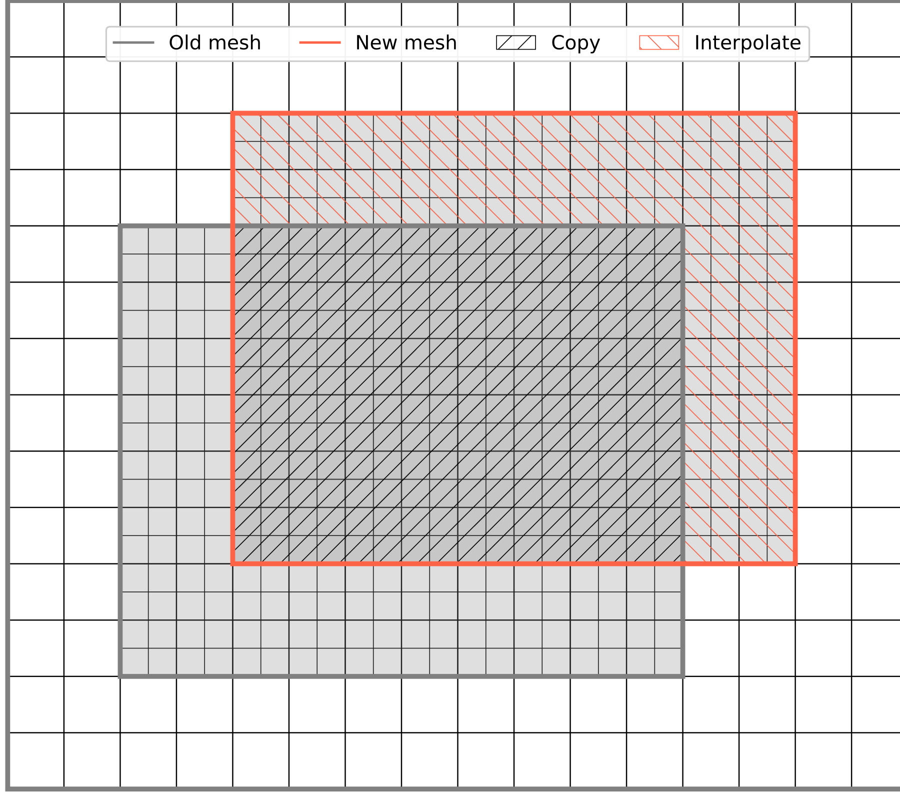


Figure 2.8: Initialization of a new mesh: The existing data is copied from the old mesh, values for the locations outside the previously refined area are interpolated from the parent mesh.

2.2.5 Synchronization

Considering, for example, the numerical solution of the differential equation

$$\begin{aligned} u_t + \nabla \cdot f(u) &= 0 \quad \text{on } P(\mathbf{f},\mathbf{s}) \times \mathbb{R}^+, \\ u &= u_\partial \quad \text{on } \partial P(\mathbf{f},\mathbf{s}) \times \mathbb{R}^+, \end{aligned}$$

on a patch $P(\mathbf{f},\mathbf{s})$ corresponding to the mesh $M(\mathbf{f},\mathbf{s})$, suitable boundary conditions u_∂ are required. For the root mesh, these are externally given. For all refined meshes $M \in \mathcal{M}^{(l)}$, $0 < l \leq q$, an additional boundary layer of cells, called **ghost cells**, is stored

$$\partial M := \{\mathbf{x} \in \mathcal{G}^{(l)} : \mathbf{x} \notin M,\ d(\mathbf{x}, M)_\infty \leq g\}.$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The size g of the boundary level depends on the number of neighboring cells the numerical integration method requires; in the case of the upwind method (4), the ghost cell layer measures one cell in each spatial direction, $g = 1$. After timestepping all meshes of a level, the boundary conditions need to be regenerated from neighboring meshes or, if located at the border between coarse and fine meshes, interpolated from the parent (cf. Figure 2.9).

Note that using a dimensionally-split scheme such as (4) requires integration of ghost cells for the intermediate steps. This greatly increases the performance overhead of small patches with relatively large ratio of ghost cells to inner cells.

Remark 2.6 (Synchronization)**.**
Boundary value data is either copied from adjacent meshes or interpolated trilinearly from the parent. The refined grid may still be at an earlier *time* than the coarser grid, $t_{n-1}^{(l-1)} \leq t_n^{(l)} \leq t_n^{(l-1)}$. In such a case, the ghost cell values are also interpolated linearly in time

$$\forall (i,j,k) : \mathbf{x}_{ijk}^{(l)} \in \partial M \cap \mathcal{M}^{(l)} \ : \ u_{ijk}^{(l)} = u_{ijk}^{(l)},$$

$$\forall (i,j,k) : \mathbf{x}_{ijk}^{(l)} \in \partial M \setminus \mathcal{M}^{(l)} \ : \ u_{ijk}^{(l)} = \frac{t_n^{(l-1)} - t_n^{(l)}}{t_n^{(l-1)} - t_{n-1}^{(l-1)}} I_{ijk}(u_{n-1}^{(l-1)}) + \frac{t_n^{(l)} - t_{n-1}^{(l-1)}}{t_n^{(l-1)} - t_{n-1}^{(l-1)}} I_{ijk}(u_n^{(l-1)}).$$

Nesting criterion (N1) guarantees that interpolation from the parent mesh is always possible.

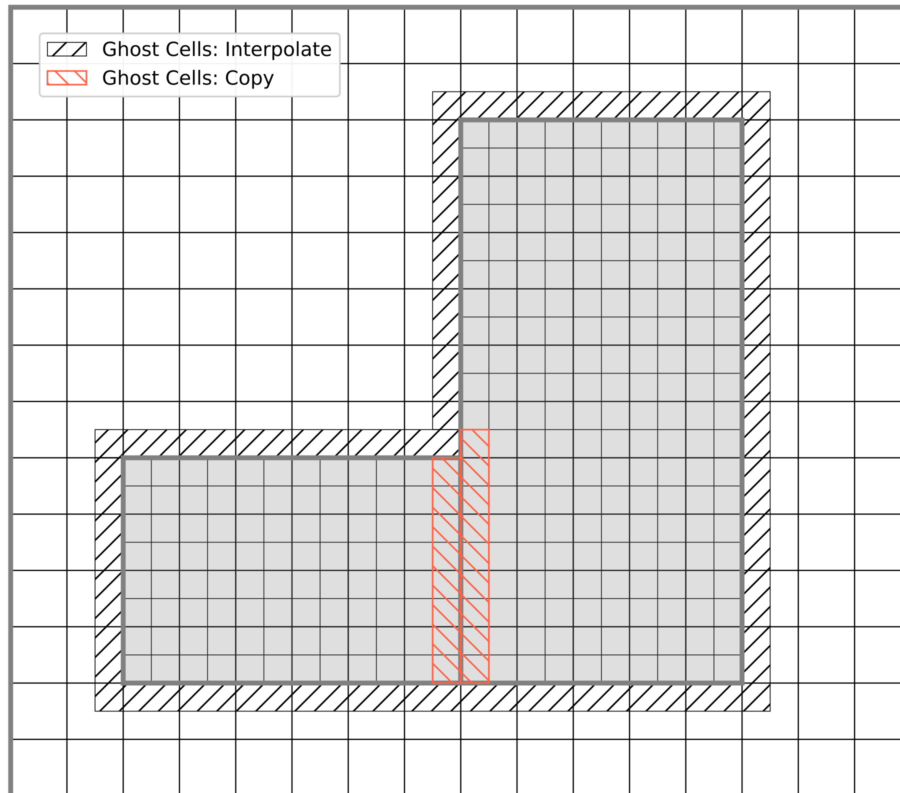


Figure 2.9: Synchronization of mesh boundary data: The ghost cell values are either copied from neighboring meshes' inner cells or interpolated from the parent mesh.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

2.2.6 Averaging

For each level $l,\ 0 \leq l < q$, meshes that lie above a finer mesh (i.e., are located at the next coarser level) have to be updated with the solution from the finer levels. This is done bottom-up by averaging the data of all finer cells contained within a coarse cell. By doing this, the numerical scheme is modified and discrete conservation is no longer guaranteed. It is possible to retain the conservation property of the numerical solver by performing a *conservation fixup* to correct the values of cells located at the border between coarse and fine meshes [1, 8].

Remark 2.7 (Average Mesh)**.**
The solution from the finer levels is propagated upwards by replacing coarse cell values with an average of the cells contained within

$$u_{ijk}^{(l)} = \frac{1}{r^3} \sum_{\alpha=0}^{r-1} \sum_{\beta=0}^{r-1} \sum_{\gamma=0}^{r-1} u_{(ir+\alpha)(jr+\beta)(kr+\gamma)}^{(l+1)}.$$

Note that at this point, the levels l and $l+1$ will always be at the same time, $t^{(l)} = t^{(l+1)}$, therefore, no interpolation in time is necessary.

The complete AMR algorithm is now given in Algorithm 2.1.

Algorithm 2.1 Adaptive Mesh Refinement

function ADVANCE(l, k)
  if $l > 0 \wedge$ time to regrid **then**
    $\mathcal{R}^{(l-1)} \leftarrow$ FLAG($\mathcal{M}^{(l-1)}$), ▷ Mark cells for refinement.
    $\mathcal{M}^{(l)} \leftarrow$ REGRID($\mathcal{M}^{(l)}, \mathcal{R}^{(l-1)}$). ▷ Adapt hierarchy.
    $u^{(l)} \leftarrow$ INITIALIZE($u^{(l)}, u^{(l-1)}$) ▷ Initialize new mesh data.
  end if
  $u^{(l)} \leftarrow$ SYNCHRONIZE($u^{(l)}$) ▷ Regenerate ghost layer values.
  $u^{(l)} \leftarrow$ INTEGRATE($u^{(l)}, k$) ▷ Timestep current level.
  if $l < q$ **then**
    for $i \in 1, \ldots, r$ **do** ▷ Advance finer grids r-times per coarse timestep.
      ADVANCE($l+1, \frac{k}{r}$) ▷ Advance finer levels recursively.
    end for
    $u^{(l)} \leftarrow$ AVERAGE($u^{(l+1)}$) ▷ Update solution with data from finer level.
  end if
end function

$t \leftarrow t_0$
while $t < t_{end}$ **do**
  $k \leftarrow \min(k^{(0)}, t_{end} - t)$
  ADVANCE(0, k)
  $t \leftarrow t + k$
end while

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3 Mesh Cell Clustering

A core part of patch-based AMR is how flagged cells are organized into refined patches. Starting from either a pre-existing mesh hierarchy or just from a set of refinement flags, the goal of clustering is to efficiently set up a patch-based nested mesh hierarchy (according to Definition 2.2) that adheres to the nesting conditions (N1)-(N5).

This task is divided into two parts:

1. In **single-level clustering** the flagged cells of one level are grouped into rectangular patches with the goal of including a minimal amount of un-flagged cells. For this step, other levels are not taken into account.
2. In **multi-level clustering**, all levels are incorporated into a nesting-criteria conforming mesh hierarchy by repeatedly applying single-level clustering. Depending on the choice of algorithm for multi-level clustering, it imposes additional constraints on the individual levels in order to satisfy the nesting conditions.

Remark 3.1.
For classical AMR (as in [1, 5, 6]), the creation of a multi-level hierarchy is rather straightforward, given a suitable algorithm for the single-level problem. In our case, the added constraint of a unique parent mesh for each child mesh (N4) as well as strict minimum mesh sizes (N5) makes this more complicated and requires special attention.

Example 1 (Demonstration Flag Hierarchy)**.**
To illustrate the clustering algorithms, a simple 4-level hierarchy (cf. Figure 3.1) is used. The base level has a resolution of 32×32 and is refined by a factor $r = 2$ at every level, yielding a grid size of 256×256 for the finest level. This representative example is adapted from [6] since it showcases the need for an additional nesting strategy; flags are located only on the third level, where 21% of the 16384 grid cells are marked and have to be propagated properly to the root level. The sharp edges in the flag structure make it visually obvious where splitting should occur.

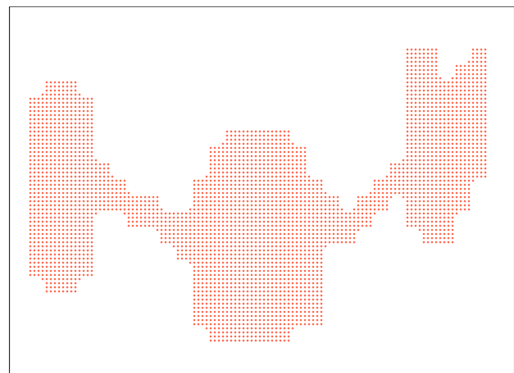


Figure 3.1: Example flags for demonstration of clustering algorithms, adapted from [6].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1 Single-Level Clustering

The goal of single-level clustering is to cover the flagged cells with conforming rectangular, axis-aligned, and non-intersecting patches as efficiently as possible. Depending on the use-case (e.g., inherent properties of the physical problem or the numerical method used), the definition of *efficient* may vary. As in [6], clustering quality is judged by the ratio of refined cells to marked cells, the mesh efficiency.

Definition 3.2 (Mesh Efficiency)**.**
For a mesh M of the patch $P(\mathbf{f}, \mathbf{s})$ and refinement function $\mathcal{R} : M \to \{0, 1\}$, the mesh efficiency is defined as the ratio of marked cells to inner mesh cells

$$\epsilon(M) := \frac{\sum_{\mathbf{x} \in M} \mathcal{R}(\mathbf{x})}{\lambda(M)}, \tag{6}$$

where $\lambda(M) := s_x s_y s_z$ is the number of cells in the mesh.
For a set of meshes $\mathcal{M}$ the pure mesh efficiency is aggregated over the individual meshes

$$\epsilon(\mathcal{M}) := \frac{\sum_{M \in \mathcal{M}} \sum_{\mathbf{x} \in M} \mathcal{R}(\mathbf{x})}{\sum_{M \in \mathcal{M}} \lambda(M)}. \tag{7}$$

Since this measure only counts the inner cells and does not take into account the added effort from interpolating, synchronizing and, if necessary, timestepping the ghost cells, the adjusted mesh efficiency is introduced.

Definition 3.3 (Adjusted Mesh Efficiency)**.**
For a given mesh M of the patch $P(\mathbf{f}, \mathbf{s})$ and refinement function $\mathcal{R} : M \to \{0, 1\}$, the mesh efficiency is defined as the ratio of marked cells to both mesh and ghost cells

$$\tilde{\epsilon}(M) := \frac{\sum_{\mathbf{x} \in M} \mathcal{R}(\mathbf{x})}{\lambda(M) + \mu(\partial M)}, \tag{8}$$

where $\lambda(M) := s_x s_y s_z$ is the number of inner mesh cells, g is the size of the ghost layer in each Cartesian direction and

$$\mu(\partial M) := (s_x + 2g)(s_y + 2g)(s_z + 2g) - s_x s_y s_z$$

is the total number of ghost cells.
For a set of meshes $\mathcal{M}$ the adjusted mesh efficiency is aggregated over the individual meshes

$$\tilde{\epsilon}(\mathcal{M}) := \frac{\sum_{M \in \mathcal{M}} \sum_{\mathbf{x} \in M} \mathcal{R}(\mathbf{x})}{\sum_{M \in \mathcal{M}} \lambda(M) + \mu(\partial M)}. \tag{9}$$

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Still, the adjusted efficiency may not accurately reflect the desired mesh quality, for example, if the computational effort required for numerical treatment of inner cells and of ghost cells differs strongly or if the computational overhead per additional mesh is large.

Therefore, a more general cost function ω is chosen in order to frame this as an optimization problem whose minimizers are mesh hierarchies with the desired efficiency properties. The task of finding an optimal clustering which covers all flagged cells is called a *rectangular covering problem.*

Definition 3.4 (Rectangular Covering Problem)**.**
The rectangular covering problem describes the task of computing an optimal rectangular covering of a finite set of points M on the grid $\mathcal{G}$, defined by a refinement function $\mathcal{R}$ as

$$M := \{\mathbf{x}_{ijk} \in \mathcal{G} \ : \ \mathcal{R}(\mathbf{x}_{ijk}) = 1\}.$$

A **rectangular covering** is a set $\mathcal{P} = (P_i)_{i=1}^n$ of n non-overlapping patches so that

$$M \subset \bigcup_{i=1}^{n} P_i,$$
$$P_i^\circ \cap P_j^\circ = \emptyset \quad \forall\, i, j: \ 1 \leq i < j \leq n.$$

An **optimal covering** is a rectangular covering $\mathcal{P}$ which minimizes the cost function

$$\begin{aligned} \omega(P) &:= \begin{cases} \lambda(P) + C_g \mu(\partial P) + C_m & \text{if} \displaystyle\sum_{ijk \in I(P)} \mathcal{R}_{ijk} > 0, \\ 0 & \text{otherwise.} \end{cases} \\ \omega(\mathcal{P}) &:= \sum_{i=1}^{n} \omega(P_i), \end{aligned} \tag{10}$$

with $C_g \geq 0$ and $C_m \geq 0$. Any patches not containing any flags are discarded during clustering and as such do not contribute any cost.

The work required for timestepping a single cell is used as unit of measurement. Then, a single ghost cell requires C_g times that effort while an additional patch is as computationally expensive as C_m additional inner cells.

There has been research into an optimal solution to this and similar problems. *Rectangle covering* problems and the related *rectangle packing* problems as well as the more general *polygon covering* have been the subject of investigation in computational geometry, but so far no optimal polynomial time algorithms exist [4, 35].

In [22], an algorithm to solve the rectangular covering problem in two dimensions was proposed. It uses a dynamic programming approach to enumerate and evaluate all relevant solutions

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and as such has exponential run-time relative to the number of flags. Since our use case requires fast clustering, this makes it unsuitable for our purpose.

A different approach is given in [21] in the context of a *best representation problem*, where, given a *signal*, a set of *representations* and a *cost function* the goal is to select a number of representations that together model the signal while minimizing the cost function. This is used, e.g., in image compression and signal analysis. The general idea is similar to that of [6]; the solution space is restricted to clusterings that can be generated by repeated splitting. A tree-like structure of all admissible tilings is generated by recursive splitting of the root domain, which is then searched for the optimal solution (relative to the restricted problem). This is much faster than a fully optimal approach but still not practicable, as it is designed for dense 2D data and its run-time would be at least $\mathcal{O}(n^2)$ (and $\mathcal{O}(n^3)$ in 3D) relative to the number of flags n in our application.

Clustering should be at most of linear complexity relative to the number of flags, as it is performed many times within a simulation. Note that as all flags need to be checked at least once, regardless of clustering algorithm, linear complexity is the lower limit. Since an optimal, linear run-time solution is unfeasible, approximate algorithms that try to minimize the cost function ω with suitable heuristics are used instead.

Initially, AMR methods used techniques adapted from artificial intelligence and computer vision meant for feature detection or pattern recognition for the generation of patch covering [1]. Nearest-neighbor clustering was used to group flagged points together, then covering rectangles are aligned with the points as best as possible. These methods allow overlapping and rotated patches, which cause a large computational overhead. They were later replaced by the Berger-Rigoutsos clustering algorithm [6].

In [23], a patch generation method based on K-Means clustering was presented. Here, a fixed number of initial centroids are used to determine a set of non-overlapping axis aligned patches. The drawback is that the number of patches must be specified ahead of time, which can be impractical.

The algorithms presented next are variations of the Berger-Rigoutsos algorithm, initially presented in [6], and Luitjens tile-based clustering as described in [25].

3.1.1 Signature-Inflection Clustering

The Berger-Rigoutsos signature-inflection algorithm (and variations thereof) is a commonly used clustering algorithm in the context of patch-based AMR as implemented in the Chombo library [36] and used in, e.g., [8, 18, 33]. It is fast, well parallelizable [25, 30] and in general produces quite efficient clusterings. Here, the Berger-Rigoutsos algorithm is modified in order to produce mesh clusterings that are more efficient (in the sense of minimizing, at least approximately, the cost function (10)).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Base Algorithm

The base algorithm is given as pseudo-code in Algorithm 3.1. It is based on flag histograms, called signature arrays, defined for a patch P as

$$
\begin{aligned}
S_x(m) &:= \sum_{j \in I_y(P)} \sum_{k \in I_y(P)} \mathcal{R}_{mjk}, \\
S_y(m) &:= \sum_{i \in I_x(P)} \sum_{k \in I_z(P)} \mathcal{R}_{imk}, \\
S_z(m) &:= \sum_{i \in I_x(P)} \sum_{j \in I_y(P)} \mathcal{R}_{ijm}.
\end{aligned}
$$

Starting from a base patch, the patch is split up at index m along an axis a into two patches (cf. Algorithm 3.2). If there is a hole in a signature array, e.g., $S_a(m) = 0,\ a \in \{x, y, z\},\ m \in I_a(P)$, the patch is split at that point. A hole in the signature array implies two disconnected areas of marked cells; since there are no marked cells at that index, at least one of the resulting patches can be shrunk along the axis a, significantly reducing mesh cell count.

In case there is no hole, the index m_s and axis a_s are determined as the maximum magnitude zero crossing (*inflection point*) of the second (discrete) derivative $\Delta S_a(m)$ of the signature arrays

$$
\begin{aligned}
\Delta S_a(m) &:= S_a(m+1) - 2S_a(m) + S_a(m-1) \\
F(m,a) &:= \begin{cases} \Delta S_a(m) - \Delta S_a(m-1) & \text{if } \operatorname{sgn}(\Delta S_a(m)) \neq \operatorname{sgn}(\Delta S_a(m-1)), \\ 0 & \text{otherwise.} \end{cases}
\end{aligned}
$$

$$
(m_s, a_s) := \underset{(m,a)}{\arg\max} \left\{ |F(m,a)| \right\}.
$$

The choice of the index is inspired by edge-detection: the resulting splitting index identifies an *edge* when viewing the flags as a binary image. It divides the patch where the largest change from flagged cells to non-flagged cells occurs.

Figure 3.2 shows a flagged grid with signature arrays, Laplacian and magnitudes, resulting in a split along the X-axis. This does not necessarily directly decrease cell count, but if applied iteratively will partition the grid into rectangular patches containing only the flagged cells (cf. Figure 3.5). If there is no hole and no inflection point, then the patch is bisected (split in half along the longest axis). As it is unnecessary to remove *all* unflagged cells, the splitting is stopped if the target efficiency ϵ_0 is reached or if no split is possible without violating the minimum size requirement (N5).

Afterwards, patches are merged together. This is not meant to be an optimal global merging strategy: Neighboring patches are simple merged together where possible. Thereby superfluous splits that did not result in a reduction of refined cells are removed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Remark 3.5.
The algorithm splits an existing patch into two parts. This means that the resulting patches never intersect, satisfying (N2). The determination of the split index is done with respect to coordinates of the coarser grid, so that the resulting splits are always aligned to the parent grid (N3).

Algorithm 3.1 Signature Clustering

```
function SignatureCluster(P_0, ε_0, m_0)
    Q ← {P_0}                                          ▷ Start with root patch.
    for P ∈ Q do                                       ▷ Iteratively process all patches.
        P ← BoundingBox(P)                             ▷ Bounding box around flags, at least size m_0.
        if ε(P) < ε_0 ∧ ∃a ∈ {x, y, z} : P_{s_a} ≥ 2m_0 then   ▷ ε_0 not reached, P large enough.
            (P_1, P_2) ← Split(P, m_0)                 ▷ Split patch into 2 parts.
            Q ← Q \ {P}                                ▷ Remove from queue.
            Q ← Q ∪ {P_1, P_2}                         ▷ Add new patches to queue.
        end if
    end for
    Q̃ ← Merge(Q)                                       ▷ Merge patches where possible.
    return Q̃                                           ▷ Return final set of patches.
end function
```

Algorithm 3.2 Signature Split

```
function Split(P, m_0)
    S_x(m) := Σ_{j∈I_y(P)} Σ_{k∈I_z(P)} R_{mjk}, 0 ≤ m ≤ n_x}     ▷ Build signature for X-axis.
    S_y(m) := Σ_{i∈I_x(P)} Σ_{k∈I_z(P)} R_{imk}, 0 ≤ m ≤ n_y}     ▷ Build signature for Y-axis.
    S_z(m) := Σ_{i∈I_x(P)} Σ_{j∈I_y(P)} R_{ijm}, 0 ≤ m ≤ n_z}     ▷ Build signature for Z-axis.
    I ← {m_0, ..., n_a − m_0} × {x, y, z}
    if ∃(i, a) ∈ I : S_a(i) = 0 then
        (i*, a*) ← (i, a)                                         ▷ Split at hole.
    else if max_{(i,a)∈I} {|F(i, a)|} > 0 then
        (i*, a*) ← argmax_{(i,a)∈I} {|F(i, a)|}.                  ▷ Split at inflection point.
    else
        (i*, a*) ← (i, a) | m_0 ≤ i ≤ n_a − m_0                   ▷ Split at midpoint.
    end if
    (P_1, P_2) ← split P at (i*, a*)
    return (P_1, P_2)
end function
```


TU Bibliothek Your knowledge hub WIEN
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

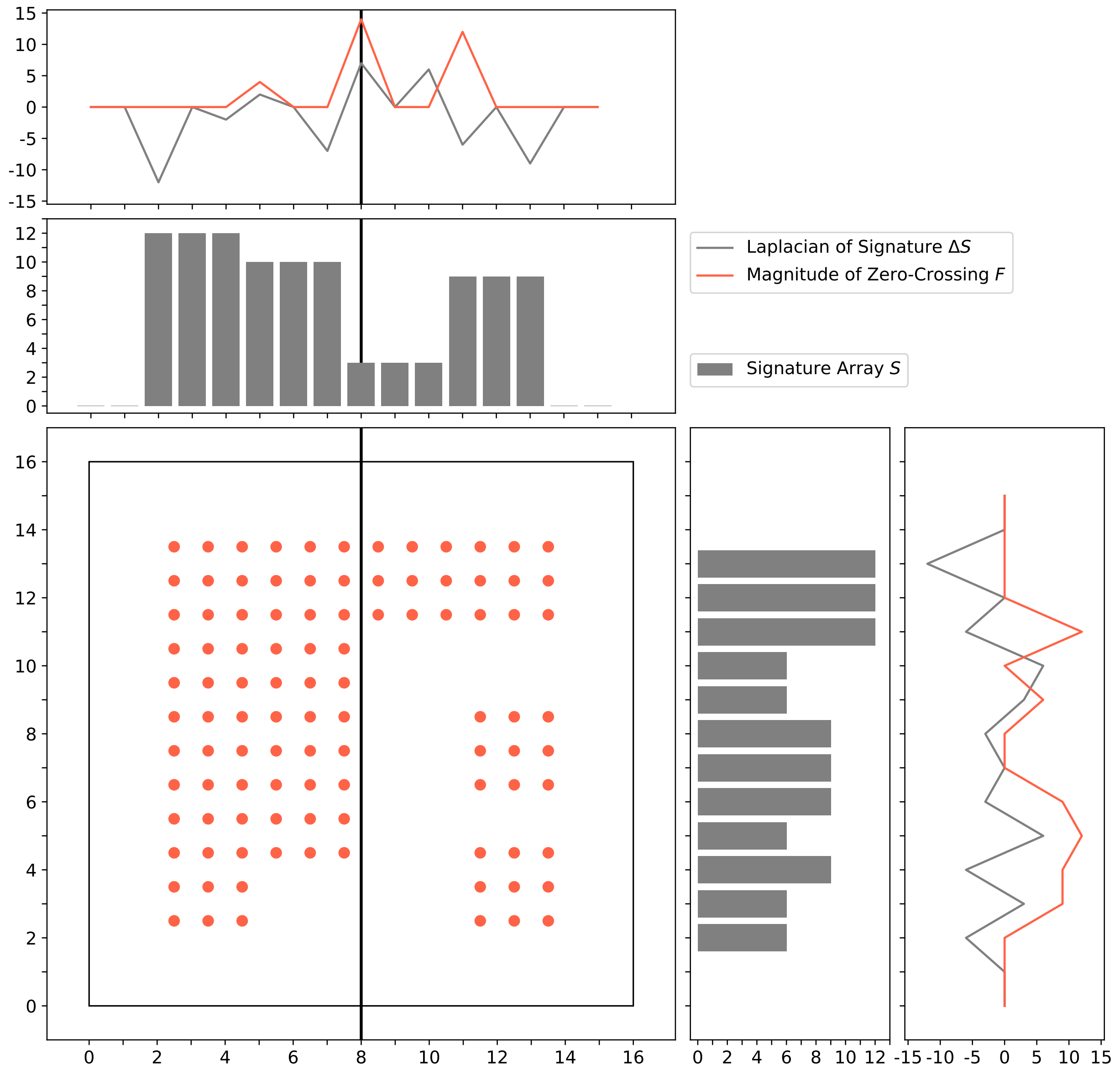


Figure 3.2: Example grid with 102 flags and the corresponding signature arrays and inflection magnitudes. As there are no holes, the patch is split at index 8 along the X-axis at the maximum of the zero-crossing magnitudes of both axes, $F(8, x) = 14$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Aspect Ratio Correction

The Berger-Rigoutsos algorithm as described above seems to favor longer, more asymmetrical patches, as is shown in Figure 3.3, which results in a higher number of ghost cells. Sometimes this leads to inclusion of unnecessary cells and as such less efficient clusterings. To mitigate this, the way the split index is chosen is adjusted by penalizing splits that result in large amounts of additional ghost cells; in essence, this prefers splits resulting in more regular (i.e., similar-sized in all dimensions) patches. To do this, the split magnitude F of a patch is adapted to take into account the mean *aspect ratio* of the resulting split patches, that is, the ratio of the shortest patch side to the respective longest patch side.

For a patch $P(\mathbf{f}, \mathbf{s})$ and a split at index m along the x-axis, the correction factor σ is given by

$$\sigma(m) := \left(\frac{\sigma^- + \sigma^+}{2}\right)^\alpha, \quad \alpha \in \mathbb{R}_0^+,$$

where σ^+ and σ^- are the aspect ratios of the two patches resulting from a split at index m

$$\sigma^+(m) := \frac{\min\{m, R_y^{\max^+} - R_y^{\min^+} + 1, R_z^{\max^+} - R_z^{\min^+} + 1\}}{\max\{m, R_y^{\max^+} - R_y^{\min^+} + 1, R_z^{\max^+} - R_z^{\min^+} + 1\}},$$
$$\sigma^-(m) := \frac{\min\{s_x - m, R_y^{\max^-} - R_y^{\min^-} - 1, R_z^{\max^-} - R_z^{\min^-} - 1\}}{\max\{s_x - m, R_y^{\max^-} - R_y^{\min^-} - 1, R_z^{\max^-} - R_z^{\min^-} - 1\}}.$$

For the X-axis, the patch lengths are simply m and $s_x - m$. The remaining side lengths are calculated from the minimum and maximum flag indices $R_a^{\min^\pm}$ and $R_a^{\max^\pm}$ with respect to the other axes, $a \in \{y, z\}$. For this, the set of flag indices for each patch is defined as

$$R^- := \{(i,j,k) \in I(P) \ : \ \mathcal{R}_{ijk} > 0, \ i < f_x + m\},$$
$$R^+ := \{(i,j,k) \in I(P) \ : \ \mathcal{R}_{ijk} > 0, \ i \geq f_x + m\}.$$

The minimum and maximum indices for each patch and direction are then determined by

$$R_y^{\max^-}(m) := \max\{j \ : \ (i,j,k) \in R^-\}, \quad R_y^{\max^+}(m) := \max\{j \ : \ (i,j,k) \in R^+\},$$
$$R_z^{\max^-}(m) := \max\{k \ : \ (i,j,k) \in R^-\}, \quad R_z^{\max^+}(m) := \max\{k \ : \ (i,j,k) \in R^+\},$$
$$R_y^{\min^-}(m) := \min\{j \ : \ (i,j,k) \in R^-\}, \quad R_y^{\min^+}(m) := \min\{j \ : \ (i,j,k) \in R^+\},$$
$$R_z^{\min^-}(m) := \min\{k \ : \ (i,j,k) \in R^-\}, \quad R_z^{\min^+}(m) := \min\{k \ : \ (i,j,k) \in R^+\}.$$

Together, this yields an adjusted split magnitude of

$$\tilde{F}(m,x) := \begin{cases} \sigma(m)\left(\Delta S_x(m+1) - \Delta S_x(m)\right) & \text{if } \operatorname{sgn}(\Delta S_x(m+1)) \neq \operatorname{sgn}(\Delta S_x(m)), \\ 0 & \text{otherwise.} \end{cases}$$

The split magnitudes for the other axes, $\tilde{F}(m,y)$ and $\tilde{F}(m,z)$, are defined analogously.
The strength of this adjustment is controlled with the additional parameter α, where $\alpha = 0$ results in the same split as the Berger-Rigoutsos algorithm.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

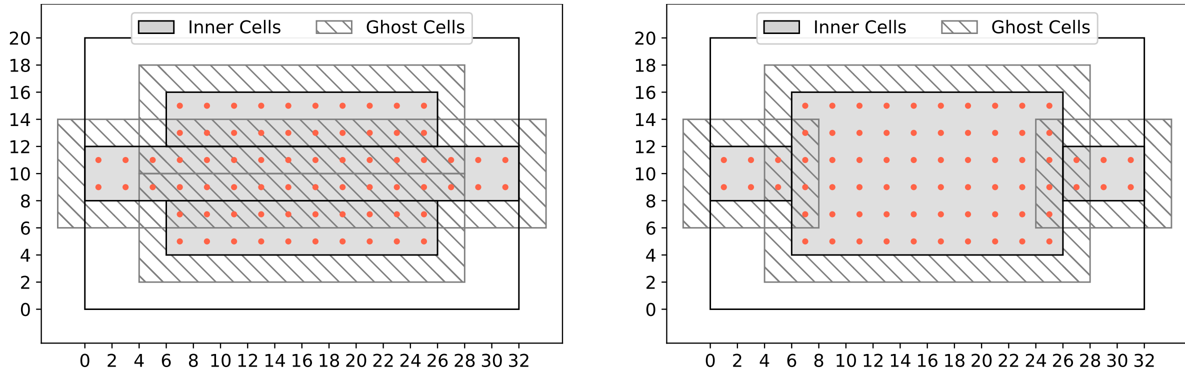


Figure 3.3: Example grid with default (left) and modified (right) splits. Either way results in 3 patches, but the number of ghost cells is reduced by the modified split.

Backtracking

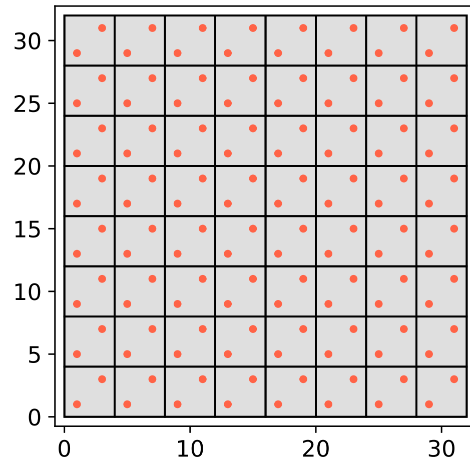


Figure 3.4: Chessboard grid with minimum width $m_0 = 4$ and refinement factor $r = 2$ where cells are flagged alternatingly. Patches are split unnecessarily if $\epsilon_0 > 0.5$.

For certain inputs, the Berger-Rigoutsos algorithm will continue splitting patches even if the resulting clustering is worse. This can be seen in the pathological example of a chessboard-like flag structure, as in Figure 3.4. Target efficiency ϵ_0 may be used to control the algorithm. However, which target efficiency corresponds to the most effective clustering is heavily input-dependent. For the chessboard problem, any target efficiency $\epsilon_0 > 0.5$ will cause the algorithm to split until the minimum size is reached, generating a large number of patches covering the same area as the original patch.

To avoid over-splitting during clustering, a *backtracking* component is added to the algorithm: Every patch P is linked with its resulting splits $P_1(P)$ and $P_2(P)$, allowing us to compute each splits *score* (value of the cost function ω) and propagate it upwards, then selecting the last split that results in an improved clustering.

This procedure can be interpreted as generating a binary tree followed by a depth-first search for the best cut-off point in each branch. It is necessary to fully traverse the tree until all splits have been visited before backtracking. An extension of this algorithm could work similarly to [21] and not only explore one split at each branching point but multiple, or even all. This was not pursued here as the additional effort scales exponentially in the number of evaluated splits which would make the algorithm too costly for use within a typical AMR simulation.

Figure 3.5 shows the evolution of the resulting patches if splitting the grid from Figure 3.2 according to the above algorithm. First, patches are split along holes, then trimmed to a tight fit. Then they are split again with split index and axis (m, a) determined as described.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.3 Backtracking

function BACKTRACK(Q)
  for $P \in Q$ **do**
    $\hat{P} \leftarrow$ PARENT(P)
    if $\omega(\hat{P}) \leq \omega(P_1(\hat{P})) + \omega(P_2(\hat{P}))$ **then** ▷ Check if splitting improves score.
      $Q \leftarrow (Q \setminus \{P_1(P), P_2(P)\}) \cup \{P\}$ ▷ Continue with 'un-split' patch.
    end if
  end for
  return Q
end function

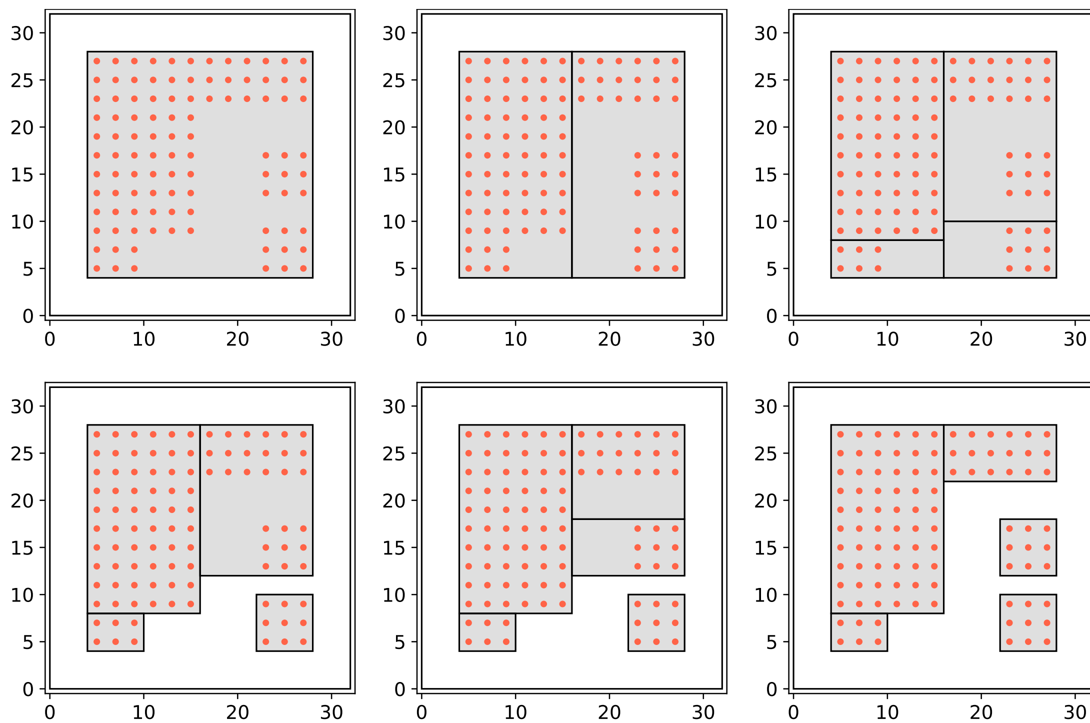


Figure 3.5: Evolution of the clustering from Figure 3.2. The grid is repeatedly split according to Algorithm 3.1 with minimum width $m_0 = 2$ and aspect ratio correction $\alpha = 2$, resulting in an efficient covering that doesn't include any unmarked cells.

Remark 3.6 (Signature Clustering Algorithm Complexity)**.**
The algorithm processes each flag multiple times, depending on how the patches are split. Computing signature arrays and magnitudes globally has complexity $\mathcal{O}(|\mathcal{R}|)$. This is then repeated recursively on each split, creating a binary tree where in the best case, each node except the lowest level, has two children. The depth of this tree is between $\mathcal{O}(|\mathcal{P}|)$ for an unbalanced tree and $\mathcal{O}(\log|\mathcal{P}|)$ for a balanced tree. For most inputs, the complexity is similar to the best case, yielding a total complexity of $\mathcal{O}(|\mathcal{R}| \log|\mathcal{P}|)$ (cf. [25]).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.1.2 Tile Clustering

Presented in [25] as a highly scalable alternative to the signature clustering algorithm, Luitjens tile clustering is also a much simpler approach. The flags are sorted into tiles (fixed-size patches) and all empty tiles are removed (Algorithm 3.4). In cases where the grid size is not divisible by the tile size, the first and last tiles in each spatial direction may have smaller side lengths.

This results in a very fast and simple clustering algorithm, but it's mesh efficiency depends highly on the choice of tile size $\mathbf{d} \in \mathbb{N}^3$.

Formally, the tile clustering is simply the subset $\mathcal{P} \subset \mathcal{T}$ of all tiles that contain at least one flag

$$\mathcal{T} := \{P(\mathbf{f}, \mathbf{s}) \ : \ \mathbf{f} \bmod \mathbf{d} = \mathbf{0},\ \mathbf{s} = \mathbf{d}\}$$
$$\mathcal{P} := \{P \in \mathcal{T} \ : \ \exists (i,j,k) \in I(P) \ : \ \mathcal{R}_{ijk} = 1\}.$$

The resulting tiles are shrunk to the bounding box of contained flags in order to reduce the cell count. Depending on the application it may be beneficial to also merge the resulting tiles to reduce mesh count.

Algorithm 3.4 Tile Clustering

```
function CLUSTER(P, d)
    Q ← {}
    for (i, j, k) ∈ I : R_ijk = 1 do
        Q ← Q ∪ {P ((i − i mod d_x, j − j mod d_y, k − k mod d_z), d)}
    end for
    for P ∈ Q do
        P ← BOUNDINGBOX(P)
    end for
    return Q
end function
```

Tile sizes need not be identical in all directions, as long as the above condition is fulfilled along each axis. However, the choice of square or cubic tiles $d := d_x = d_y = d_z$ is usually optimal as it minimizes the number of ghost layer cells in relation to inner cells.

Figure 3.6 shows the tile clustering of a circularly flagged example grid with size 64×64 and tile sizes $d = 4$ and $d = 8$. As can be seen the choice of tile size represents a trade-off between cell count and mesh count, where smaller tiles include less superfluous cells but result in a higher mesh count.
Larger tile sizes d will result in less tiles with less efficiency. Small tile sizes result in more tiles with higher efficiency, but the overhead of having more patches might negate this, depending on the implementation of the remaining AMR algorithm and the numerical integration algorithm used for timestepping as well as the characteristics of the computing architecture with respect to parallelization.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

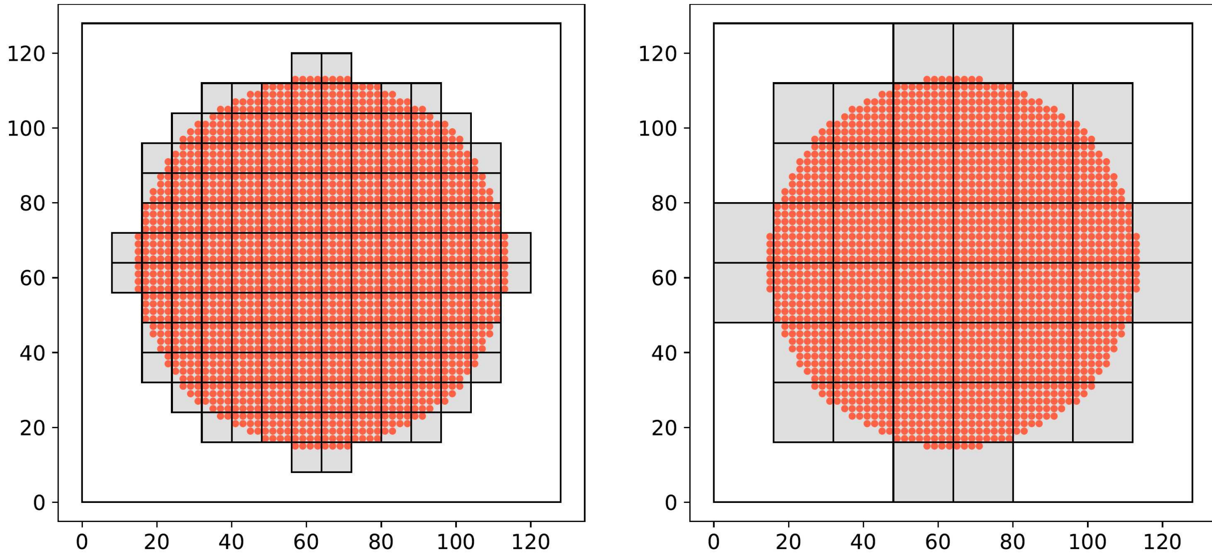


Figure 3.6: Tile clustering of circular area of interest with tile sizes $d = 4$ and $d = 8$. Smaller tile size results in a smaller number of unmarked cells being included but an increase in ghost cell count owing to the increased number of patches.

Tile clustering was also discussed and improved upon in [30], where an additional merge step is used to reduce the number of tiles. This is not done here, since the regularity of standard tile clustering turns out to be very useful when combining multiple levels into a nested grid hierarchy.

Other ways of forcing more regularly shaped patches were explored in [33], e.g., by bounding the minimum and maximum edge size or maximum cell count. Goals in these cases are usually not only regularity in terms of aspect ratio, but also uniformity of mesh size (resulting in easier parallelization), which is not necessarily needed here since parallelization is considered only on shared-memory architectures where scheduling and distribution is easier.

Remark 3.7.
Tile clustering evidently creates a set of unique, non-intersecting patches, satisfying (N2). Tile size is given with respect to the coarser grid coordinates, ensuring that tiles are always aligned correctly (N3).

Remark 3.8 (Tile Clustering Algorithm Complexity)**.**
Tile clustering only needs to consider each flag once to create the initial patch set. The number of resulting patches is always smaller than or equal to the number of flags. As such, its complexity is simply $\mathcal{O}(|\mathcal{R}|)$. The effective run-time and memory usage can be further reduced by using a data structure adapted to the tile-structure - storing only one flag per tile.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2 Multi-Level Clustering

Since AMR works on more than one level, the above algorithms need to be integrated into a hierarchical framework to generate a multi-level clustering of all levels. The challenge is creating an efficient mesh hierarchy while conforming to the nesting criteria from Definition 2.3.

Traditionally, an AMR mesh hierarchy does not need to conform to (N4) (i.e., meshes need not always have unique parents). Because of this additional constraint, algorithms like those used in [1, 3, 18] are insufficient for our use-case. Two approaches for multi-level clustering are investigated: a signature-based top-down approach, which starts clustering the top level using signature clustering and then recursively descends the hierarchy downwards and, alternatively, a tiling-based bottom-up approach that starts at the bottom and works its way upwards.

Both methods are followed by a post-processing step traversing the hierarchy in the corresponding opposite direction to ensure nesting and further improve efficiency. The benefits and difficulties of the two methods differ as the nesting criteria have to be enforced in different ways.

As in single-level clustering, the quality of the resulting mesh hierarchy is evaluated based on its efficiency and score, which are aggregated from the individual levels.

Definition 3.9 (Mesh Hierarchy Efficiency and Score.)**.**
For a given mesh hierarchy $\mathcal{L} = \left(\mathcal{L}^{(l)}\right)_{l=0}^{q}$, the mesh cell counts for each hierarchy level

$$\mathcal{L}^{(m)} = \left(\mathcal{G}^{(l)}, \mathcal{M}^{(l)}, \mathcal{P}^{(l)}, \mathcal{R}^{(l)}\right), \quad 0 < l \leq q,$$

are combined into a pure hierarchy efficiency of

$$\epsilon(\mathcal{L}) := \frac{\sum\limits_{1 \leq l \leq q} \sum\limits_{M \in \mathcal{M}^{(l)}} \sum\limits_{\mathbf{x} \in M} \mathcal{R}^{(l-1)}(\mathbf{x})}{\sum\limits_{1 \leq l \leq q} \sum\limits_{M \in \mathcal{M}^{(l)}} \lambda(M)}, \tag{11}$$

and an adjusted hierarchy efficiency of

$$\tilde{\epsilon}(\mathcal{L}) := \frac{\sum\limits_{1 \leq l \leq q} \sum\limits_{M \in \mathcal{M}^{(l)}} \sum\limits_{\mathbf{x} \in M} \mathcal{R}^{(l-1)}(\mathbf{x})}{\sum\limits_{1 \leq l \leq q} \sum\limits_{M \in \mathcal{M}^{(l)}} \lambda(M) + \mu(\partial M)}. \tag{12}$$

The root mesh is not taken into account here, as its size is fixed.

As in Definition 3.4, The quality of mesh hierarchies is ranked based on their value of the cost function ω, which is the sum of the cost of the individual level's clusterings

$$\omega(\mathcal{L}) := \sum_{0 \leq l \leq q} \omega(\mathcal{P}^{(l)}). \tag{13}$$

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

3.2.1 Top-Down Clustering

Starting with the coarsest level, an initial patch set is created by clustering it using a single-level signature-inflection clusterer. This results in a set of non-overlapping ((N2) and (N3) conforming) patches covering all flagged points at the coarsest level. Those patches are then recursively clustered again, implicitly fulfilling (N4). The challenge is then the last criterion: neighboring mesh cells must only differ by at most one level (N1).

To ensure (N1), the flags first need to be propagated from the lowest level upwards to determine which cells need to be refined. Starting from the second-finest level, all cells that are within one cell of a flag at the next lower level will also be flagged (Algorithm 3.5). The addition of further nesting-induced buffer cells may be required.

Algorithm 3.5 Propagate Flags

function PROPAGATE($(\mathcal{R}^{(l)})_{l=0}^{q}, k$)
  for $l \in \{q-2, \dots, 0\}$ **do**
    $\mathcal{R}^{(l)} \leftarrow \mathcal{R}^{(l)} \cup \{x \in \mathbb{N}_0^3 : d(x, \mathcal{R}^{(l+1)})_\infty \leq k\}$ ▷ Add $k \geq 1$ cells around lower level flags.
  end for
end function

First, let the minimum patch size be $m_0 = r$. Then Algorithm 3.6 is used to ensure proper nesting (N1) of the new child patches:

1. Start with a patch to be refined. Shrink patch to the bounding box of all contained flags.
2. Check if that patch has any flags at the direct borders that have uncovered neighbors. If not, proceed normally.
3. If there is a border and a cell which has an uncovered neighbor, split the patch at $m_0/r = 1$ distance from the corresponding edge.
4. Repeat for both patches, checking all outer borders.

For $m_0 > 1$, it is not always possible to split the edge-patches as required without violating the minimum-size constraint; an additional buffer of size $k = \max\left(\frac{m_0}{r^2}, 1\right)$ has to be set around

Algorithm 3.6 Split Edges

function SPLITEDGES($P_0, \mathcal{P}_N, m_0$)
  $Q \leftarrow \{\text{BOUNDINGBOX}(P_0)\}$
  for $P \in Q$ **do**
    if $\exists\, \mathbf{x} \in P, \tilde{P} \in \mathcal{P}_N \;:\; d(\mathbf{x}, \partial P_0)_\infty < 1 \wedge d(\mathbf{x}, \tilde{P})_\infty > 1$ **then**
      $\{P_1, P_2\} \leftarrow$ split P at m_0/r distance from corresponding edge.
      $Q \leftarrow (Q \setminus \{P\}) \cup \{P_1, P_2\}$
    end if
  end for
  return Q
end function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Algorithm 3.7 Top-Down Clustering

```
function TopDownCluster((𝒫^(l))_{l=0}^q, (ℛ^(l))_{l=0}^q, ϵ0, m0)
    Propagate((ℛ^(l))_{l=0}^q, m0/r^2)          ▷ Propagate flags with nesting-induced buffer.
    𝒫^(1) ← SignatureCluster(P^(0), ϵ0, m0)              ▷ Cluster the root patch.
    for l ∈ {1, ..., q − 1} do
        SortFlags(ℛ^(l), 𝒫^(l))                          ▷ Sort flags into existing patches.
        𝒫_N ← Neighbors(P, 𝒫^(l))                        ▷ Get patch neighbors.
        for P ∈ 𝒫^(l) do
            Q ← SplitEdges(P, 𝒫_N, m0)                   ▷ Split edge patches as described above.
            𝒫^(l+1) ← ⋃_{P̃∈Q} SignatureCluster(P̃, ϵ0, m0) ∪ 𝒫^(l+1)   ▷ Cluster each patch.
        end for
    end for
    for l ∈ {q − 1, ..., 1} do                           ▷ Shrink patches if possible.
        𝒫^(l) ← Shrink(𝒫^(l))
    end for
    for l ∈ {1, ..., q} do                               ▷ Merge suitable patches together.
        𝒫^(l) ← Merge(𝒫^(l))
    end for
end function
```

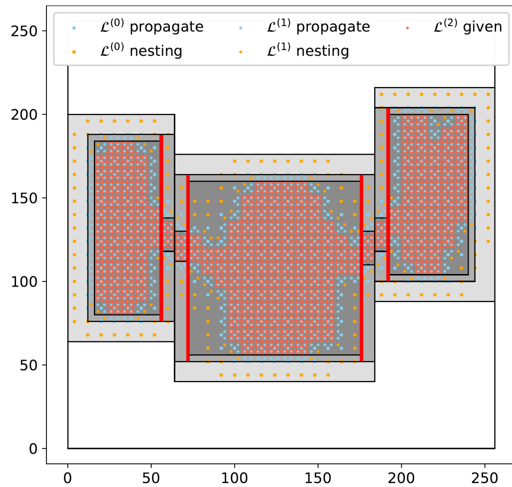
flags at the lower level. Then, as previously, patches are split at m_0/r distance from boundaries if required. Because of the additional buffering, this is indeed always possible.

This process is then repeated at the next lower level using the created patches as input (cf. Algorithm 3.7). Then the resulting patches patches are trimmed: From the bottom up, all patches are shrunk as much as possible while still covering all required flags. Since the flags on finer levels are already propagated to the coarser levels, covering all flags ensures the child patches are covered as well. For this step, nesting-induced flags are ignored and most of the 'unnecessarily' included cells are trimmed off. A final top-down pass checks for possible patches that can be merged together to reduce the number of meshes and ghost cells.

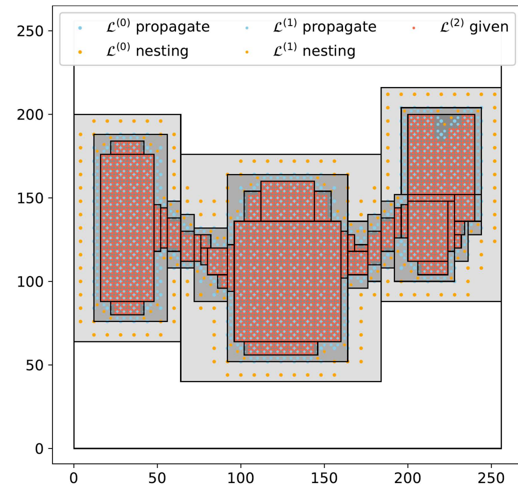
Figure 3.7 shows the 3 steps of the top-down clustering algorithm: First, for each externally *given* flag (e.g., from error estimation), a *propagate* flag is added at all neighboring cells of the next coarser level. Additionally, $m_0/r^2 = 1$ *nesting* flag is added in each direction. Then, the 3 existing level 1 patches are shrunk and split according to Algorithm 3.6 to guarantee proper nesting (marked red in Figure 3.7a). The resulting patches are clustered, the process is repeated for the finer levels (cf. Figure 3.7b) and lastly, all patches are shrunk as much as possible, cutting away superfluous cells only needed for nesting (cf. Figure 3.7c). Without Algorithm 3.6 to ensure proper nesting, the resulting clustering does not fulfill the nesting criterion (N1), as a number of patches have a direct border with an under-refined area (marked red in Figure 3.7d).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

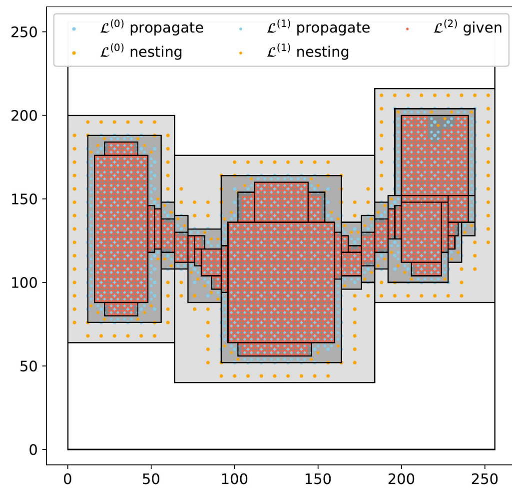
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



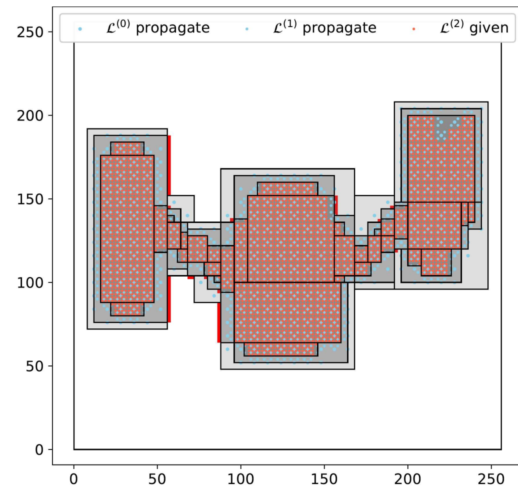
(a) Nesting induced splits only - invalid hierarchy.



(b) After clustering - valid hierarchy.



(c) After shrinking - final, valid hierarchy.



(d) Without edge-splits - invalid hierarchy!

Figure 3.7: Nested mesh hierarchy for Example 3.1 with and without proper nesting. Flags are colored and labeled by level $\mathcal{L}^{(l)}$ and flag type: *Given* flags are externally imposed refinement markers, while *propagate* and *nesting* flags are added by the algorithm itself.

Remark 3.10 (Partial Regrid)**.**
During timestepping of the AMR algorithm, a regrid is not done for the full hierarchy at every timestep, but rather from a level l downwards. This requires that all flags (including those added by the nesting algorithm) are contained within the patches of level $\mathcal{L}^{(l)}$. If that is not the case, level l also needs to change. This occurs if the area of interest moves too fast and suggests that the buffer region is not large enough. The AMR algorithm deals with this by regridding as many levels as required but this does imply a too small buffer size or suboptimal choice of refinement flags and causes a significant increase in computational effort.

3.2.2 Bottom-Up Clustering

An alternative approach is to start at the lowest level and work upwards. This means (N1) is fulfilled implicitly, because patches can be added where required. The disjunct patch and alignment criteria (N2) and (N3) are given as long as the clustering algorithm yields correct patches, which is the case for the algorithms presented above. The algorithm used in the literature [8, 18, 30, 33] uses this approach together with signature clustering to create the patch hierarchy. This is not possible in our case, since the resulting patches do not have unique parents, violating (N4).

Instead, a tiling clusterer is used as detailed in Algorithm 3.4. Choosing the tile size as multiple of the refinement factor ensures that tiles always align with the parent grid, as long as tile sizes decrease appropriately with increased refinement. Of course, tiles must always be larger then the minimum width. Thus, for each level $\mathcal{L}^{(l)}$ tile size must fulfill

$$d^{(l)} = k^{(l)} r \geq m_0,\ k \in \mathbb{N}, \qquad d^{(l+1)} \leq d^{(l)} r.$$

At every level, all tiles containing flags are then identified and added to the patch hierarchy. The choice of tile size guarantees that every patch is fully covered by a parent tile, with each parent containing up to $({}^{rd^{(l)}}/{}_{d^{(l+1)}})^3$ child tiles. Then, additional tiles are added next to child tiles wherever a tile borders an area that is not refined enough to satisfy (N1).

Up to this point each level of the patch hierarchy contains fully uniform patches. The total cell count is further reduced by shrinking tiles where possible. This is also done bottom-up to ensure the shrunk patches still adequately cover all flags as well as child patches. Then, the number of meshes may be decreased by merging suitable patches together. This is a fast process, since only neighboring tiles need to be checked. However, the patches resulting from this naive merging procedure are no longer uniform and in the resulting clustering is in most cases still less efficient than signature clustering.

Figure 3.8 shows the resulting nested mesh hierarchy when clustering Example 1 with and without subsequent shrinking and merging. As expected, all patches are properly nested as additional tiles are added where needed, even if no flags are present within.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

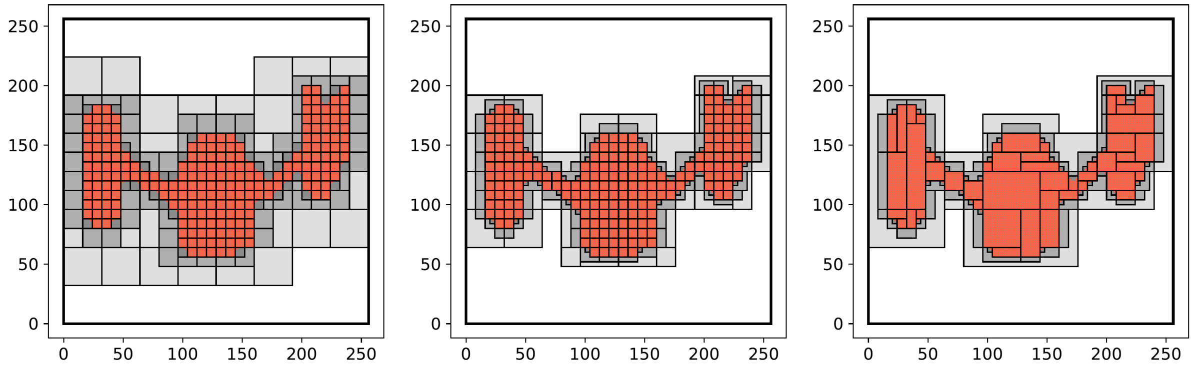


Figure 3.8: Example 1 with patch hierarchy generated by bottom-up clustering. The left graph shows the resulting patch hierarchy without shrinking or merging. Tiles not containing any flags are added where necessary to ensure no patch borders an un-refined area. The middle graph shows the patch hierarchy after shrinking patches where possible. This not only shrinks but completely removes many tiles if they are no longer required for proper nesting. Successive merging (on the right) reduces the number of meshes, but as the merging process is not optimal it results in mostly irregular, elongated patches.

Algorithm 3.8 Bottom-Up Clustering

function BottomUpCluster($(\mathcal{P}^{(l)})_{l=0}^{q}, (\mathcal{R}^{(l)})_{l=0}^{q}, \mathbf{d}$)
  for $l \in \{q-1, \dots, 0\}$ **do**
    $\mathcal{P}^{(l+1)} \leftarrow$ TileCluster($\mathcal{R}^{(l)}, \mathbf{d}$) ▷ Add tiles that contain flags.
    if $l+1 < q$ **then**
      $\mathcal{P}^{(l+1)} \leftarrow$ Nest($\mathcal{P}^{(l+2)}$) ▷ Add tiles that border child tiles.
    end if
  end for
  for $l \in \{q, \dots, 1\}$ **do** ▷ Optional shrink step.
    $\mathcal{P}^{(l)} \leftarrow$ Shrink($\mathcal{P}^{(l)}$)
  end for
  for $l \in \{1, \dots, q\}$ **do** ▷ Optional merge step.
    $\mathcal{P}^{(l)} \leftarrow$ Merge($\mathcal{P}^{(l)}$)
  end for
end function

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

4 Implementation

4.1 Requirements

The AMR algorithm, including timestepping, synchronization, interpolation and averaging as described in Section 2, as well as the mesh generation algorithms presented in Section 3 were implemented as a stand-alone C++11 library called `Nest`. The goals for this implementation were the development of a clustering algorithm that generates nested mesh hierarchies conforming to our nesting criteria (N1)-(N5) that is efficient, parallelizable and can be integrated with existing code with minimal additional effort.

The AMR implementation presented here consists of the following parts:

- AMR Shell
  - Mesh timestepping: Coordinate time advances of the mesh hierarchy.
  - Mesh synchronization: Exchange ghost layer data at each timestep.
  - Mesh interpolation: Interpolate newly created meshes from parent mesh.
  - Mesh averaging: Propagate more accurate data from finer meshes upwards.
- Regridding: Create new mesh hierarchy from existing meshes and given flags.
  - Flag propagation: Ensure flags are covered at coarser levels to guarantee nesting.
  - Creation of patch hierarchy from flags: Cluster flags to create new patch hierarchy, keep track of patch parents and children, compute patch neighbors.
  - Adaption of existing mesh hierarchy with new flags: Create new meshes from the new patch hierarchy and initialize from existing meshes or interpolate.
- Upwind Integrator: First order scheme, for demonstration purposes.

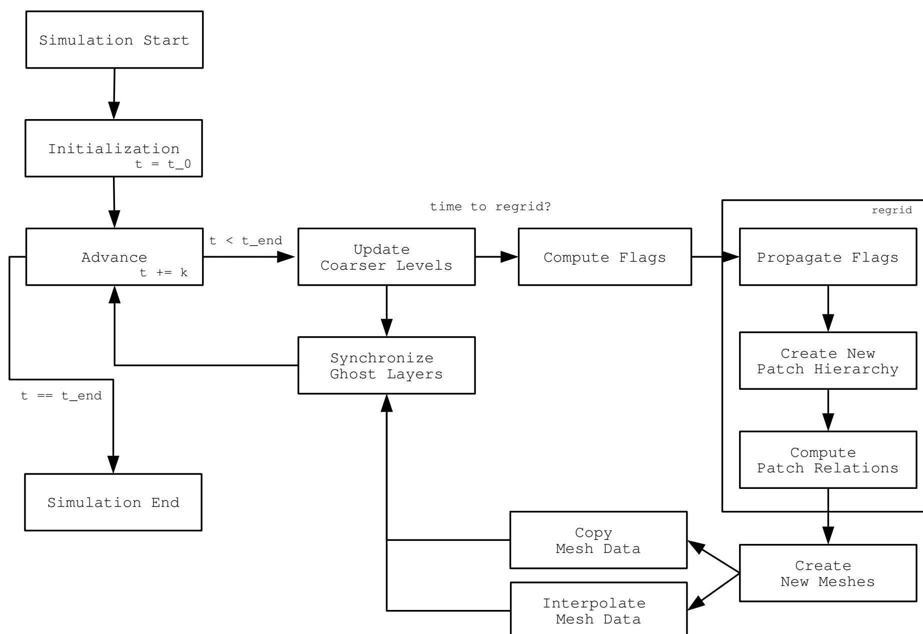


Figure 4.1: Flowchart illustrating the main tasks of the AMR procedure: timestepping, flagging, regridding, initialization and synchronization.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

Figure 4.1 shows an overview of the AMR process as implemented. The box labeled `regrid` contains the steps of the regridding process, which is encapsulated to allow independent use of only the mesh generation code without a full AMR simulation.

The implementation provides improved mesh hierarchy generation with the aim to ease integration with already available AMR software and numerical solvers. This required a strong de-coupling of the nested mesh generation from the remainder of the AMR algorithm. Both parts were developed specifically for this thesis and do not depend on third-party AMR code. In particular, the mesh generation takes as input only a level and all existing patches at that level (which are to be left unchanged) as well as all flags at that level and all levels below, without direct access to mesh data.

The output should then be a new, maximally efficient, (N1)-(N5) conforming patch hierarchy. The patches at and above the passed level should be left in place but are allowed to change if required to ensure proper nesting.

4.2 Interface

The main components of the AMR library are `Nest::Hierarchy`, `Nest::Grid`, `Nest::Mesh` and `Nest::Patch`. These classes are modeled after the definitions for Cartesian grid / mesh and patch hierarchy (Definition 2.1 and Definition 2.2).

- `template<class GridType = Grid> class Nest;`
  Main class for managing the whole AMR hierarchy, including all levels, patch levels and flags (cf. Figure 4.2).

- `class PatchGrid;`
  PatchGrid class for managing the patches of a level. Each patchgrid contains a number of patches including patch relations such as child-parent and neighbors. Does not contain any cell data.

- `class Grid;`
  Grid class for managing the meshes of a level. Each grid contains a number of meshes, including mesh relations such as child-parent and neighbors. Meshes are (re-)created from a patchgrid by calling the regrid function.

- `class Mesh;`
  Mesh class for managing mesh data, which is stored internally in column-major format. Must allocate space for mesh data twice, since integration and flagging functions work in-place on meshes.

Usage of C++ templates allows for strong code modularization; even the grid and mesh classes can be easily replaced if the relevant methods are implemented (cf. Appendix A.1). This modularity allows for extension and usage of different numerical integration algorithms as well as swapping internal data structures if desired.

The `Nest` code can be used either for full AMR based simulation workflows or only as a mesh hierarchy generator. In the first case, a numerical solver as well as a flagging strategy

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.

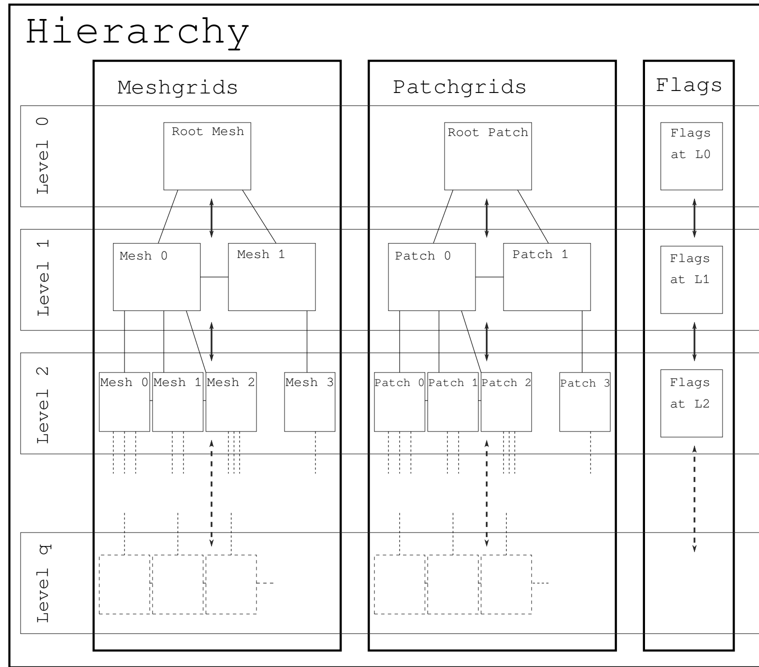


Figure 4.2: The AMR hierarchy implementation is modeled after Definition 2.2: It consists of multiple levels, each in turn containing meshes, patches and flags.

have to be supplied as callables with signatures as given in Code Sample 4.1. Modern C++ gives the user complete freedom on the exact implementation, as the callables can be provided as standard functions, lambdas or custom classes. To allow efficient in-place timestepping of mesh data as well as time interpolation during synchronization (Section 2.2.5), meshes need to provide duplicate data storage containing the current and last mesh data values as well as a mechanism to swap previous and current values.

Code Sample 4.1: Timestepping and Flagging.

```
auto timestep = [](Mesh& mesh_,          // input mesh
                   Double3D start,       // start coordinate
                   Double3D delta,       // spatial resolution
                   double t_,            // current time
                   double k_             // temporal stepsize
                  ) -> void {...}

auto flag = [](Mesh& mesh,            // input mesh
               Double3D delta         // spatial resolution
              ) -> std::vector<Flag> {...}
```

Internally, the flags are stored per-level in a hash-map to facilitate fast access to individual flags and allow for the addition of buffer flags as well as flag propagation. Flags are added and removed directly by calling the `Flags::set` and `Flags::unset` methods. It is possible to flag a cell for refinement by more than one level, in which case the corresponding cells are marked at the lower levels. Note that flag positions are indexed by their global grid coordinates consistent with the respective level.

For a full simulation, the hierarchy class handles the complete flagging and regrid process

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

via the `generate(…)` function (cf. Code Sample 4.2). However, if the nested mesh generation is used separately, it needs to be passed a `Nest::PatchGrid` as well as `Nest::Flags`.

Code Sample 4.2: Patch Hierarchy (Re-)Generation.

```
void generate(Flags* flags_ptr,               // new flags
              PatchGrid* grid_ptr,            // existing patches
              NestingMode mode                // either TopDown or BottomUp
             );
```

The level pointed to by `grid_ptr` is not modified, while the levels below are cleared and a new patch hierarchy covering all new flags is created. To ensure that the creation of a valid patch hierarchy is possible, the flags need to be propagated upwards, including additional nesting-induced buffer cells if using the top-down clusterer (cf. Remark 3.10). This is done by calling `propagate_flags` (cf. Code Sample 4.3) before `generate`. The patch hierarchy created by either top-down or bottom-up clustering is then guaranteed to be nested correctly. When using the bottom-up clusterer and regridding the full hierarchy, or if a buffer zone large enough to always cover enough cells around the lower levels is added, no flag propagation is necessary.

Code Sample 4.3: Flag Propagation.

```
bool propagate_flags(
    Grid*& grid_ptr,                    // grid at current level
    PatchGrid*& patchgrid_ptr,          // patchgrid at current level
    Flags*& flags_ptr,                  // flags at current level
    NestingMode nesting_mode            // either TopDown or BottomUp
);
```

A code sample of a full AMR simulation workflow as described in Section 5.2 is given in Appendix A.2.

4.3 Parallelization

AMR readily lends itself to parallelization due to the fact that the meshes are timestepped individually. Since the mesh boundaries need to be kept synchronized, difficulties can arise when scaling to a large parallelization degree, depending on how the meshes and their connectivities are managed. A number of methods have been developed for highly scalable AMR on large scale distributed-memory computer systems [18, 25, 30, 31].

The goals here are different, targeting shared-memory parallelization for workstation-class computers instead of distributed memory systems. Another difference lies in the algorithmic constraints: The clustering algorithm described in this work receives as input an existing sparse data structure containing all flag positions. The algorithms used in, e.g., [20, 25] parallelize based on the existing meshes, never storing the complete list of flags at once. This leads to a very efficient algorithm especially for tile-based clustering, as the tile generation is done locally by each worker (i.e., thread).

Since our requirements dictate global metadata, i.e., centralized storage for flags, patches and patch connectivity, the local tile generation approach from [20] is not applicable here. Instead,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

the focus lies on global parallelization of the signature-based top-down clustering algorithm within the given algorithmic constraints.

Parallelization of the top-down algorithm can be accomplished by domain decomposition, which is inherently given by the clustering on each level but the first. Since the top level contains only a small fraction of the total cells (and total flags) and in turn makes up only a negligible fraction of the total computation time, the top level is not decomposed. Instead, the patch generation for this level is parallelized as follows: The processing of each new patch, generated by repeated splitting, is done in parallel when possible: The root patch is processed serially, then the two resulting patches in parallel, and so on. For every finer level, parallelization is achieved by first parallelly sorting the flags into the existing patches, then clustering all patches individually in parallel.

To optimize performance, patches are first sorted in descending order by amount of flags contained within and then scheduled dynamically. A balanced distribution of the workload is only guaranteed if the flags are either evenly distributed or the number of meshes is significantly higher than the available processor cores, as is the case here. The time spent on merging and shrinking the resulting patches is insignificant in comparison to sorting and clustering, these steps are done serially.

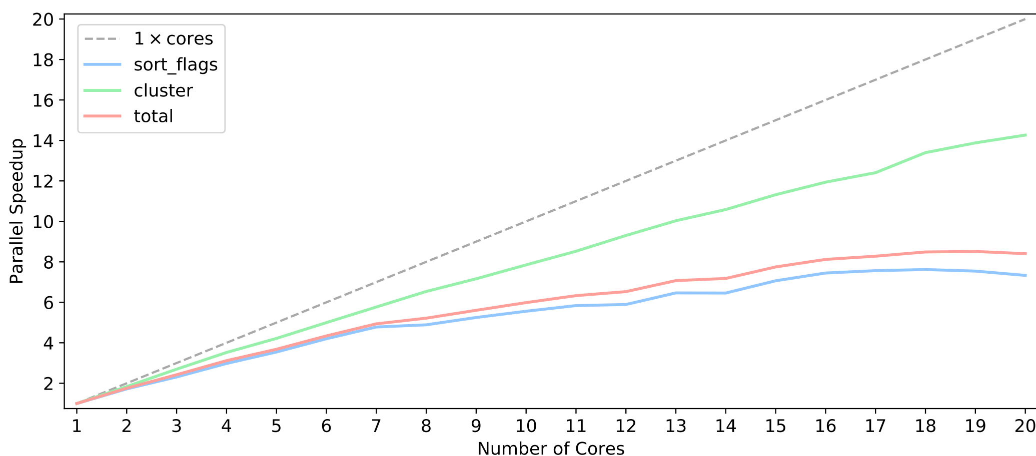


Figure 4.3: Parallel speedup of the clustering algorithm for the circular interface from Example 3 with root grid size 64×64, five levels and a refinement factor of $r = 4$. For each level $\mathcal{L}^{(l)}$ cells less than $0.05 + 4 \left\| n^{(l)} \right\|^{-1}$ away from the interface are flagged, for a total of $\approx 3.4 \times 10^6$ flags.

The parallel efficiency is evaluated via Example 3 (will be discussed in detail in the next section) at 5 levels with a total of $\approx 3.4 \times 10^6$ flags using a dual socket Intel Xeon Gold 6248 (Cascade Lake) benchmarking platform (2×20 cores). Thread pinning was used to pin all threads to a single processor and only the generation of the patch hierarchy is included in the timings, no flag propagation or mesh initialization. The clustering algorithm scales quite well with additional cores (cf. Figure 4.3) with a maximum speedup of ≈ 14 for 20 cores. The flag-sorting scales worse, with a maximum speedup of ≈ 7.5. In total, the complete patch generation algorithm receives a maximum speedup of ≈ 8.5 for 20 cores.

The clustering algorithm within each patch is completely independent from a computational perspective and will parallelize well as long as the number of meshes is significantly larger than

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

the number of cores to allow for efficient load balancing. The sorting procedure is limited by memory bandwidth and the need for concurrent access of the global flag data structure.

Table 4.1 shows that the most time intensive steps are sorting and clustering, while computation of patch connectivities (marking parent / child and neighbor relations), merging and shrinking is negligible.

num_threads	sort_flags	cluster	connectivity	shrink	merge	total
1	0.5369	0.3765	0.0052	0.0026	0.0008	0.9251
2	0.3118	0.2050	0.0054	0.0026	0.0009	0.5269
3	0.2322	0.1396	0.0054	0.0027	0.0009	0.3818
4	0.1802	0.1069	0.0055	0.0026	0.0009	0.2968
5	0.1515	0.0893	0.0057	0.0027	0.0009	0.2513
6	0.1278	0.0753	0.0058	0.0027	0.0009	0.2132
7	0.1123	0.0653	0.0057	0.0027	0.0009	0.1874
8	0.1098	0.0576	0.0058	0.0027	0.0009	0.1773
9	0.1023	0.0525	0.0059	0.0027	0.0009	0.1650
10	0.0966	0.0480	0.0059	0.0027	0.0008	0.1545
11	0.0920	0.0442	0.0059	0.0027	0.0008	0.1461
12	0.0911	0.0405	0.0059	0.0027	0.0009	0.1417
13	0.0830	0.0375	0.0060	0.0027	0.0010	0.1308
14	0.0831	0.0356	0.0060	0.0027	0.0009	0.1289
15	0.0760	0.0333	0.0059	0.0027	0.0009	0.1194
16	0.0721	0.0315	0.0059	0.0027	0.0010	0.1139
17	0.0710	0.0303	0.0060	0.0027	0.0010	0.1117
18	0.0705	0.0281	0.0061	0.0027	0.0010	0.1090
19	0.0712	0.0271	0.0060	0.0027	0.0010	0.1087
20	0.0732	0.0264	0.0060	0.0027	0.0010	0.1101

Table 4.1: Timings (in seconds) for parallelized clustering of the benchmark problem presented in Figure 4.3. Times are averaged over 10 independent runs.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5 Numerical Experiments

Based on the implementation discussed in the previous section, this section presents detailed results of numerical experiments to investigate the feasibility of the algorithms developed in Section 3. The two mesh generation algorithms are evaluated with several representative examples and compared to each other. Finally, a vortex flow simulation gives insights into the algorithms performance as part of the full AMR workflow.

5.1 Clustering Examples

In the following, single-level and multi-level clustering examples are presented for top-down and bottom-up clustering algorithms. The results justify our adaptations to the signature clustering algorithm and provide a basis for choosing between top-down and bottom-up approaches.

The quality of a clustering is judged in terms of cells refined relative to cells flagged (cf. Definition 3.2), also taking into account the number of ghost cells (cf. Definition 3.3). In order to also account for varying computational effort between inner cells and ghost cells as well as additional overhead for the number of meshes, a mesh clustering is identified to be most efficient if it minimizes the cost function ω from Definition 3.4 for given weights for ghost cell effort C_g and mesh overhead C_m. This only corresponds to minimizing the pure efficiency ϵ if $C_m = C_g = 0$ or to minimizing the adjusted efficiency $\tilde{\epsilon}$ if $C_g > 0$ and $C_m = 0$.

In practice, the weights C_g and C_m should be chosen in accordance with the performance characteristics of the numerical integration algorithm used as well as simulation parameters such as regridding period and refinement factor.

For the examples discussed in this section, no computational overhead for additional meshes (i.e., $C_m = 0$) and half the effort required for ghost cells compared to inner mesh cells (i.e., $C_g = 1/2$) is assumed, if not otherwise specified.

5.1.1 Top-Down Signature Clustering

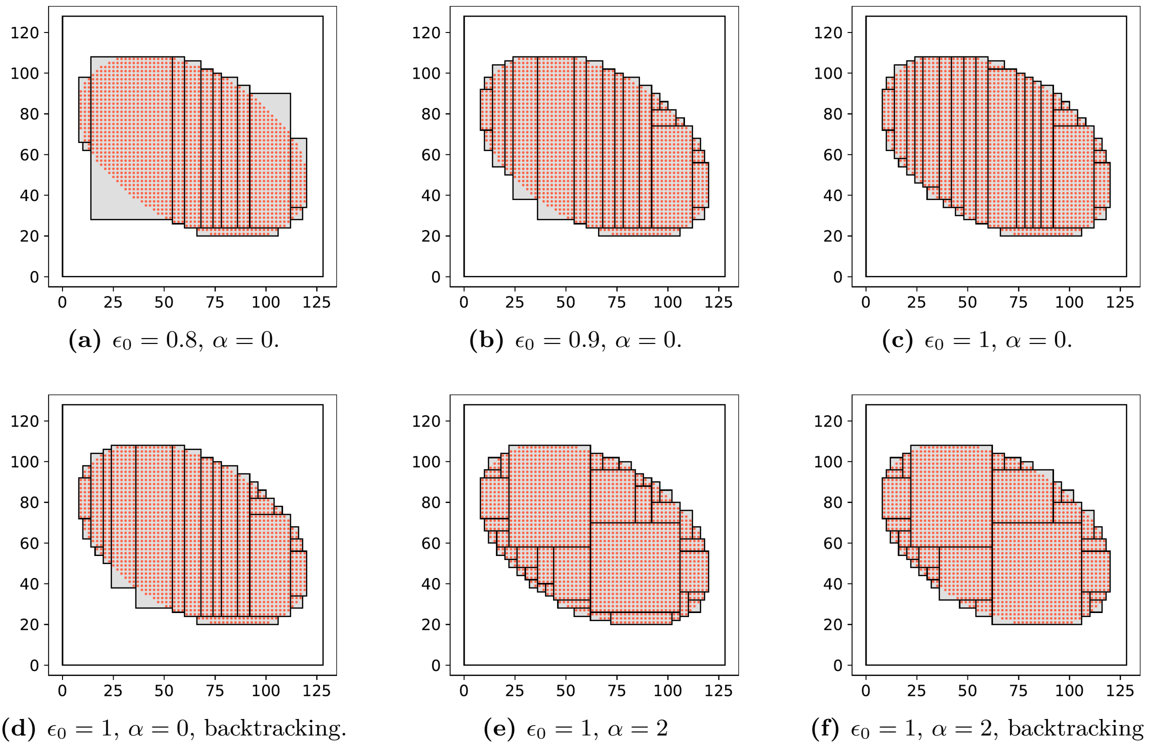
Example 2 (Elliptical and Circular Areas of Interest)**.**
On a single level grid of the domain $\Omega = [0, 1]^2$ with resolution 64×64 and refinement factor $r = 2$, flags are placed in the center of the grid in shapes of either a rotated ellipse or a circle. All cells $(x, y) \in \mathcal{G}_\Omega$ satisfying

$$\frac{((x - c_x)\cos(\phi) + (y - c_y)\sin(\phi))^2}{a^2} + \frac{((x - c_x)\sin(\phi) - (y - c_y)\cos(\phi))^2}{b^2} \leq 1,$$

are marked for refinement, where $c = (0.5, 0.5)$, $a = b = 0.4$, $\phi = 0$ for the circle and $a = 0.3$, $b = 0.5$, $\phi = \pi/3$ for the ellipse. These example geometries (cf. Figure 5.1 and Figure 5.2) showcase clustering qualities, particularly the benefits of aspect ratio correction.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

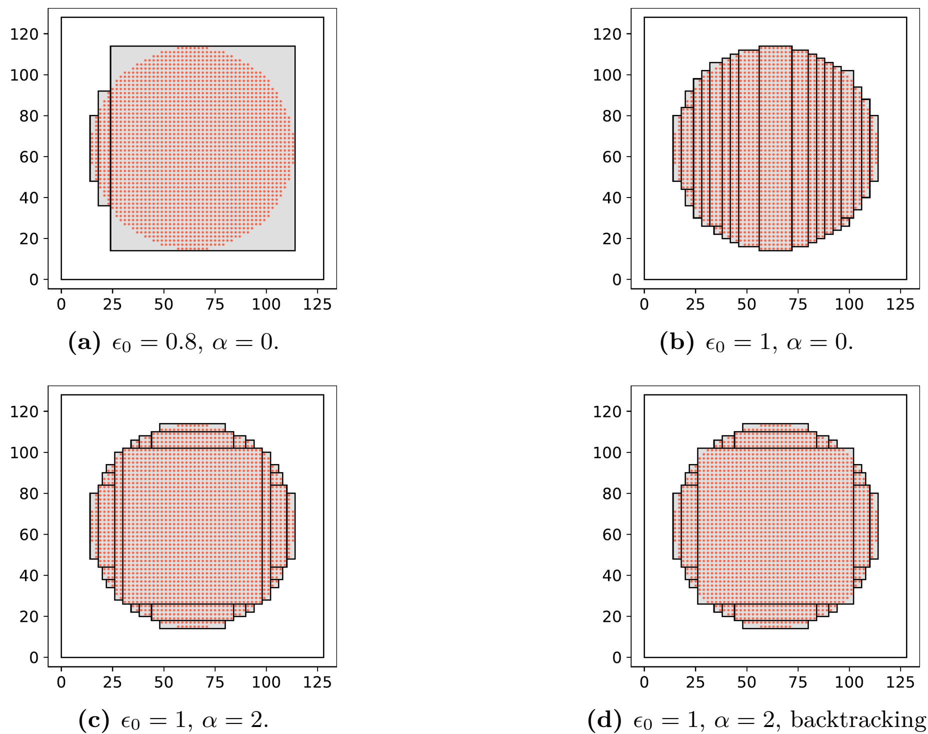
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) $\epsilon_0 = 0.8$, $\alpha = 0$. **(b)** $\epsilon_0 = 0.9$, $\alpha = 0$. **(c)** $\epsilon_0 = 1$, $\alpha = 0$.

(d) $\epsilon_0 = 1$, $\alpha = 0$, backtracking. **(e)** $\epsilon_0 = 1$, $\alpha = 2$ **(f)** $\epsilon_0 = 1$, $\alpha = 2$, backtracking

Figure 5.1: Signature clustering results for an elliptical area of interest for different parameters with minimum width $m_0 = 4$. Increasing the target efficiency reduces inner cell count but increases total cell count and creates irregular, elongated patches. A combination of aspect ratio correction $\alpha = 2$ and backtracking minimizes the number of total cells.



(a) $\epsilon_0 = 0.8$, $\alpha = 0$. **(b)** $\epsilon_0 = 1$, $\alpha = 0$.

(c) $\epsilon_0 = 1$, $\alpha = 2$. **(d)** $\epsilon_0 = 1$, $\alpha = 2$, backtracking

Figure 5.2: Signature clustering results for a circular area of interest for different parameters with $m_0 = 4$. The standard Berger-Rigoutsos algorithm creates very long and thin patches, whereas aspect ratio correction combined with backtracking results in a significantly lower mesh cell count.

Single-Level Clustering

Figure 5.1 shows different clusterings of the elliptical area of interest from Example 2 at a single level. With a low target efficiency the classical Berger-Rigoutsos clustering algorithm stops too soon and generates only a small number of meshes n_m. The resulting clustering highly varies in patch size and still contains a high number of unnecessary cells. The reason for this is the locality of the termination condition: splitting continues until the target efficiency is reached for each individual patch, meaning a large patch may still contain a large portion of unflagged cells compared to the rest of the clustering, while smaller patches are split even if the savings in cell count are relatively small compared to the total cell count. This is shown in Figure 5.1a, where a single patch contains more than half of the unflagged but refined cells.

Without the aspect ratio correction ($\alpha = 0$) and backtracking, a higher target efficiency causes generation of a large number of very long patches; this does reduce inner cell count n_i but is still suboptimal, because these patches have a large number of ghost cells n_g relative to inner cells, resulting in a higher total cell count n_c than with lower target efficiency.

This behaviour is even more apparent on a completely circular flagged area (cf. Figure 5.2). Without aspect ratio correction, maximum split magnitudes along X-axis and Y-axis are the same, causing splits to be taken almost exclusively along the X-axis.

Since the last splits make the final clustering worse than not splitting, clustering with backtracking b results in the same output as with a lower target efficiency, without having to guess the optimal target efficiency for the given input. An even better clustering in terms of total cell count is achieved with the aspect ratio correction $\alpha = 2$, resulting in more regular, square patches.

In Table 5.1 and Table 5.2, the best result (i.e., the lowest objective function ω) is shown to be achieved by a combination of aspect ratio correction and backtracking in both examples. This is not guaranteed to be a global minimum when considering all possible clusterings but in general an acceptable approximation. Note that the best clustering in terms of minimizing ω does not necessarily coincide with the lowest pure efficiency ϵ, as pure efficiency does not take the ghost cells into account. The circular example in Figure 5.2 shows that the clustering algorithm is not symmetry preserving - it does not necessarily generate a symmetric patch covering, even if the input flags are perfectly symmetric.

	ϵ_0	α	b	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
Figure 5.1a	0.8	0	No	7216	13	9880	8128	1752	0.8878	0.7304	9004
Figure 5.1b	0.9	0	No	7216	23	9748	7512	2236	0.9606	0.7403	8630
Figure 5.1c	1.0	0	No	7216	31	10124	7388	2736	0.9767	0.7128	8756
Figure 5.1d	1.0	0	Yes	7216	24	9752	7504	2248	0.9616	0.7400	8628
Figure 5.1e	1.0	2	No	7216	42	9232	7364	1868	0.9799	0.7816	8298
Figure 5.1f	1.0	2	Yes	7216	33	9016	7440	1576	0.9699	0.8004	8228

Table 5.1: Clustering statistics for the signature clusterings of the elliptical example from Figure 5.1.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

	ϵ_0	α	b	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
Figure 5.2a	0.8	0	No	7744	3	10052	9464	588	0.8183	0.7704	9758
Figure 5.2b	1.0	0	No	7744	22	10952	7936	3016	0.9758	0.7071	9444
Figure 5.2c	1.0	2	No	7744	27	9644	7952	1692	0.9738	0.8030	8798
Figure 5.2d	1.0	2	Yes	7744	25	9380	7984	1396	0.9699	0.8256	8682

Table 5.2: Clustering statistics for the signature clusterings of the circular example from Figure 5.2.

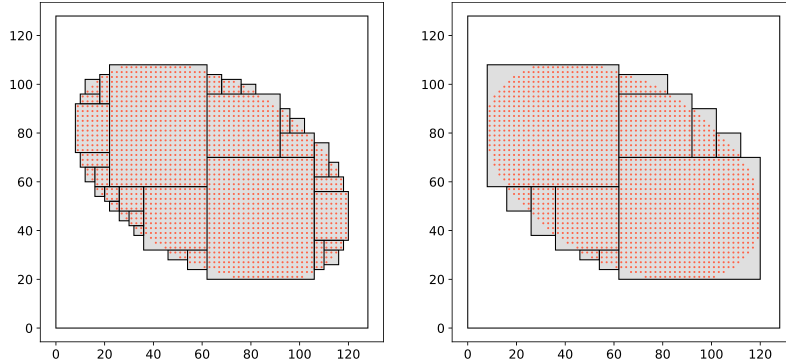
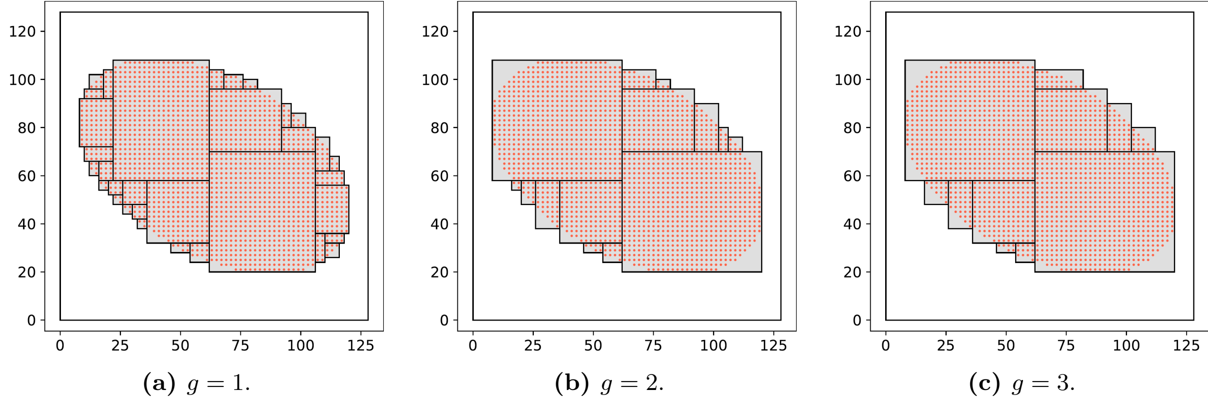


Figure 5.3: Clusterings with and without mesh count penalty for $\epsilon_0 = 1$, $\alpha = 2$ and backtracking. On the left, clustering was done with no mesh overhead $C_m = 0$. The right figure shows the clustering with $C_m = 10$, resulting in a clustering with less patches but more mesh cells.

To simulate a small overhead per mesh created, an additional penalty for mesh number (i.e., $C_m = 10$) is added to the cost function ω. This causes the backtracking algorithm to prefer a slightly different clustering which reduces the number of meshes from $n_m = 33$ to $n_m = 11$ at the cost of increasing the inner cell count by $\approx 5\%$ (cf. Figure 5.3). Note that the choice of C_m and C_g does not influence where each split is taken, only how the procedure determines at what point to stop splitting up patches (cf. Algorithm 3.3).

Generating patches with sides of similar lengths is even more important if the ghost layer size g is increased. A larger ghost layer increases the amount of mesh cells required as boundary conditions on each individual patch by a factor g. Figure 5.4 shows clusterings created with $g \in \{1, 2, 3\}$. With larger ghost layers, less patches are created since the additional ghost cells created by splitting outweigh the savings from reducing inner cell count. As with the cost function weights C_m and C_g, this does not influence the choice of split index: the clusterings from Figure 5.4a and Figure 5.4b can be obtained from Figure 5.4c by further splitting the corresponding patches.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub



(a) $g = 1$. **(b)** $g = 2$. **(c)** $g = 3$.

Figure 5.4: Clusterings of the elliptical example for different ghost layer sizes. For $g = 1$, a large number of patches are generated, cutting away most of the unmarked cells. For $g = 2$, the number of ghost cells per patch doubles, making it more efficient to create only a small number of patches. Further increasing ghost layer size to $g = 3$ again reduces patch count.

In the context of level-set methods, the areas of interest are usually not large contiguous areas (or volumes) but rather small, elongated shapes around the 0-level-set Γ.

Example 3 (Circular Interface)**.**
Consider a single- or multi-level grid hierarchy discretizing the physical domain $\Omega := [0, 1]^2$, with a resolution of 128×128 on the root level. The circular interface Γ is defined by

$$\Gamma := \{\mathbf{x} \in \Omega : (x_0 - 0.5)^2 + (x_1 - 0.5)^2 = 0.3^2\}.$$

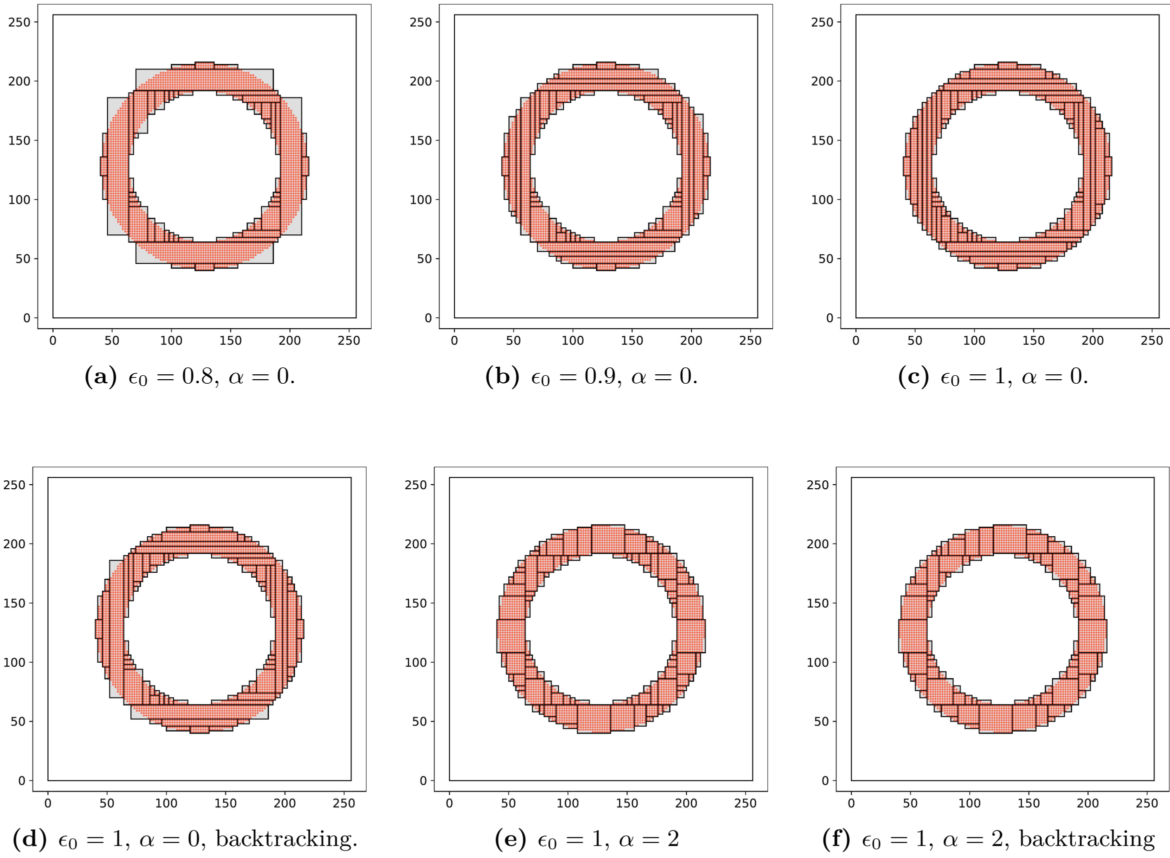
Grid cells around the interface are flagged for refinement based on their distance to the interface and their level $\mathcal{L}^{(l)}$, $l \in 0, \dots, q$ (cf. Figure 2.7)

$$R^{(l)}_{ijk} := \begin{cases} 1, & \text{if } d(\mathbf{x}_{ijk}, \Gamma) < \frac{10}{\|n^{(l)}\|_\infty}, \\ 0, & \text{else.} \end{cases}$$

The same problem is chosen in [25] for benchmarking single-level clustering algorithms as it resembles an expanding blast-wave which is representative of problems commonly solved with AMR simulation.

Figure 5.5 shows clusterings of the initial circular interface from Example 3. While the difference in total cell count is not as big as in previous examples, aspect ratio correction does yield a clustering with much more regular patches. With $\alpha = 2$ and backtracking (cf. Figure 5.5f), the total cell count is reduced by $\approx 7\%$ compared to the clustering without backtracking and aspect ratio correction (cf. Figure 5.5d). In addition to reducing the cell count, the relative uniformity and regularity of the aspect ratio corrected clustering can be beneficial for load balancing during parallelization [20, 30].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub



(a) $\epsilon_0 = 0.8$, $\alpha = 0$. **(b)** $\epsilon_0 = 0.9$, $\alpha = 0$. **(c)** $\epsilon_0 = 1$, $\alpha = 0$.

(d) $\epsilon_0 = 1$, $\alpha = 0$, backtracking. **(e)** $\epsilon_0 = 1$, $\alpha = 2$ **(f)** $\epsilon_0 = 1$, $\alpha = 2$, backtracking

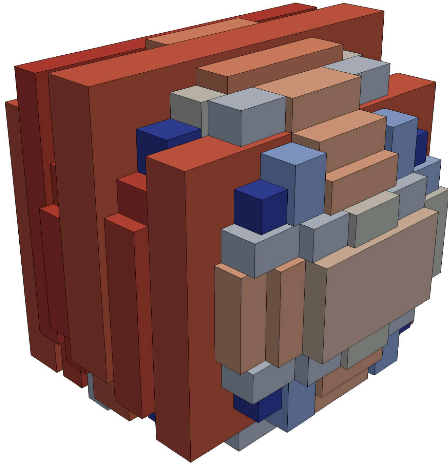
Figure 5.5: Signature clustering of the circular interface (Example 3) with different parameters. The standard Berger-Rigoutsos algorithm (i.e., $\alpha = 0$) favors long but thin patches. Combining aspect ratio correction with backtracking results in the most efficient clustering in terms of total cell count.

The presented algorithms can be generalized for arbitrary dimensions, but the focus of this work is on 2D and 3D problems. Figure 5.6 shows the results from clustering a spherical area of interest. With a lower target efficiency, the generated patches are highly irregular; a large number of long, flat patches is created. Increasing the target efficiency lowers the inner cell count at the cost of increased mesh number and ghost cell count. Adding aspect ratio correction again results in more regular, cube-like patches as in the 2D case. The best clustering is produced with target efficiency $\epsilon_0 = 1$, aspect ratio correction $\alpha = 2$ and backtracking (cf. Table 5.3). This clustering, shown in Figure 5.6d, decreases the total cell count by 24424 to $n_c = 598080$ compared to Figure 5.6a as the split locations are improved and suboptimal patch splits are discarded.

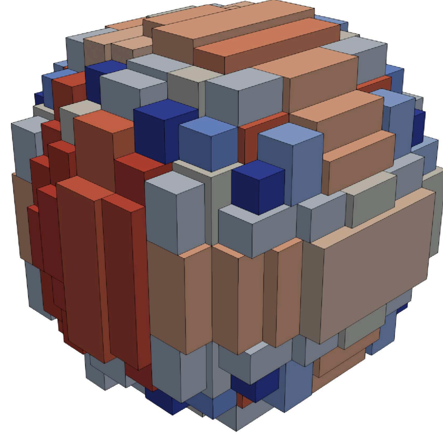
The signature clustering algorithm behaves very similarly in two and three dimensions; for three dimensions, the algorithm performs the same actions as in two dimensions, but also considers splits along the additional axis. Therefore, the focus is mostly on 2D comparisons, which are easier to generate, visualize and analyze; the results are expected to be, in general, transferable to the 3D case.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

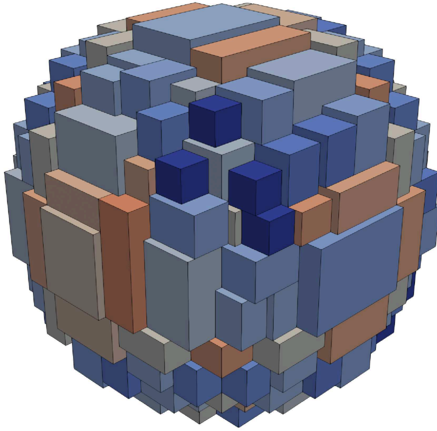
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.



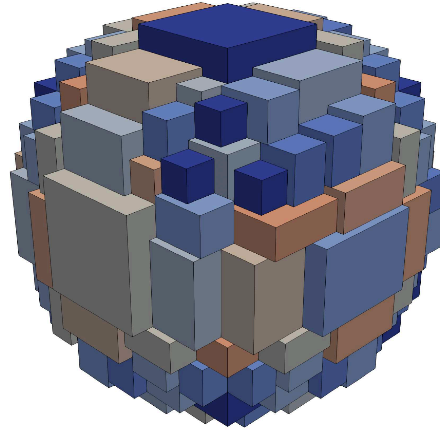
(a) Target efficiency 0.8.



(b) Target efficiency 1.



(c) Target efficiency 1 and aspect ratio correction.



(d) Aspect ratio correction and backtracking.

Figure 5.6: Signature clustering of spherical area of interest with different parameters. Patches are colored by ratio of shortest to longest side, with blue being a high ratio and red a low ratio. As in the 2D case, a combination of aspect ratio correction and backtracking yields the best results in terms of total cell count.

ϵ_0	α	b	n_f	n_m	n_c	n_g	ϵ	$\tilde{\epsilon}$	ω
0.8	0	No	254272	102	622504	217816	0.8169	0.6052	731412
1.0	0	No	254272	163	588320	266840	0.8644	0.5947	721740
1.0	2	No	254272	181	594768	248256	0.8550	0.6032	718896
1.0	2	Yes	254272	161	598080	231440	0.8503	0.6131	713800

Table 5.3: Clustering statistics of a spherical area of interest (cf. Figure 5.6); comparing results for different parameter combinations.

Multi-Level Clustering

In a multi-level-setting, the top-down clustering method may require flagging additional cells around the lower level flags to ensure the finer meshes are properly embedded. The number of nesting flags added, $k = \max\left(\frac{m_0}{r^2}, 1\right)$, depends only on the refinement factor r and the minimum width m_0 (cf. Section 3.2.1).

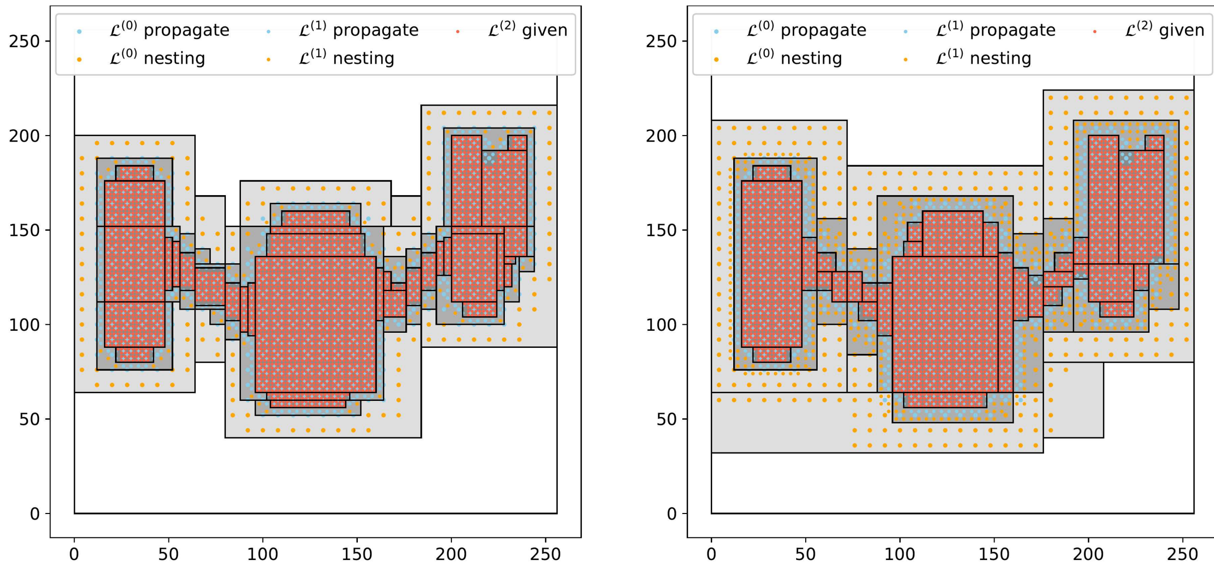


Figure 5.7: Top-Down clustering of Example 3.1. The left figure shows the patch hierarchy with a minimum width of $m_0 = 4$. This results in a required nesting-induced buffer zone of $k = 1$ added around lower level flags. For a higher minimum width $m_0 = 8$ (on the right), the required nesting flags increase to $k = 2$. Most cells only required for nesting are removed by shrinking.

Figure 5.7 shows a complete patch hierarchy for Example 1. Here, only the lowest level is flagged and flags need to be propagated upwards, as otherwise no patches would be placed on top levels. For $m_0 = 4$ a single additional nesting-induced flag is placed around the flags of finer levels; for $m_0 = 8$, this layer is twice as large. While many of the cells only required for nesting are afterwards cut off to reduce cell count, they do influence the choice of split location, leading to the creation of larger patches for larger minimum widths which may impact clustering efficiency negatively.

During an AMR simulation, this is does not occur often as the error (or other properties on which the flag selection is based) is expected to be continuous and as such flags on finer levels will in most cases be covered sufficiently.

The effect of generating slim patches is drastically amplified by the nesting method (Algorithm 3.6) when using the signature clustering algorithm for creating a multi-level hierarchy. Since patches are split at the edges, a long and narrow parent patch could result in the generation of up to three long and even narrower child patches.

Figure 5.8 shows the 4-level hierarchical clustering of the same circular interface as above without aspect ratio correction. No additional nesting flags are required here, as the area flagged on each level is already larger than required. As in Figure 5.5, the top level is split up into very

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

long, thin patches. To ensure nesting, especially (N1), these patches' children are split again along the shorter axis, resulting in even thinner patches. With $\alpha = 2$ (cf. Figure 5.9), the patch aspect ratios are kept mostly balanced, which results in a reduction in total mesh cell count n_c at all levels.

The clustering statistics are specified individually for each level in Table 5.4 and Table 5.5. The efficiency of the root level is $\epsilon = \tilde{\epsilon} = 0$ as no refinement flags exist for this level. Similarly, there are no flags at the lowest level, $n_f = 0$. Aspect ratio correction with $\alpha = 2$ causes an increase in adjusted efficiency $\tilde{\epsilon}$ as well as a decrease in total score ω when using aspect ratio correction. The adjusted splitting choices result in much more regular and uniform patches, reducing the combined cell count of the patch hierarchy by $\approx 4\%$ to 195364 but increasing the combined number of meshes n_m by $\approx 15\%$.

$\mathcal{L}$	n_m	n_f	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
0	1	4680	16900	16384	516	0.0000	0.0000	16642
1	56	9504	25204	20624	4580	0.9077	0.7427	22914
2	95	19120	53540	42772	10768	0.8888	0.7100	48156
3	194	0	106888	86068	20820	0.8886	0.7155	96478

Table 5.4: Clustering statistics for signature clusterings of the circular interface from Figure 5.8 (without aspect ratio correction, $\alpha = 0$).

$\mathcal{L}$	n_m	n_f	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
0	1	4680	16900	16384	516	0.0000	0.0000	16642
1	53	9504	24992	20852	4140	0.8978	0.7490	22922
2	101	19120	50780	42824	7956	0.8877	0.7486	46802
3	241	0	102692	85020	17672	0.8996	0.7448	93856

Table 5.5: Clustering statistics for signature clusterings of the circular interface from Figure 5.9 (with aspect ratio correction, $\alpha = 2$).

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

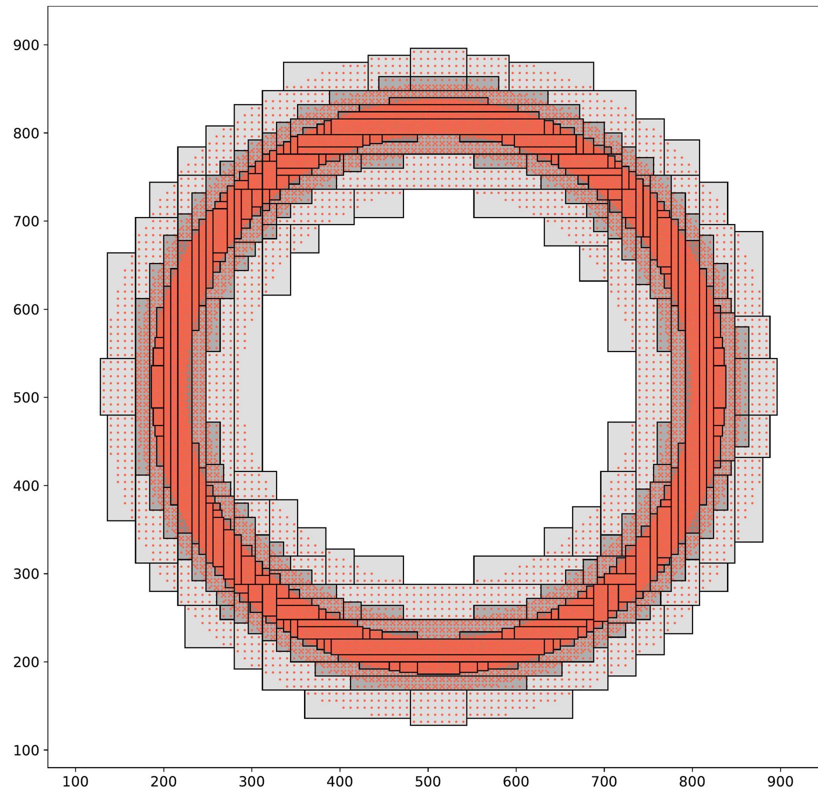


Figure 5.8: Top-Down hierarchy for the circular interface (Example 3) with 4 levels. Without aspect ratio correction, $\alpha = 0$, long and thin patches are created on the top level and in turn on the lower levels.

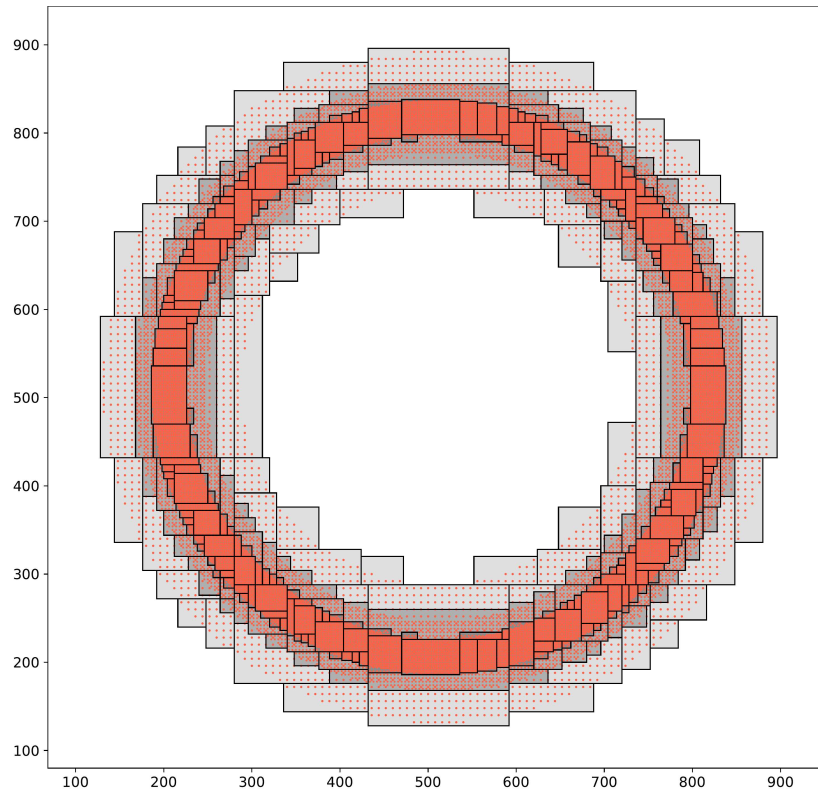


Figure 5.9: Top-Down hierarchy for the circular interface (Example 3) with 4 levels. With aspect ratio correction, $\alpha = 2$, the resulting patches are mostly square-like except where additional splits were required for proper nesting.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Another aspect is the choice of refinement ratio r. Commonly either $r = 2$ or $r = 4$ is used with the trade-off again being between mesh count and cell count. A higher refinement ratio needs fewer levels (and in turn fewer meshes) to reach the desired spatial resolution.

On the other hand, it increases cell counts if the area to be refined is the same between the two choices (i.e., the flagged cells depend only on their physical location and not the refinement ratio). Figure 5.10 compares top-down clustering of a hierarchy of 5 levels and refinement factor $r = 2$ to a hierarchy with 3 levels and refinement factor $r = 4$. Both root grids have a size of 128×128, yielding the same resolution on the lowest grid. Cells at level $\mathcal{L}^{(l)}$ are in both cases marked if their distance from the interface Ω is smaller than $10 \left\|\mathbf{n}^{(l)}\right\|_{\infty}^{-1}$.

The clustering statistics (cf. Table 5.6) are aggregated over the whole hierarchy as per Definition 3.9; the 5-level hierarchy with $r = 2$ has a lower total cell count as well as a lower ω-value at the cost of a significantly higher mesh count n_m than the 3-level hierarchy. Although it contains fewer total mesh cells, it exhibits a lower pure and adjusted mesh efficiency as more ghost cells are included.

This behaviour is expected in general: consider a square-shaped patch of size k. When refining the patch, it will have $k^2 r^2$ inner cells and $4(k+1)r$ ghost cells. This equals a ratio of $\frac{4(k+1)}{rk^2}$ of ghost cells to inner cells, meaning a clustering with a higher refinement factor will usually require less ghost cells compared to inner cells.

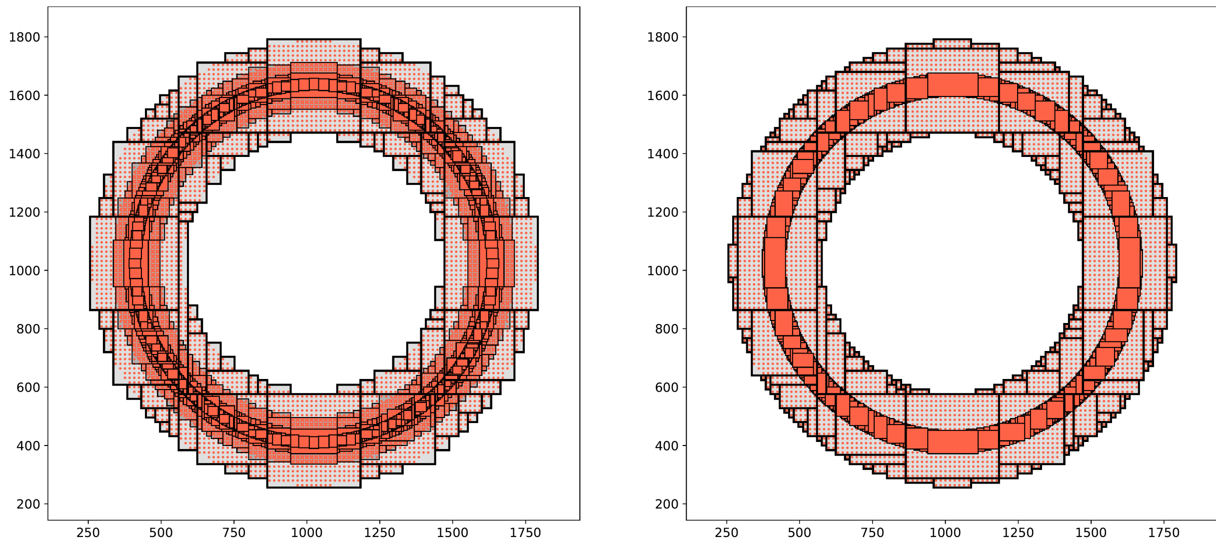


Figure 5.10: Signature clustering of the circular interface (Example 3) with refinement factors $r = 2$ (left) and $r = 4$ (right). A lower refinement factor of requires more levels (in this case 5 compared to 3) and in turn a higher number meshes to reach the same grid resolution. Still, the efficiency of the clustering is better with $r = 2$, as the impact of refining an unmarked cell is lower.

r	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
2	287136	1507	393628	316220	77408	0.9080	0.7295	354924
4	380800	955	456564	399712	56852	0.9527	0.8341	428138

Table 5.6: Clustering statistics for Figure 5.10 comparing refinement factors $r = 2$ and $r = 4$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

5.1.2 Bottom-Up Tile Clustering

The main benefit of tile clustering is the generation of equally sized patches, which is desirable especially for distributed-memory parallelization [25] as it greatly simplifies load balancing. The trade-off to make in this case is the choice of tile size d. Using large tiles implies fewer tiles but an increased number of cells, while more, smaller tiles are more efficient in terms of mesh count but require additional effort for keeping mesh boundaries in sync.

Figure 5.11 shows patch hierarchies for Example 3 with 4 levels generated by bottom-up clustering with tile sizes $d = 16$, $d = 32$ and $d = 64$. Patches are not merged but shrunk to decrease cell count where possible. The most efficient clustering with respect to both pure and adjusted efficiency, as well as ω-score for $C_g = 1/2$, $C_m = 0$, is achieved with $d = 24$ (cf. Table 5.7). This clustering contains a total of $n_m = 395$ meshes and $n_c = 207756$ mesh cells, of which $\approx 16\%$ are ghost cells.

While a smaller tile size of $d = 8$ does result in less inner mesh cells, the clustering contains six times as many patches and in turn almost three times as many ghost cells. With larger tile sizes, e.g., $d = 64$, the amount of patches and ghost cells is reduced, but the number of inner cells increases drastically again as many unneeded cells are included; the clustering is inefficient especially when assuming no mesh overhead, as is done here.

Therefore, the tile size should be chosen small if mesh overhead and ghost cell effort is relatively small. If mesh overhead is high compared to the effort required for each mesh cell, $C_m \gg 0$, bigger tile sizes will be more efficient. In general, the choice of an optimal tile size is also highly dependent on the particular problem geometry: If the flagged area features a lot of small-scale features, equally small tiles are needed for generating an efficient clustering.

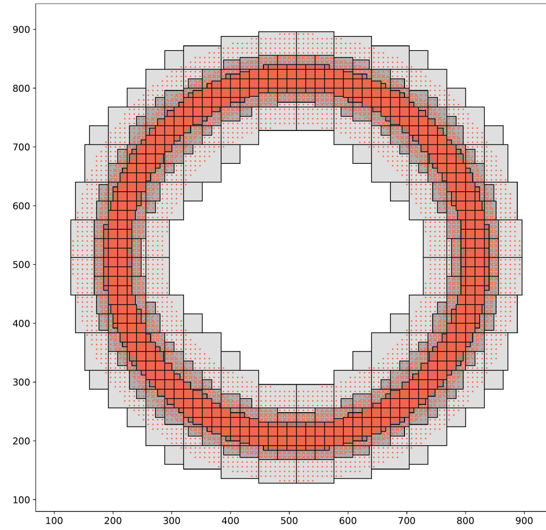
The bottom-up tile clustering algorithm is independent of dimension in the sense that a 3D clustering with tile size (d_x, d_y, d_z) is the same as separately clustering 2D slices of size d_z with tile size (d_x, d_y). As such, it is possible to evaluate clustering performance on 2D examples as the results are also applicable to the 3D case.

d	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
8	133216	2501	266900	176384	90516	0.7553	0.4991	221642
16	133216	773	218148	171408	46740	0.7772	0.6107	194778
24	133216	395	207756	174840	32916	0.7619	0.6412	191298
32	133216	257	210052	183008	27044	0.7279	0.6342	196530
48	133216	143	219460	198736	20724	0.6703	0.6070	209098
64	133216	97	244772	226624	18148	0.5878	0.5442	235698

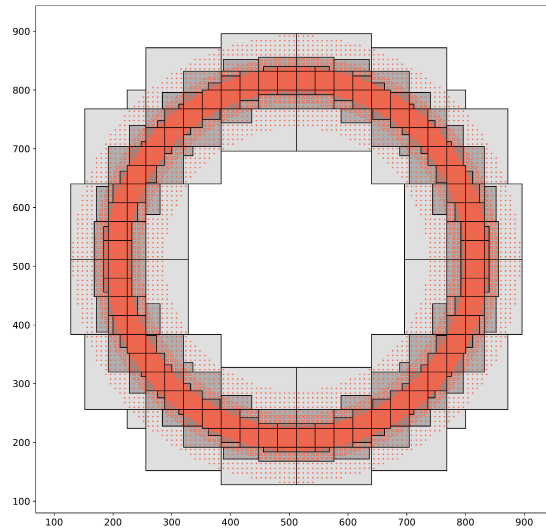
Table 5.7: Clustering statistics for bottom-up clustering of circular interface (Example 3) comparing the results for different tile sizes.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

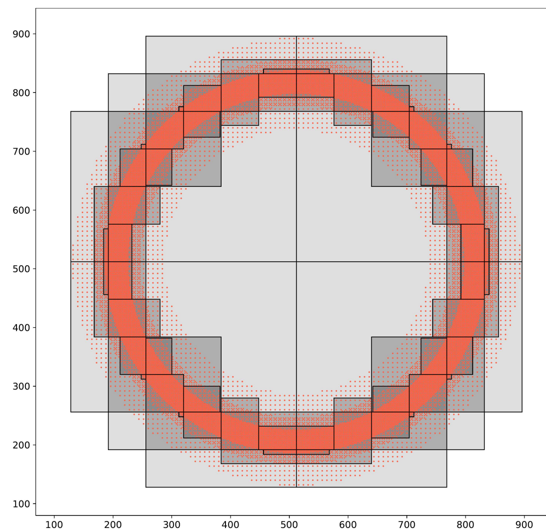
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN



(a) Tile size $d = 16$.



(b) Tile size $d = 32$.



(c) Tile size $d = 64$.

Figure 5.11: Patch hierarchy for the circular interface from Example 3 with 4 levels, generated by bottom-up tile clustering with different tile sizes. Patches are shrunk after clustering, but not merged.

5.1.3 Comparison of Signature Clustering and Tile Clustering

Since the goal of this work is mainly to reduce overall cell and mesh count, the bottom-up tile clustering algorithm generally yields worse results than the top-down signature clustering algorithm. Even if the mesh count is reduced by merging patches, tile clustering results in significantly more cells than signature clustering.

Figure 5.12 shows top-down and bottom-up clusterings for the hierarchy from Example 1. Although the top-down algorithm requires additional nesting cells which may negatively influence clustering efficiency on the individual levels, the resulting patch hierarchy is better in terms of both mesh count as well as cell count (cf. Table 5.8).

A similar result is shown in Figure 5.13, which compares top-down and bottom-up clusterings for the circular interface from Example 3 with 3 levels. Top-down clustering again performs better, with $\approx 5\%$ less mesh cells and $\approx 25\%$ less meshes (cf. Table 5.9).

As the merging strategy simply considers all neighbors in no particular order, the merging process is of course not optimal (though it always decreases mesh count and ghost cell count). An optimal merging strategy in this context is not realistic, as it is essentially the same problem as the original clustering problem: If a suitable merging strategy existed, it could be used directly on the individual cells. On the other hand, not merging at all would result in a lower quality clustering according to the metrics used here, because merging will always increase pure and adjusted efficiency and will reduce cost as superfluous ghost cells are removed and the mesh count is decreased.

Signature clustering as presented here has the additional advantage of not having to choose a problem-dependent control parameter, such as target efficiency or tile size, making the algorithm more resilient to differing problem geometries without the need for manual adjustments. For our use-case of an AMR simulation on a shared-memory computation architecture, top-down clustering produces more suitable nested mesh hierarchies. Bottom-up clustering should be used mainly if patch uniformity or symmetry is important, or if the use-case requires the patch generation to be much faster than the signature clustering algorithm.

	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
Top-Down	19804	61	26508	21944	4564	0.7471	0.6374	24226
Bottom-Up	19804	64	27924	23144	4780	0.7092	0.6056	25534

Table 5.8: Clustering statistics for patch hierarchies of Example 1 from Figure 5.12.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

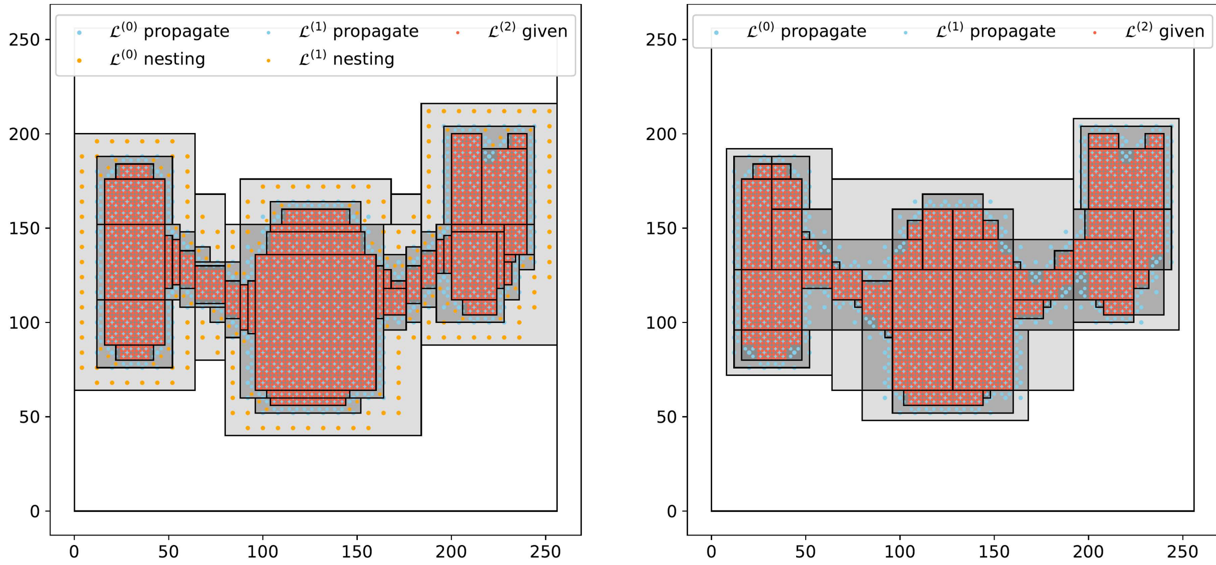


Figure 5.12: Patch hierarchies for Example 1 from top-down clustering (left) and bottom-up clustering (right). Bottom-up clustering with tile size $d = 8$ results in an equal number of patches after merging, but a higher total cell count compared to the top-down clustering.

	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
Top-Down	56736	276	90920	75848	15072	0.7480	0.6240	83384
Bottom-Up	56736	354	95920	76560	19360	0.7411	0.5915	86240

Table 5.9: Clustering statistics for patch hierarchies of circular interface (Example 3) from Figure 5.13.

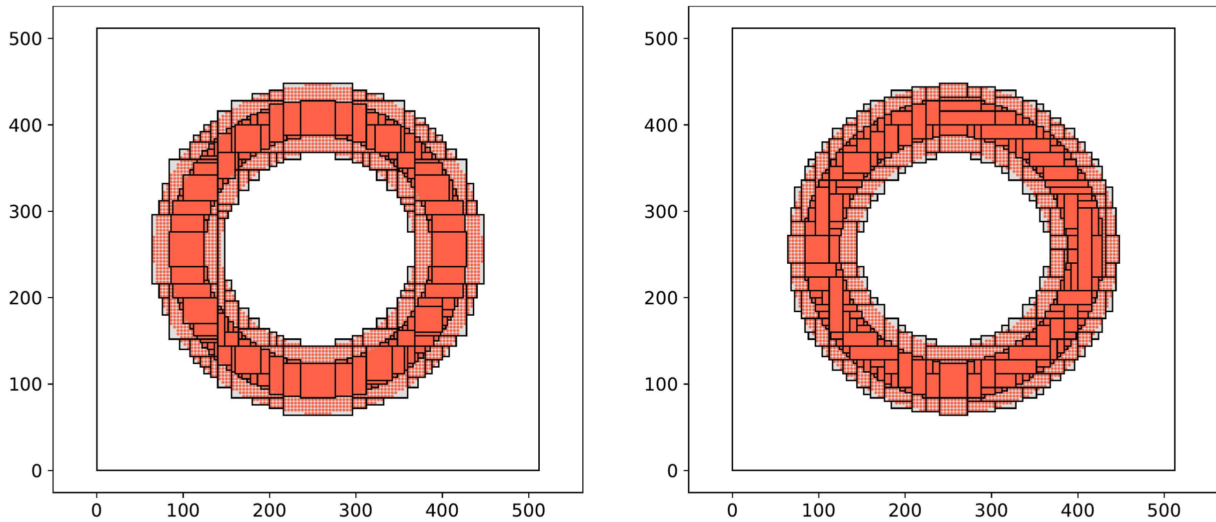


Figure 5.13: Patch hierarchies for circular interface (Example 3) with 3 levels, comparing the results of top-down clustering (left) with those from bottom-up clustering (right). Bottom-up clustering with tile size $d = 8$ generates a higher number of patches after merging and a higher total cell count compared to the top-down approach.

5.2 Vortex Flow Simulation

A full AMR simulation is performed based on an example given in [10], the deformation of a circular interface

$$\Gamma_0 := \{\mathbf{x} \in [0,1]^2 : (x_0 - C_0)^2 + (x_1 - C_1)^2 = R^2\}$$

with radius $R = 0.1$ and center $\mathbf{C} = (0.5, 0.75)$ by a vortex flow

$$\begin{aligned} u_t(\mathbf{x},t) + \begin{pmatrix} \sin(\pi x_0)^2 \sin(2\pi x_1) \\ -\sin(\pi x_1)^2 \sin(2\pi x_0) \end{pmatrix} \nabla u(\mathbf{x},t) &= 0 && \text{on } [0,1]^2 \times \mathbb{R}^+, \\ u(\mathbf{x},0) &= d_\pm(\mathbf{x},\Gamma_0) && \text{on } [0,1]^2. \end{aligned} \tag{14}$$

The above equation is advanced in time from $t_0 = 0$ to $t_{end} = 2$ starting from the initial condition $u(\mathbf{x},0)$. Top-down clustering is used to create a nested mesh hierarchy of 5 levels with a refinement factor of $r = 2$, doubling the resolution per level.

At every 4th timestep on each level $\mathcal{L}^{(l)}$, $l < 4$, all cells where $|u(\mathbf{x},t)| < 4 \left\|\mathbf{n}^{(l)}\right\|_\infty^{-1}$ are flagged for refinement and clustered into a new patch hierarchy; then, the new meshes are initialized and the process repeats. To ensure that the area of interest on a finer grid does not move out of the area covered by coarse patches and causes a premature regrid, a buffer zone of size 4 is added around the flagged cells on all but the lowest level.

Individual meshes are integrated using the upwind scheme (4). The root mesh has a resolution of 128×128 and the finest level has a resolution of 2048×2048. Figure 5.14 shows the time evolution of the level-set function $u(\mathbf{x},t)$, the interface $\Gamma_t = \{\mathbf{x} \in \Omega : u(\mathbf{x},t) = 0\}$ and the patch hierarchy at $t \in \{0,1,2\}$. No re-distancing or other level-set methods are used here.

Table 5.10 shows the mesh hierarchy statistics for the vortex flow simulation. Even though the flagged area is much larger in the coarser levels, the number of mesh cells increases as the resolution increases. On average, $\approx 30\%$ of all cells and $\approx 60\%$ of all meshes are located at the finest level. The efficiency is lowest for the finest level, as the flagged region at this level is often smaller than the minimum patch size $m_0 = 8$ since no buffer zone is included at this level.

A simulation using a single mesh with the same size as the root mesh does not provide a usable solution, since the 0-level-set vanishes completely at about $t = 1.5$. Using a single grid with the same resolution as the finest level of the hierarchy would equal a total of $(128 \cdot 2^4)^2 = 4194304$ cells. Our grid hierarchy only contains ≈ 178000 cells on average, which corresponds to a reduction in cells by $\approx 95\%$. This depends of course on how the area of interest is determined as well as on the size of the buffer zone but will in general significantly reduce the time of computation as long as the majority of the domain does not need to be refined.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

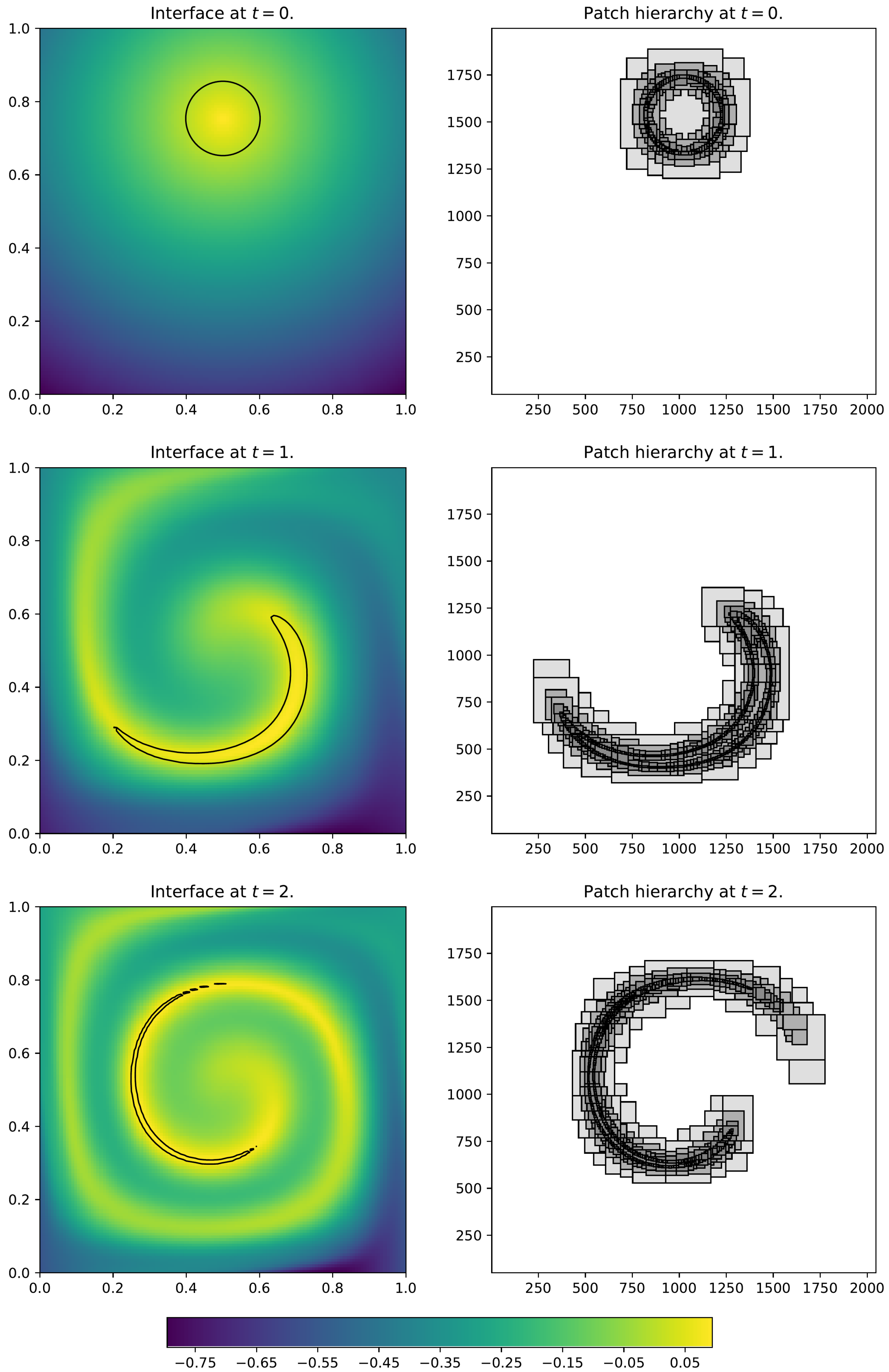


Figure 5.14: Evolution of the interface Γ_t, the level-set function u and the corresponding nested mesh hierarchy from $t = 0$ to $t = 3$. Every 4 timesteps, new mesh hierarchies are adapted from the existing meshes based on the refinement flags.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

$\mathcal{L}$	n_f	n_m	n_c	n_i	n_g	ϵ	$\tilde{\epsilon}$	ω
0	2316.96	1.00	16900.00	16384.00	516.00	0.00	0.00	16642.00
1	5064.51	25.20	12075.90	10114.70	1961.21	0.92	0.77	11095.30
2	11409.70	55.41	26656.70	22288.70	4368.05	0.91	0.76	24472.70
3	8896.32	129.94	60734.80	50452.60	10282.20	0.90	0.75	55593.70
4	0.00	305.64	61288.90	45259.40	16029.60	0.79	0.58	53274.10

Table 5.10: Mean mesh hierarchy statistics for vortex flow simulation. The number of flags n_f, cells n_c, n_i, n_g, and meshes n_m as well as the clustering quality measures ϵ, $\tilde{\epsilon}$, ω are averaged over the number of timesteps at each level.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

6 Conclusion

In this thesis, two methods for generating nested mesh hierarchies for adaptive mesh refinement are presented, analyzed and implemented. Specifically, known algorithms for mesh hierarchy creation are adapted to perform within stricter constraints and are optimized to reduce the number of meshes as well as mesh cells.

The AMR method imposes certain restrictions on the mesh hierarchies, which are further extended in this thesis by external requirements stemming from performance considerations and a representative application programming interface of an exemplary level-set framework. The latter is to allow investigation into the practical usage of the developed algorithms as drop-in replacements for optimizing the nested mesh generation and adaptation process. For this reason, a novel top-down mesh generation algorithm, based on Berger-Rigoutsos signature clustering [6], is developed to create efficient clusterings both on a single level as well as complete, nesting criteria conforming multi-level hierarchies. The algorithm is compared to a bottom-up tile-based mesh generation algorithm, adapted from Luitjens tile clustering [25], to fit the extended nesting criteria.

These new mesh generation algorithms as well as the core AMR algorithm are implemented as a C++ library with a clearly defined application programming interface, allowing integration into existing simulation frameworks. The novel top-down mesh generation implementation supports shared-memory parallelization, achieving considerable speedups. Implementation examples, benchmarks and code samples are presented and discussed.

The performance of the novel top-down mesh generation algorithm is, same as that of the underlying algorithms, highly dependent on the problem geometry. Still, the performance is promising based on the analyzed examples and in some cases provides a significant improvement over previous methods. In most cases, the novel top-down algorithm yields superior results. On a single level, the total number of cells is reduced by up to 10% compared to the original Berger-Rigoutsos algorithm. The bottom-up tile-based algorithm performs better if uniformity and regularity of the mesh hierarchy is important.
In general, it is hard to judge the quality of a nested mesh hierarchy since real-world performance of an AMR procedure strongly depends on the chosen integration algorithm and other factors. Further investigations are desirable into what constitutes a high-quality nested mesh hierarchy with respect to optimizing AMR simulation time.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A Code Samples

A.1 Mesh and Grid Interfaces

Code Sample A.1: `Nest` interface definitions for custom `Grid` and `Mesh` classes.

```
class MeshClass {
  public:
    using GridType = GridClass;  // class name of respective grid class

    MeshClass* parent();
    Patch domain() const;
    double& data(index_type i, index_type j, index_type k);
    double data(index_type i, index_type j, index_type k) const;
    double& data_old(index_type i, index_type j, index_type k);
    double data_old(index_type i, index_type j, index_type k) const;
    list<MeshClass*>& neighbors();
    list<MeshClass*>& children();
    void swap_data();
};

class GridClass {
  public:
    using MeshType = MeshClass;      // corresponding mesh class
    using MeshBaseType = MeshClass;  // base mesh class
    using MeshContainer = list<unique_ptr<MeshClass>>; // container type

    GridClass(const GridParameters& grid_parameters);
    GridClass(GridClass& coarser_grid);
    GridClass* coarser_grid() const;
    GridClass* finer_grid() const;
    Patch domain() const;
    Double3D delta() const;
    double& time();
    double& time_old();
    MeshContainer::const_iterator begin() const;
    MeshContainer::const_iterator end() const;

    // create new mesh, add to container, return pointer
    MeshClass* new_mesh(Index3D start,
                        Index3D size,
                        MeshClass* parent = nullptr,
                        MeshClass* old = nullptr);

    // erase meshes from it0 to it1
    void erase(MeshContainer::const_iterator it0,
               MeshContainer::const_iterator it1);
};
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

A.2 Vortex Flow Simulation

Code Sample A.2: Vortex Flow - Setup and Simulation.

```
Config::refinement::num_levels = 5;                  // number of levels
Config::refinement::ghost_layer_size = {1, 1, 0};   // one cell ghost layer
Config::refinement::refinement_factor = {2, 2, 1}; // z-axis is not refined
Config::refinement::min_width = {8, 8, 1};          // minimum patch size
Config::refinement::regrid_period = 2;              // w.r.t. coarse level

// create hierarchy with top-down clustering
TopDownHierarchy<Grid> hierarchy;

// define initial interface: circle with radius 0.125 around (0.5, 0.75)
auto f = [](double& out, Double3D xy) {
    out = 0.125 - std::sqrt((square(xy.x - 0.5) + square(xy.y - 0.75)));
};

constexpr double T = 2;

// define timestepping function
auto timestep = [](Mesh& mesh_,          // input mesh
                   Double3D start,       // start coordinate
                   Double3D delta,       // spatial resolution
                   double t_,            // current time
                   double k_             // temporal stepsize
                  ) -> void
{
    Solvers::timestep_upwind(
        mesh_,
        start,
        delta,
        t_,
        k_,
        +[](double x, double y, double t) {
          return 1 * square(std::sin(M_PI * x)) *
            std::sin(2 * M_PI * y);
        },
        +[](double x, double y, double t) {
            return -1 * square(std::sin(M_PI * y)) *
              std::sin(2 * M_PI * x);
        });
};

// set timestepsize
double k = 1.0 / static_cast<double>(N) * 0.9;

// advance to T
hierarchy.advance(0.0, T, k, f, timestep,
                  Solvers::flag_if_small<Mesh>);
```

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

References

[1] M. Berger and J. Oliger. 'Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations'. In: *Journal of Computational Physics* 53.3 (1984), pp. 484–512. DOI: 10.1016/0021-9991(84)90073-1.

[2] W. Hackbusch. 'Local Defect Correction Method and Domain Decomposition Techniques'. In: *Defect Correction Methods*. Vol. 5. Vienna: Springer Vienna, 1984, pp. 89–113. ISBN: 978-3-7091-7023-6. DOI: 10.1007/978-3-7091-7023-6_6.

[3] M. Berger. 'Data Structures for Adaptive Grid Generation'. In: *SIAM Journal on Scientific and Statistical Computing* 7.3 (1986), pp. 904–916. DOI: 10.1137/0907061.

[4] C. Levcopoulos. 'Fast Heuristics for Minimum Length Rectangular Partitions of Polygons'. In: *Proceedings of the Second Annual Symposium on Computational Geometry*. ACM Press, 1986, pp. 100–108. ISBN: 978-0-89791-194-8. DOI: 10.1145/10515.10526.

[5] M. Berger and P. Colella. 'Local Adaptive Mesh Refinement for Shock Hydrodynamics'. In: *Journal of Computational Physics* 82.1 (1989), pp. 64–84. DOI: 10.1016/0021-9991(89)90035-1.

[6] M. Berger and I. Rigoutsos. 'An Algorithm for Point Clustering and Grid Generation'. In: *IEEE Transactions on Systems, Man, and Cybernetics* 21.5 (1991), pp. 1278–1286. DOI: 10.1109/21.120081.

[7] K. G. Powell, P. L. Roe and J. Quirk. 'Adaptive-Mesh Algorithms for Computational Fluid Dynamics'. In: *Algorithmic Trends in Computational Fluid Dynamics*. Springer New York, 1993, pp. 303–337. ISBN: 978-1-4612-7638-8. DOI: 10.1007/978-1-4612-2708-3_18.

[8] J. Bell et al. 'Three-Dimensional Adaptive Mesh Refinement for Hyperbolic Conservation Laws'. In: *SIAM Journal on Scientific Computing* 15.1 (1994), pp. 127–138. DOI: 10.1137/0915008.

[9] W. J. Coirier and K. G. Powell. 'Solution-Adaptive Cartesian Cell Approach for Viscous and Inviscid Flows'. In: *AIAA Journal* 34.5 (1996), pp. 938–945. DOI: 10.2514/3.13171.

[10] R. J. LeVeque. 'High-Resolution Conservative Algorithms for Advection in Incompressible Flow'. In: *SIAM Journal on Numerical Analysis* 33.2 (1996), pp. 627–665. DOI: 10.1137/0733033.

[11] M. J. Berger and R. J. LeVeque. 'Adaptive Mesh Refinement Using Wave-Propagation Algorithms for Hyperbolic Systems'. In: *SIAM Journal on Numerical Analysis* 35.6 (1998), pp. 2298–2316. DOI: 10.1137/S0036142997315974.

[12] J. A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. 2nd ed. Cambridge Monographs on Applied and Computational Mathematics 3. Cambridge University Press, 1999. 378 pp. ISBN: 978-0-521-64204-0.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[13] U. Trottenberg, C. W. Oosterlee and A. Schüller. *Multigrid*. Academic Press, 2001. 631 pp. ISBN: 978-0-12-701070-0.

[14] M. R. Dorr, F. Garaizar and J. A. Hittinger. 'Simulation of Laser Plasma Filamentation Using Adaptive Mesh Refinement'. In: *Journal of Computational Physics* 177.2 (2002), pp. 233–263. DOI: 10.1006/jcph.2001.6985.

[15] D. Enright et al. 'A Hybrid Particle Level Set Method for Improved Interface Capturing'. In: *Journal of Computational Physics* 183.1 (2002), pp. 83–116. DOI: 10.1006/jcph.2002.7166.

[16] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge: Cambridge University Press, 2002. ISBN: 978-0-511-79125-3. DOI: 10.1017/CBO9780511791253.

[17] S. Li and L. R. Petzold. 'Solution Adapted Mesh Refinement and Sensitivity Analysis for Parabolic Partial Differential Equation Systems'. In: *Large-Scale PDE-Constrained Optimization*. Vol. 30. Springer Berlin Heidelberg, 2003, pp. 117–132. ISBN: 978-3-540-05045-2. DOI: 10.1007/978-3-642-55508-4_7.

[18] R. Deiterding. 'Construction and Application of an AMR Algorithm for Distributed Memory Computers'. In: *Adaptive Mesh Refinement - Theory and Applications*. Vol. 41. Springer-Verlag, 2005, pp. 361–372. ISBN: 978-3-540-21147-1. DOI: 10.1007/3-540-27039-6_26.

[19] L. F. Diachin et al. 'Parallel Adaptive Mesh Refinement'. In: *Parallel Processing for Scientific Computing*. Society for Industrial and Applied Mathematics, 2006, pp. 143–162. ISBN: 978-0-89871-619-1. DOI: 10.1137/1.9780898718133.ch8.

[20] B. T. Gunney, A. M. Wissink and D. A. Hysom. 'Parallel Clustering Algorithms for Structured AMR'. In: *Journal of Parallel and Distributed Computing* 66.11 (2006), pp. 1419–1430. DOI: 10.1016/j.jpdc.2006.03.011.

[21] Y. Huang et al. 'Fast Search for Best Representations in Multitree Dictionaries'. In: *IEEE Transactions on Image Processing* 15.7 (2006), pp. 1779–1793. DOI: 10.1109/TIP.2006.873465.

[22] S. Porschen. 'On Rectangular Covering Problems'. In: *International Journal of Computational Geometry & Applications* 19.4 (2009), pp. 325–340. DOI: 10/cgw4rn.

[23] W. W. Dai. 'Issues in Adaptive Mesh Refinement'. In: *IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum*. Atlanta, GA, USA: IEEE, 2010, pp. 1–8. ISBN: 978-1-4244-6533-0. DOI: 10.1109/IPDPSW.2010.5470758.

[24] R. J. LeVeque, D. L. George and M. J. Berger. 'Tsunami Modelling with Adaptively Refined Finite Volume Methods'. In: *Acta Numerica* 20 (2011), pp. 211–289. DOI: 10.1017/S0962492911000043.

[25] J. Luitjens and M. Berzins. 'Scalable Parallel Regridding Algorithms for Block-Structured Adaptive Mesh Refinement'. In: *Concurrency and Computation: Practice and Experience* 23.13 (2011), pp. 1522–1537. DOI: 10.1002/cpe.1719.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[26] K. Museth. ‘VDB: High-Resolution Sparse Volumes with Dynamic Topology’. In: *ACM Transactions on Graphics* 32.3 (2013), pp. 1–22. DOI: 10.1145/2487228.2487235.

[27] S. A. Northrup. ‘A Parallel Implicit Adaptive Mesh Refinement Algorithm for Predicting Unsteady Fully-Compressible Reactive Flows’. PhD thesis. University of Toronto, 2014. 191 pp.

[28] F. Golay et al. ‘Block-Based Adaptive Mesh Refinement Scheme Using Numerical Density of Entropy Production for Three-Dimensional Two-Fluid Flows’. In: *International Journal of Computational Fluid Dynamics* 29.1 (2015), pp. 67–81. DOI: 10.1080/10618562.2015.1012161.

[29] S. L. Cornford et al. ‘Adaptive Mesh Refinement Versus Subgrid Friction Interpolation in Simulations of Antarctic Ice Dynamics’. In: *Annals of Glaciology* 57.73 (2016), pp. 1–9. DOI: 10.1017/aog.2016.13.

[30] B. T. Gunney and R. W. Anderson. ‘Advances in Patch-Based Adaptive Mesh Refinement Scalability’. In: *Journal of Parallel and Distributed Computing* 89 (2016), pp. 65–84. DOI: 10.1016/j.jpdc.2015.11.005.

[31] F. Löffler et al. ‘A New Parallelization Scheme for Adaptive Mesh Refinement’. In: *Journal of Computational Science* 16 (2016), pp. 79–88. DOI: 10.1016/j.jocs.2016.05.003.

[32] K. T. Mandli et al. ‘Clawpack: Building an Open Source Ecosystem for Solving Hyperbolic PDEs’. In: *PeerJ Computer Science* 2 (2016), e68. DOI: 10.7717/peerj-cs.68.

[33] A. Talpaert. ‘Direct Numerical Simulation of Bubbles with Adaptive Mesh Refinement with Distributed Algorithms’. PhD thesis. Université Paris-Saclay, 2017. 209 pp.

[34] F. Gibou, R. Fedkiw and S. Osher. ‘A Review of Level-Set Methods and Some Recent Applications’. In: *Journal of Computational Physics* 353 (2018), pp. 82–109. DOI: 10.1016/j.jcp.2017.10.006.

[35] J. E. Goodman, J. O’Rourke and C. D. Toth. *Handbook of Discrete and Computational Geometry*. 3rd ed. CRC, 2018. ISBN: 978-1-351-64591-1.

[36] M. Adams et al. *Chombo Software Package for AMR Applications Design Document*. Technical Report. Lawrence Berkeley National Laboratory, 2019, p. 206.

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar.
The approved original version of this thesis is available in print at TU Wien Bibliothek.