Numerical Algorithms (2021) 87:1147–1176
https://doi.org/10.1007/s11075-020-01003-7

ORIGINAL PAPER

A coarsening algorithm on adaptive red-green-blue refined meshes

Stefan A. Funken[1] · **Anja Schmidt**[1]

Received: 20 January 2020 / Accepted: 17 August 2020 / Published online: 1 October 2020

Abstract
Adaptive meshing is a fundamental component of adaptive finite element methods. This includes refining and coarsening meshes locally. In this work, we are concerned with the red-green-blue refinement strategy in two dimensions and its counterpart-coarsening. In general, coarsening algorithms are mostly based on an explicitly given refinement history. In this work, we present a coarsening algorithm on adaptive red-green-blue meshes in two dimensions without explicitly knowing the refinement history. To this end, we examine the local structure of these meshes, find an easy-to-verify criterion to adaptively coarsen red-green-blue meshes, and prove that this criterion generates meshes with the desired properties. We present a MATLAB implementation built on the red-green-blue refinement routine of the `ameshref`-package (Funken and Schmidt 2018, 2019).

Keywords Coarsening · Meshes · Grids · Adaptivity · Refinement · Adaptive finite element method · RGB · Red-green-blue

Mathematics Subject Classification (2010) 65M50

1 Introduction

Adaptive meshing is a popular tool to efficiently solve partial differential equations where solutions exhibit local singularities [18]. In time-dependent problems, singularities, interfaces, and forces may move or change in time. This requires coarsening meshes locally. Otherwise, the algorithm's efficiency would decrease with time since

✉ Anja Schmidt
anja.schmidt@uni-ulm.de

Stefan A. Funken
stefan.funken@uni-ulm.de

[1] Institute for Numerical Mathematics, Helmholtzstraße 20, 89081, Ulm, Germany

degrees of freedom needed for an earlier time step are not released as the singularity or interface progresses. To this end, it is common to deploy coarsening algorithms to maintain the adaptive efficiency [2, 23]. Furthermore, coarsening routines are used in multigrid techniques where a sequence of coarse and fine meshes is needed [17, 19].

Local geometric refinement is a major part of adaptive meshing. The goal is to reduce the element size by adding further nodes to a given mesh. Several refinement strategies are known which have desired properties and are therefore well suited for adaptive meshing. An overview and a list of public code are provided by Schneiders in [24]. Local coarsening is the counterpart of local refinement and is thus also an important part of adaptive meshing. There are different approaches to coarsening. Local coarsening refers to deleting nodes from a given mesh to increase the element size. Possible approaches are based on edge collapsing [1, 19], centroidal Voronoi tessellations [26], or the refinement history [2, 7, 16, 23]. The latter approach aims to invert the refinement based on the refinement history. Desired properties such as the inscribed ball condition [8] are automatically fulfilled during coarsening. The first two approaches, in contrast, do not use the refinement history. Desired properties are thus not automatically preserved within the coarsening process.

Early works on coarsening based on the refinement history refer to the hierarchical structure of the refinement and use this information to coarsen elements to their corresponding parent element [16, 23]. Chen and Zhang proposed a new concept to identify admissible-to-coarsen nodes without explicitly knowing the hierarchical structure for the newest vertex bisection (NVB) [7]. Bartels and Schreier generalized this result to any dimension for triangulations created by bisections [2]. To the best of our knowledge, this has not been done for other refinement strategies of triangular meshes. To this end, we bridge the gap and present a new criterion to adaptively coarsen meshes generated by the red-green-blue (RGB) refinement strategy in two dimensions introduced in [4] and implemented in the `ameshref`-package [11, 12]. The only information we use to describe a mesh is the element-connectivity and the coordinates of the vertices. No information about neighbors or parent-child connections is stored. A key observation within this paper is that this minimal data structure can also be kept for coarsening, i.e., no additional information is needed to coarsen the meshes. However, as hierarchical data is non-present, the determination of nodes that can be eliminated-while preserving desired properties-is more difficult. We present an algorithm that determines those "admissible" nodes.

This paper is organized as follows. In Section 2, we introduce some notations and definitions and shortly present the red-green-blue refinement. We highlight the requirements for the data structure of an RGB refinement implementation such that the proposed RGB coarsening algorithm can be realized based on this implementation. We further present the RGB implementation in the `ameshref`-package as we build our coarsening routine on this code. In Section 3, we focus on coarsening requirements and compare the RGB refinement to the newest vertex bisection. For newest vertex bisection, a coarsening strategy is already known. Thus, we show the limitations of this approach for RGB and, in Section 4, adapt it in a way that some ideas can be carried over and the limitations motivate the RGB coarsening

algorithm presented and examined. In Section 5, we focus on the efficient implementation in MATLAB by use of vectorization and conclude with numerical experiments presented in Section 6.

2 Preliminaries

Let Ω be a polygonal domain in $\mathbb{R}^2$. An element $T \subset \mathbb{R}^2$ is a triangle including edges. We call $\mathcal{T}$ a *triangulation* of Ω if:

- $\mathcal{T}$ is a finite set of elements T with positive area $|T| > 0$,
- The union of all elements in $\mathcal{T}$ covers the closure $\overline{\Omega}$,
- For $T_i, T_j \in \mathcal{T}$ with $T_i \neq T_j$ for $i \neq j$ holds $\mathring{T}_i \cap \mathring{T}_j = \emptyset$, where $\mathring{T}$ denotes the interior of T.

We denote the set of all vertices of a triangulation $\mathcal{T}$ with $\mathcal{N}$, and the set of all edges with $\mathcal{E}$. With this, $\mathcal{N}(T) := \{v \in \mathcal{N} \mid v \in T\}$ is the set of nodes of an element $T \in \mathcal{T}$. Analogously, $\mathcal{E}(T) := \{e \in \mathcal{E} \mid e \subset \partial T\}$ is the set of edges of an element $T \in \mathcal{T}$. We index $\mathcal{T}$ and $\mathcal{N}$ with a zero when we reference to the initial triangulation $\mathcal{T}_0$ and the nodes $\mathcal{N}_0$ in the initial triangulation. We call $\mathcal{T}$ a *conforming triangulation* of Ω if additionally:

- For all T_i, T_j with $T_i \neq T_j$ for $i \neq j$ holds that $T_i \cap T_j$ is the empty set, a common node or a common edge.

The aforementioned definition prevents a triangulation from having hanging nodes. A node $v \in \mathcal{N}$ is called *hanging node* if for some element $K \in \mathcal{T}$ it satisfies $v \in \partial K \setminus \mathcal{N}(K)$. We define an *extended conforming triangulation* $(\mathcal{T}, \mathrm{ref}_{\mathcal{T}})$ where $\mathcal{T}$ is a conforming triangulation and $\mathrm{ref}_{\mathcal{T}}$ is a mapping $\mathrm{ref}_{\mathcal{T}} : \mathcal{T} \to \mathcal{E}(\mathcal{T})$ that assigns a *reference edge* to each triangle $T \in \mathcal{T}$ such that for $T \in \mathcal{T}$ holds: $\mathrm{ref}_{\mathcal{T}}(T) \in \mathcal{E}(T)$. For a triangle $T \in \mathcal{T}$ with reference edge ref_T, a *refinement* $(r(T), \mathrm{ref}_{r(T)})$ is a finite set of triangles such that:

- For all $\tilde{T} \in r(T)$ holds $\tilde{T} \subset T$,
- $\bigcup_{\tilde{T} \in r(T)} \tilde{T} = T$,
- For all $\tilde{T}, \hat{T} \in r(T)$ with $\tilde{T} \neq \hat{T}$ holds that $\tilde{T} \cup \hat{T}$ is the empty set, a common node or a common edge, and
- For all $\tilde{T} \in r(T)$ a new reference edge $\mathrm{ref}_{r(T)} : r(T) \to \mathcal{E}(r(T))$ is assigned such that for $\tilde{T} \in r(T)$ holds $\mathrm{ref}_{r(T)}(\tilde{T}) \in \mathcal{E}(\tilde{T})$.

We call $(\tilde{\mathcal{T}}, \mathrm{ref}_{\tilde{\mathcal{T}}})$ a *refinement of a triangulation* $\mathcal{T}$ if:

- Each $(T, \mathrm{ref}_T) \in (\mathcal{T}, \mathrm{ref}_{\mathcal{T}})$ is refined to $(r(T), \mathrm{ref}_{r(T)})$, and
- The resulting triangulation $(\tilde{\mathcal{T}}, \mathrm{ref}_{\tilde{\mathcal{T}}})$ is an extended conforming triangulation.

The last point in particular ensures that the resulting triangulation does not have any hanging nodes. Eliminating hanging nodes by refining further elements is called CLOSURE. For further details, we refer to Sections 2.2 and [12].

In this work, we are concerned with the red-green-blue refinement in two dimensions.

Definition 1 (red-green-blue refinement (RGB), cf. [4]) We call a refinement $(r(T), \text{ref}_{r(T)})$ of a triangle T with reference edge $\text{ref}(T)$ a

- *red* refinement if triangle T is divided into four subtriangles by joining the midpoints of its edges;
- *green* refinement if triangle T is divided into two subtriangles by joining the midpoint of the reference edge $r(T)$ to the vertex opposite to this edge;
- *blue* refinement if triangle T is divided into three subtriangles by joining the midpoint of the reference edge $r(T)$ to the vertex opposite to this edge and to the midpoint of one of the other edges;

and for each subtriangle a new reference edge is assigned according to Fig. 1.

Reference edges are chosen such that during the refinement process all formed triangles starting from an initial triangle T_0 fall into at most four similarity classes [22, 25]. This ensures that degeneracies are avoided and Ciarlet's inscribed ball condition [8] is satisfied for a family of triangulations $\mathcal{T}_h$ formed by the refinements. This property is often referred to as *shape regularity* of a triangulation. The assignment of reference edges is clearly prescribed through the refinement process. There still remains the question of how to select the reference edges in the initial triangulation $\mathcal{T}_0$. Obviously, the choice has some impact on the locality of the adaptive mesh. An intuitive choice is, e.g., the longest edge. Further possibilities are discussed in [4, 27]. In depictions, we refrain from labeling the reference edges whenever it is irrelevant for the context.

2.1 Requirements for the data structure of an RGB refinement implementation

In this work, the implementation of the proposed coarsening algorithm is built on the RGB refinement implementation in the `ameshref`-package [11, 12]. This is why we focus on this concrete data structure. However, the proposed coarsening algorithm can also be based on other RGB refinement implementations without explicit refinement history. For this to work, the following must be ensured:

R1. Reference edges must be incorporated in the data structure.

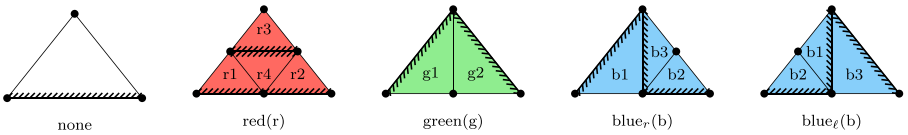


Fig. 1 From left to right: Initial triangle (none) and its possible refinements red, green, blue$_r$, and blue$_\ell$. Reference edges are highlighted by hatched lines. The letters r, g, and b denote the type of refinement and the numbers indicate the storage sequence of the newly created elements

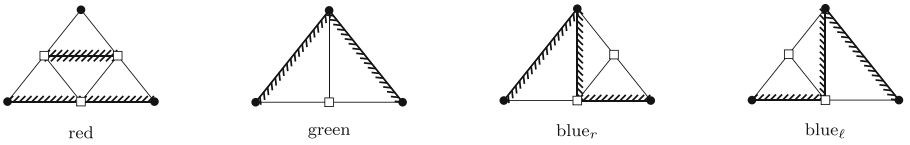


Fig. 2 Newest vertices per element (white squares) displayed for each refinement pattern. The nodes that were last added in an element are the newest vertices. The set of newest nodes does not include any nodes from the initial triangulation

R2. The data structure needs to be designed such that newest vertices, cf. Fig. 2, can easily be determined.
R3. Elements have to be numbered in a way that a blue refinement leads to the same numbering as an application of two green refinements.
R4. The middle element of red refinement patterns, cf. Fig. 3, must be identifiable.
R5. Child elements are to be stored consecutively at the former position of the parent element, cf. Figs. 1 and 4.

Let us examine each of these listed points in more detail. **R1.** Reference edges play a crucial role in RGB refinement and are thus also important for coarsening. We know how the reference edges are chosen during the refinement process. Thus, this is one key information in joining elements back together and determining the reference edge of the parent element, cf. the pattern *none* in Fig. 1. However, this information on its own is not sufficient. **R2.** In each refinement step, new nodes are added. To this end, the newest nodes are the nodes that are first removed in a coarsening step. It is therefore important information as it provides the node candidates for removal, cf. Fig. 2. These node candidates do not give any information about the patterns that lie around these nodes. However, they are important when it comes to joining elements back together. Our algorithm distinguishes between a red and a green pattern. **R3.** As a blue pattern is created by a green refinement of a green-refined element, we can deal with a blue pattern via a two-step removal of green patterns. For this to work, it is required that the numbering of a blue pattern leads to the same numbering as the application of two green refinements. Thus, it only remains to distinguish red and green patterns from each other. Where a green refinement is the result of a bisection of the element, a red refinement creates four subtriangles by joining its midpoints together. **R4.** This means that there is one triangle with new nodes only, that we call

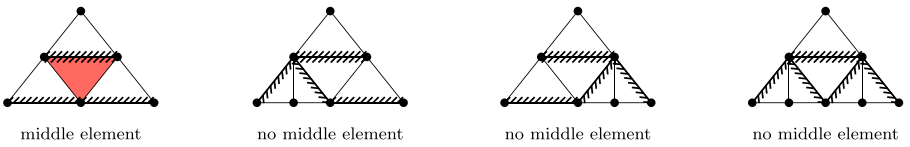


Fig. 3 Red middle element is painted in color. Middle elements whose neighbors have a different refinement level are not considered in the set of red middle elements. To determine a red middle element, for each element all three neighbors that share an edge with this element are determined and the location of the reference edge is compared. If it matches the refinement pattern on the left, it is a red middle element. If the surrounding leads to other combinations as shown, e. g., in the other three patterns, it is not a red middle element

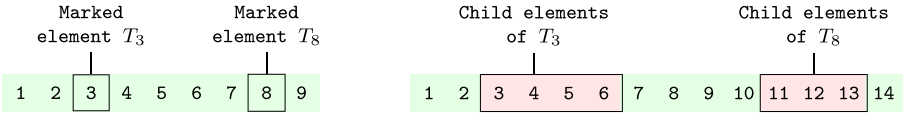


Fig. 4 Numbering of elements before (left) and after refinement (right). Subtriangles of an element are stored at the previous position of this element and successive positions, rather than appending the new elements at the end of the array. The position of other elements is then shifted by the number of newly created elements

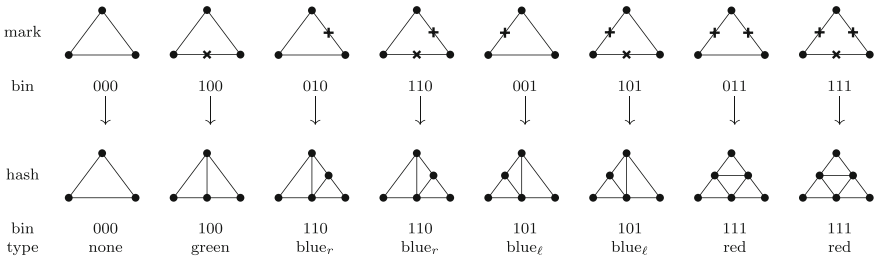
a red middle element, cf. Fig. 3. Such a middle element does not exist for green or blue patterns and therefore distinguishes red patterns uniquely from green or blue patterns. **R5.** To determine the color of the pattern, we make use of the property that subelements of one and the same element are stored consecutively, cf. Fig. 1. This ensures that elements that need to be joined together when coarsening are implicitly given in the data structure. So far, we have come to a point where we determined node candidates for removal and know their surrounding patterns. With this information, elements can be coarsened once. To join elements together in a subsequent coarsening step, it is necessary that the local information of former joined elements can still be reconstructed. We can ensure this by storing the subtriangles of a refined element at the previous position of the parent element and successive positions, rather than appending the new elements at the end of an array. The position of other triangles is then shifted by the number of newly created elements, cf. Fig. 4. If we reverse this operation when coarsening, we make sure that elements with the same parent element are stored consecutively after a coarsening step. Therefore, our coarsening routine is able to coarsen back to the initial triangulation and no explicit history tree is needed to invert the refinement.

In the following subsection, we introduce our MATLAB implementation of the RGB refinement. The fact that the requirements **R1.**, **R3.**, and **R5.** are met is made clear by the following explanation of the data structure used as well as the RGB refinement and the corresponding storage of new coordinates and elements. How to meet the requirements **R2.** and **R4.** is discussed in Section 5.

2.2 Our MATLAB implementation of RGB refinement

In this section, we give some insights into the implementation of the RGB refinement in the `ameshref`-package [11–13]. We focus on the parts that are essential for our coarsening routine. For a more thorough explanation, we refer to [12, 13]. We represent a triangulation $\mathcal{T} = \{T_1, \ldots, T_M\}$ with nodes $\mathcal{N} = \{v_1, \ldots, v_N\}$ as follows: The x- and y-coordinates of the nodes $\mathcal{N}$ are stored within an $N \times 2$ array `coordinates`. Furthermore, we represent the element-connectivity within an $M \times 3$ array `elements` where one row stores the indices of the element's three vertices $v_i, v_j, v_k \in \mathcal{N}$ with $i, j, k \in \{1, \ldots, N\}$. Optionally, boundary edges can be stored in an additional array with indices of the edge's two vertices. As depicted in Fig. 1, reference edges play a crucial role. Instead of storing this information in an additional data structure, we capture the reference edge implicitly as the edge

Table 1 Mapping of eight possible markings to the five patterns allowed in RGB refinement. For each hash, a binary number is given



between the first two vertices of an element indexed by the first two entries in the array `elements` (**R1.**). Elements are numbered counterclockwise.

In adaptive procedures, a set of marked elements $\mathcal{M}$ is given. We flag elements $T \in \mathcal{M}$ by marking each edge of the element for bisection. Obviously, neighboring elements $T \notin \mathcal{M}$ are affected indirectly by this marking. A CLOSURE step is performed to avoid creating hanging nodes. As mentioned, the assignment of reference edges ensures the shape regularity of the triangulation. To this end, the reference edge needs to be bisected before any other edge of this element is refined. For this reason, we mark edges according to the hash map shown in Table 1 and loop through this CLOSURE step until no further markers are added. Then, we refine the elements according to Fig. 1 (**R3.**) and save the new elements at the corresponding position in the array as depicted exemplarily in Fig. 4 (**R5.**). This is essential for coarsening of more than just one layer as the recursive information is implicitly given in the array `elements`. Without this, any hierarchical information is lost; and thus, coarsening of more than one layer is impossible. As a direct consequence of this way of storing the refined elements, adjacent blue and green patterns can no longer be distinguished from each other, cf. Fig. 5. It thus makes sense to consider inverting green and red refinement only.

Figure 6 serves to illustrate the data structure used in the `ameshref`-package as well as the RGB refinement and the corresponding storage of the new coordinates and elements.

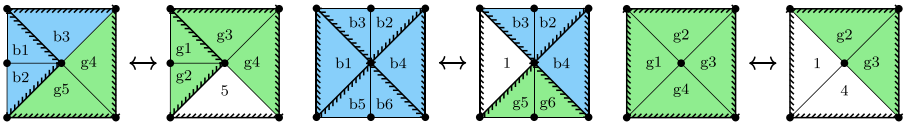
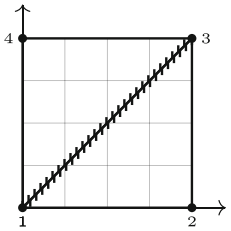


Fig. 5 Three examples to show that adjacent blue(b) and green(g) patterns cannot be distinguished on the basis of numbering: [Actual connection of elements] ↔ [Another conceivable connection based on this numbering]

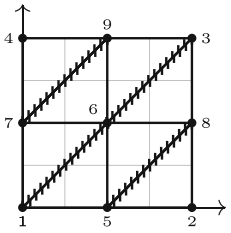
(**a**) Initial triangulation with reference edges displayed as hatched lines.



	coordinates	
1	0	0
2	2	0
3	2	2
4	0	2

	elements		
1	1	3	4
2	3	1	2

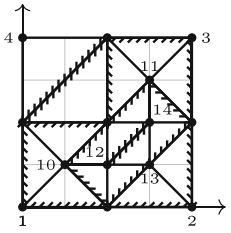
(**b**) Refined mesh obtained by marking both elements in the initial triangulation and performing an RGB refinement.



	coordinates	
1	0	0
2	2	0
3	2	2
4	0	2
5	1	0
6	1	1
7	0	1
8	2	1
9	1	2

	elements		
1	1	6	7
2	6	3	9
3	7	9	4
4	9	7	6
5	3	6	8
6	6	1	5
7	8	5	2
8	5	8	6

(**c**) Adaptive triangulation obtained by RGB refining the mesh in (B) for the marked element number 8.



	coordinates	
1	0	0
2	2	0
3	2	2
4	0	2
5	1	0
6	1	1
7	0	1
8	2	1
9	1	2
10	0.5	0.5
11	1.5	1.5
12	1	0.5
13	1.5	0.5
14	1.5	1

	elements		
1	7	1	10
2	6	7	10
3	9	6	11
4	3	9	11
5	7	9	4
6	9	7	6
7	8	3	11
8	11	6	14
9	8	11	14
10	10	5	12
11	6	10	12
12	1	5	10
13	2	8	13
14	5	2	13
15	5	13	12
16	13	8	14
17	12	14	6
18	14	12	13

Fig. 6 **a** Initial triangulation with reference edges displayed as hatched lines. The array `coordinates` lists the x- and y-coordinates of the nodes; the corresponding indices are labeled in the meshes. The array `elements` specifies the element-connectivities by indexing the corresponding coordinates. The edge between the first two nodes in an element corresponds to the reference edge. **b** Refined mesh obtained by marking both elements in the initial triangulation and performing an RGB refinement. New coordinates are appended to `coordinates` whereas new elements are stored in `elements` at the previous position of the unrefined element and successive positions. The rest is shifted by the amount of new included elements. Here, element numbers 1 to 4 are the red refinement of element 1 in (A), and element numbers 5 to 8 correspond to a red refinement of element number 2 in (A). New reference edges are highlighted and stored analogously. **c** Adaptive triangulation obtained by RGB refining the mesh in (B) for the marked element number 8. This causes a CLOSURE step to eliminate arising hanging nodes as shown in Table 1. New coordinates are appended to `coordinates`, reference edges are highlighted and stored as the edge between the first two nodes of an element, newly generated elements are stored at the previous position of the parent element and the rest is shifted, e.g., the green refinement of element 1 in (B) corresponds to element numbers 1 and 2, the green refinement of element 2 in (B) corresponds to element number 3 and 4, etc.

3 Coarsening requirements: red-green-blue refinement vs. newest vertex bisection

The goal of geometric refinement is to reduce the element size by adding further nodes to a given triangulation. In other words, one wants to increase the number of degrees of freedom. Coarsening, conversely, decreases the number of degrees of freedom in a triangulation, i.e., eliminates nodes of a triangulation. However, there are still some questions remaining. Let $\tilde{\mathcal{T}}$ be a refinement of a triangulation $\mathcal{T}$ satisfying shape regularity. How to eliminate nodes:

- To receive a triangular mesh (i.e., quadrilateral elements are not part of the triangulation)?
- To receive a shape regular mesh (i.e., the inscribed ball condition is satisfied)?
- To receive a conforming triangulation (i.e., a triangulation without hanging nodes)?
- To undo/invert a refinement without knowing the refinement history explicitly?

In literature, there are different approaches on coarsening-dealing with these details in different manners. The most common approach is to use edge collapsing known from Delaunay algorithms. This does not require to know the refinement history at all and the mesh quality is assured in the process of edge collapsing. We refer to [1, 19]. Coarsening can also be done by clustering into regions via the centroidal Voronoi tesselation, cf. [26]. The new mesh is then constructed via its dual-a Delaunay triangulation. As a further coarsening algorithm, we would like to mention the concept of using the refinement history. More precisely, the history is used to invert the refinement procedure. Most works based on this approach use a hierarchical structure, i.e., store the refinement history explicitly; see, e.g., [16, 23]. To the best of our knowledge, in 2D, there is only one work on non-hierarchical coarsening for the refinement procedure *newest vertex bisection* by Chen and Zhang [7]. The newest vertex bisection (NVB) differs from RGB refinement in one pattern. Instead of a red refinement, a bisec(3)-operation is used, i.e., instead of joining midpoints of the element's edges, the element is divided into four subtriangles by joining the midpoint of the reference edge to the vertex opposite to this edge and the midpoints of the remaining edges, cf. Fig. 7. Chen and Zhang found an easy-to-verify criterion to determine whether nodes

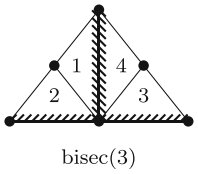


Fig. 7 Newest vertex bisection differs from RGB refinement through the use of a bisec(3)-operation instead of the red pattern. A bisec(3)-operation is essentially a green refinement of an element and each of the child elements is again green-refined

are allowed to be eliminated or not. This works well because NVB is implemented by a sequence of bisections and those can easily be undone. In other words, NVB consists of successive green refinements. The same is not true for the abovementioned red-green-blue refinement. We still see that the green pattern emerges from a bisection and the blue pattern arises from two subsequent bisections, cf. Fig. 1. However, the red pattern does not originate from a bisection of elements and thus the criterion proposed by Chen and Zhang fails to work for the RGB refinement. In this work, we discuss this issue and propose an easy-to-verify criterion to determine nodes for elimination in an RGB refined triangulation.

To this end, let us first investigate Chen and Zhang's approach to determine admissible nodes for the newest vertex bisection (in their paper called good-for-coarsening node), cf. [7]. Defining the patch:

$$\mathcal{R}_v := \{T \in \mathcal{T} \mid v \in T\},$$

the *valence* $\#\mathcal{R}_v$ counts the elements that are contiguous to a node $v \in \mathcal{N}$. Let

$$\mathcal{N}_{\text{new}} := \{v \in \mathcal{N} \backslash \mathcal{N}_0 \mid v \text{ is newest vertex of some } T \in \mathcal{T}\}$$

be the set of newest nodes in a triangulation $\mathcal{T}$. Chen and Zhang claim that the set of admissible nodes is characterized by the set:

$$\mathcal{N}_{\text{adm}} := \{v \in \mathcal{N}_{\text{new}} \ : \ \#\mathcal{R}_v = 4 \text{ or } \#\mathcal{R}_v = 2\}.$$

In Fig. 8, this idea is illustrated. The set $\mathcal{N}_{\text{adm}}$ is shown to be non-empty. This is an important requirement if one wants to assure that this criterion is useful in practical implementations. In short, a set of admissible nodes $\mathcal{N}_{\text{adm}}$ is determined with this criterion. Adaptive coarsening can then be pursued by elimination of the set of nodes $\mathcal{N}_{\text{adm}} \cap \mathcal{N}_{\text{mark}}$ where $\mathcal{N}_{\text{mark}}$ is the set of nodes that are marked through a given marking strategy. This method is powerful as it determines a set of nodes $\mathcal{N}_{\text{adm}} \cap \mathcal{N}_{\text{mark}}$ for which it is ensured that eliminating these nodes by joining elements together to its parent element does not introduce any hanging nodes.

Let us now apply this easy-to-verify criterion for the red-green-blue refinement. One can easily see that green and blue refinements carry over, i.e., green refinements are removed for a valence of two or four and blue refinements are removed in a two-step process-deleting one green refinement and then the subsequent one. This is favorable because, as already mentioned, adjacent green and blue patterns cannot be distinguished in our data structure. However, as blue patterns are not considered

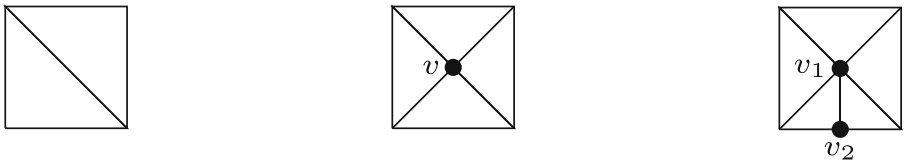


Fig. 8 Left: Initial mesh. Middle: Newest vertex v with $\#\mathcal{R}_v = 4$. This vertex can be removed. Right: Newest vertices v_1, v_2 with $\#\mathcal{R}_{v_1} = 5$ and $\#\mathcal{R}_{v_2} = 2$. Only v_2 can be removed

separately, but only as a sequence of green patterns, this does not pose any implementation problems. Applying this criterion to red patterns, it fails to detect the nodes that can be deleted, cf. Fig. 9.

For this reason, a new criterion needs to be developed to cover red and green patterns at once. We closely follow the ideas from Chen and Zhang for NVB but incorporate the red middle element in our computations. We again consider the set of newest vertices:

$$\mathcal{N}_{\text{new}} := \{v \in \mathcal{N} \setminus \mathcal{N}_0 \mid v \text{ is } \textit{newest vertex} \text{ of some } T \in \mathcal{T}\},$$

cf. Fig. 2. Let further:

$$\mathcal{M} := \{T \in \mathcal{T} \mid T \text{ is a } \textit{red middle element}\}$$

be the set of red middle elements in our triangulation $\mathcal{T}$. Red middle elements are determined by comparing the position of reference edges of neighboring elements sharing an edge with this middle element, cf. Fig. 3. The complement:

$$M^C := \mathcal{T} \setminus \mathcal{M}$$

is then used to define an adapted patch:

$$\tilde{\mathcal{R}}_v := \left\{T \in \mathcal{T} \mid v \in \mathcal{N}(T), T \in \mathcal{M}^C\right\}$$

and with

$$\mathcal{N}_{\text{candidates}} := \left\{v \in \mathcal{N}_{\text{new}} \ : \ \#\tilde{\mathcal{R}}_v = 4 \text{ or } \#\tilde{\mathcal{R}}_v = 2\right\}$$

a set of candidates for elimination is found. In both cases of Fig. 9, the adapted valence $\#\tilde{\mathcal{R}}_{v_1} = 4$ and thus v_1 is considered a candidate for elimination. The valences for other vertices stay the same with this adapted definition of a patch. Let us remark that checking only one node for a red pattern is inadequate. During a red refinement, three new nodes are created at once. To this end, it does not suffice to look at only one node, in contrast to NVB. Moreover, checking all three nodes of a red pattern is not enough either because the neighboring pattern may be a red pattern and thus additional two nodes need to be taken into account. In the process of eliminating nodes, a whole chain of red patterns has to be followed to determine which nodes can actually be eliminated without creating a hanging node. To determine the set of admissible nodes $\mathcal{N}_{\text{adm}}$ (here we consider $\mathcal{N}_{\text{adm}}$ to be the set of nodes that does not create hanging nodes when eliminated) we have to follow this chain of red patterns

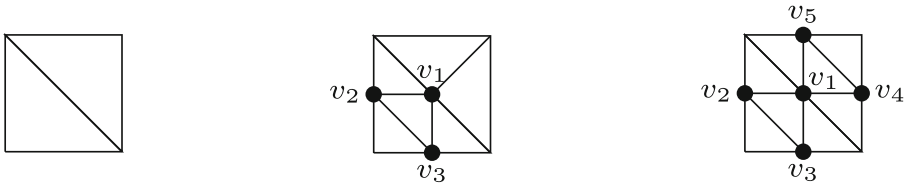


Fig. 9 Left: Initial mesh. Middle: Newest vertices v_1, v_2, v_3 with $\#\mathcal{R}_{v_1} = 5$ and $\#\mathcal{R}_{v_2} = \#\mathcal{R}_{v_3} = 3$. Right: Newest vertices $v_i, i = 1, \ldots, 5$ with $\#\mathcal{R}_{v_1} = 6$ and $\#\mathcal{R}_{v_i} = 3$ for $i = 2, \ldots, 5$. Although all nodes in both pictures could be removed, the NVB criteria cannot cover these cases, because this criterion demands $\#\mathcal{R}_v$ to be 2 or 4

and if for all nodes v laying along this chain holds $v \in \mathcal{N}_{\text{candidates}}$, it follows that $v \in \mathcal{N}_{\text{adm}}$. An easy example shows that the so determined set $\mathcal{N}_{\text{adm}}$ may possibly be empty. Consider the triangulation $(\mathcal{T}, \text{ref}_{\mathcal{T}})$ shown in Fig. 10. Here, $\mathcal{N}_{\text{adm}} = \emptyset$ since the vertex v with weight $\#\tilde{\mathcal{R}}_v = 5$ blocks all vertices along the chain from deletion. These vertices are connected through red middle elements along the loop.

Due to this property, this method is not suitable for practical purposes as we may end up in a case where the mesh is not coarsened at all. In addition, we have not even considered adaptivity here. In contrast to Chen and Zhang's method, we cannot use the set of nodes $\mathcal{N}_{\text{adm}} \cap \mathcal{N}_{\text{mark}}$ for adaptive deletion. In our case, we need to include $\mathcal{N}_{\text{mark}}$ within the determination of admissible nodes $\mathcal{N}_{\text{adm}}$ since we considered a whole chain of red patterns to avoid the creation of hanging nodes. If a node in this chain is not marked for deletion, it causes the same blockage as a node with valence unequal to 2 or 4. In order to design a practically useful algorithm, we drop the requirement to avoid hanging nodes and rather work with a CLOSURE step.

Remark 1 We see that even though NVB and RGB refinement only differ by one pattern and both refinement methods are easily implemented, finding a coarsening strategy for RGB is more involved without explicit knowledge of the refinement history. This is due to the red pattern and the resulting loss of a binary structure of the refinement history. In numerical experiments for refinement, no differences were found that would place one method above the other. However, the loss of a binary structure has more consequences, for example in the analysis of adaptive finite element methods. The analysis of convergence rates rely on a mesh overlay property; see [3, 6, 27] for the first contributions and [5] for an axiomatic contribution with a historical overview. This mesh overlay property is automatically fulfilled for binary tree refinement structures but does not hold for the RGB refinement as shown in [20].

4 The RGB coarsening algorithm

In this section, we present our RGB coarsening algorithm. We use the ideas presented in Section 3 but loosen the conditions to the set $\mathcal{N}_{\text{adm}}$. In the previous section, we considered $\mathcal{N}_{\text{adm}}$ to be the set of nodes that does not create hanging nodes when eliminated. To this end, it was necessary to look at the chain of red patterns. Now, $\mathcal{N}_{\text{adm}}$ declares the set of candidates for removal that are marked, cf. $\mathcal{N}_{\text{candidates}}$ in Section 3 with additional marking parameter, i.e., the approach is local. The main goal is to

Fig. 10 Left: Exemplary triangulation $(\mathcal{T}, \text{ref}_{\mathcal{T}})$. The vertex v with $\#\tilde{\mathcal{R}}_v = 5$ (in white) blocks all vertices along the loop (dotted) from deletion. These vertices are connected through red middle elements. Right: Initial triangulation $(\mathcal{T}_0, \text{ref}_{\mathcal{T}_0})$ of the left mesh

determine this set efficiently with a non-hierarchical data structure. As soon as the pattern is determined, deleting the pattern is an easy task. This may introduce some hanging nodes. A subsequent CLOSURE step eliminates hanging nodes to obtain a conforming triangulation. This is a practical approach and guarantees that coarsening is done locally. Algorithm 1 describes our coarsening algorithm with an additional CLOSURE step. Figure 11 illustrates this procedure.

Algorithm 1 Coarsen a conforming RGB refined triangulation $(\mathcal{T}, \mathrm{ref}_{\mathcal{T}})$ of $(\mathcal{T}_0, \mathrm{ref}_{\mathcal{T}_0})$ locally at the nodes in $\mathcal{N}_{\mathrm{mark}}$.

procedure COARSEN($\mathcal{T}, \mathrm{ref}_{\mathcal{T}}, \mathcal{N}_{\mathrm{mark}}$)
  $\mathcal{N}_{\mathrm{new}} \leftarrow \{v \in \mathcal{N} \backslash \mathcal{N}_0 \mid v \text{ is } \textit{newest vertex} \text{ of some } T \in \mathcal{T}\}$ ▷ see Fig. 2
  $\mathcal{M} \leftarrow \{T \in \mathcal{T} \mid T \text{ is a } \textit{red middle element}\}$ ▷ see Fig. 3
  $\mathcal{M}^C \leftarrow \mathcal{T} \setminus \mathcal{M}$
  $\tilde{\mathcal{R}}_v \leftarrow \left\{T \in \mathcal{T} \mid v \in \mathcal{N}(T), T \in \mathcal{M}^C\right\}$
  $\mathcal{N}_{\mathrm{adm}} \leftarrow \left\{v \in \mathcal{N} \;\middle|\; \#\tilde{\mathcal{R}}_v \in \{2,4\} \text{ and } v \in \mathcal{N}_{\mathrm{new}} \text{ and } v \in \mathcal{N}_{\mathrm{mark}}\right\}$
  $\mathcal{N}_{\mathrm{block}} \leftarrow \mathcal{N} \setminus \mathcal{N}_{\mathrm{adm}}$
  while $\mathcal{N}_{\mathrm{block}}$ changes **do** ▷ CLOSURE
    $\mathcal{M}_{\mathrm{block}} \leftarrow \{T \in \mathcal{M} \mid v \in \mathcal{N}_{\mathrm{block}} \cap \mathcal{N}(T)\}$
    $\mathcal{N}_{\mathrm{block}} \leftarrow \mathcal{N}_{\mathrm{block}} \cup \{v \in \mathcal{N}(\mathcal{M}_{\mathrm{block}}) \mid v \text{ is opposite to } \mathrm{ref}_{\mathcal{T}}(M), M \in \mathcal{M}_{\mathrm{block}}\}$
  end while
  $\mathcal{N}_{\mathrm{hang}} \leftarrow \mathcal{N}_{\mathrm{block}} \cap \mathcal{N}_{\mathrm{new}}$
  $\mathcal{N}_{\mathrm{adm}} \leftarrow \mathcal{N}_{\mathrm{adm}} \setminus \mathcal{N}_{\mathrm{hang}}$
  Create new elements and reference edges $(\hat{\mathcal{T}}, \mathrm{ref}_{\hat{\mathcal{T}}})$ according to Fig. 11.
  return $(\hat{\mathcal{T}}, \mathrm{ref}_{\hat{\mathcal{T}}})$
end procedure

Let us elaborate whether a hanging node can be created through Algorithm 1. A hanging node can be created by coarsening a pattern present at that node. Green patterns are fully removed. Red patterns may be coarsened into a green or blue pattern or they are fully removed. If the node is eliminated from the boundary, no hanging node is introduced. If this node is in the interior of the domain, it is shared by two neighboring patterns. A hanging node is introduced if the connection to this node is eliminated on one side but not on the other. This might be the case if one pattern falls into the presented cases shown in Fig. 12 and the other does not. We further argue that this cannot happen. First, all newest nodes $\mathcal{N}_{\mathrm{new}} := \{v \in \mathcal{N} \backslash \mathcal{N}_0 \mid v \text{ is } \textit{newest vertex} \text{ of some } T \in \mathcal{T}\}$ are collected. It might happen that a node v is the newest node of one pattern but not of the other. The definition still includes this node, as it is the newest node of *some* $T \in \mathcal{T}$. Figure 13 shows possible cases where this occurs. We see that for those cases holds that $\#\tilde{\mathcal{R}}_v \notin \{2,4\}$ and v is thus not considered for elimination. As a consequence, neither the one nor the other pattern is coarsened at that node and thus no hanging node is introduced. Furthermore, this can happen for a case shown in Fig. 14. But those cases cannot arise as only conforming triangulations are considered as input. There still remains the case,

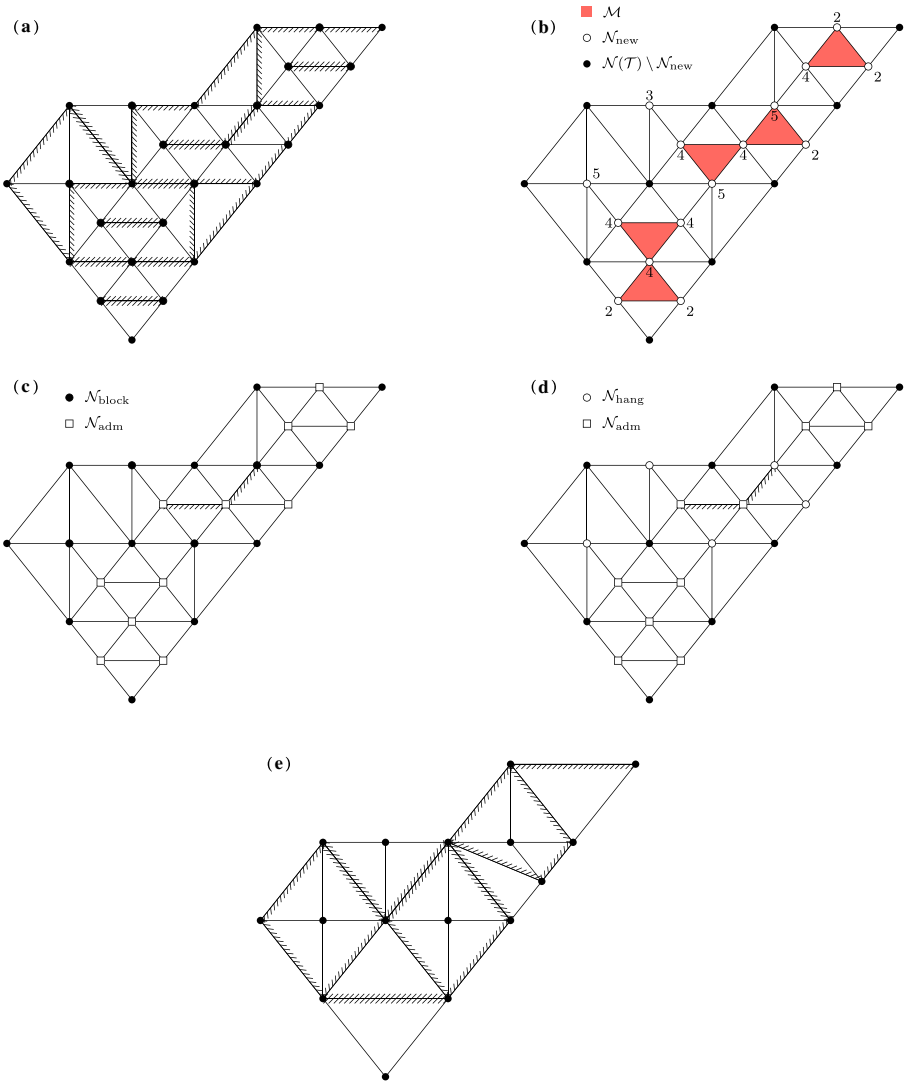


Fig. 11 Illustration of Algorithm 1: $(\hat{\mathcal{T}}, \text{ref}_{\hat{\mathcal{T}}}) = \text{COARSEN}\big(\mathcal{T}, \text{ref}_{\mathcal{T}}, \mathcal{N}(\mathcal{T})\big)$. **a** Exemplary triangulation $\mathcal{T}$ obtained by RGB refinement with reference edges $\text{ref}_{\mathcal{T}}$ shown as hatched lines. All nodes are marked for elimination, i. e. , $\mathcal{N}_{\text{mark}} = \mathcal{N}(\mathcal{T})$. **b** Illustration of newest nodes (white dots) and red middle elements (backed with color). The valence $\#\tilde{\mathcal{R}}_v$ is shown for the newest vertices $v \in \mathcal{N}_{\text{new}}$. **c** Determination of $\mathcal{N}_{adm}$ and $\mathcal{N}_{block}$ according to lines 6-7 of Algorithm 1. Reference edges are shown where it is relevant. **d** CLOSURE step in lines 8-13 of Algorithm 1. If a node of a red middle element is blocked, the node opposite to the reference edge of this middle element needs to be blocked, too. Otherwise the patterns do not follow the path of reference edges anymore. The reference edges of the corresponding red middle elements are marked. **e** Output mesh $(\hat{\mathcal{T}}, \text{ref}_{\hat{\mathcal{T}}})$

where this node is the newest node for both patterns. In the refinement process, we have red, green, and blue patterns. In the coarsening step, we only coarsen red and green patterns. To this end, we examine what happens if blue patterns are present

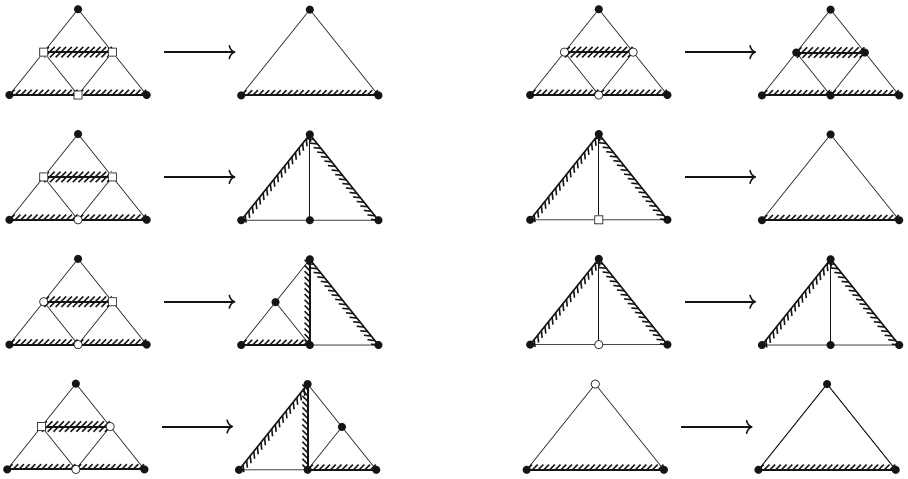


Fig. 12 Create new elements and reference edges ($\hat{\mathcal{T}}, \hat{ref}_{\hat{\mathcal{T}}}$) according to the depiction for nodes in $\mathcal{N}_{\text{adm}}$ (white squares) and $\mathcal{N}_{\text{hang}}$ (white dots). The patterns used are the same as allowed in the RGB refinement process. Properties such as the shape regularity are thus preserved

at a new node; see Fig. 15. In this case, $\#\mathcal{R}_v \notin \{2, 4\}$ thus the connection to this node is not eliminated. The set $\mathcal{N}_{\text{adm}}$ calculated first will only get smaller during the algorithm and therefore no new cases causing hanging nodes will occur. If a node is blocked, it is not eliminated by either pattern. If it is admissible, it is eliminated by both patterns. Overall, the algorithm generates a conforming triangulation. The shape regularity is preserved, as lines 8–11 ensure that the reference edge is always marked and thus an element T is coarsened into triangles that are in four similarity classes only. The inscribed ball condition is thus satisfied. This shows:

Theorem 1 (Output COARSEN) *Let* $(\mathcal{T}, ref_{\mathcal{T}})$ *be a conforming triangulation obtained by RGB refinement of an initial conforming triangulation* $(\mathcal{T}_0, ref_{\mathcal{T}_0})$ *and* $\mathcal{N}_{\text{mark}} \subset \mathcal{N}$. *Then COARSEN*$(\mathcal{T}, ref_{\mathcal{T}}, \mathcal{N}_{\text{mark}})$ *from Algorithm 1 generates a conforming and shape regular triangulation.*

Remark 2 The CLOSURE step in Algorithm 1 might introduce new connectivities in the coarsened mesh. The shape regularity of the mesh is still preserved because the new connectivities are the same as the ones allowed in the refinement process. However, in adaptive methods, the required interpolation process is more involved by creating new connectivities, especially when evaluating non-nodal values. The authors in [21] show an interpolation approach for the red-green refinement. A similar ansatz can be used in our case.

Our coarsening operation is not completely inverse to RGB refinement. A blue refinement of an element results in three child elements. One application of the coarsening algorithm does not coarsen the blue refinement to its parent element but to two

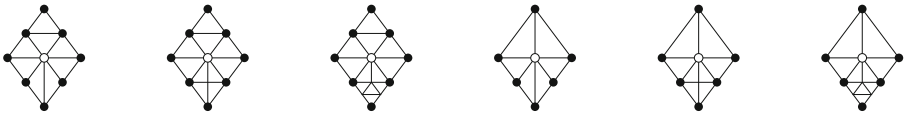


Fig. 13 Possible cases for which a node v (white dot) is the newest node of one pattern but not of the other. Here, the node v is the newest node for the upper pattern but not for the lower one. We see that for all cases $\#\mathcal{R}_v \notin \{2, 4\}$ and thus those nodes are not considered for elimination

child elements of the parent element. Additional patterns created during the CLOSURE step also do not follow an inverse operation of the RGB refinement. However, we can relate these patterns to a corresponding mesh obtained by NVB refinement. More specifically, as soon as one or two nodes of a red pattern are eliminated, our CLOSURE step does not go back to the parent element but to blue or green children of this parent element. We thus handle the mesh as if the mesh was NVB refined with this set of newly added nodes. The corresponding mesh can be defined as a bijective function and is discussed in detail in [14].

Even though the coarsening operation is not inverse, the following result applies: Algorithm 1 can fully recover the initial triangulation provided that the initial triangulation $(\mathcal{T}_0, \mathrm{ref}_{\mathcal{T}_0})$ is of weak BDD type.

Definition 2 (weak BDD-property, cf.[4, 15]) An element $T \in \mathcal{T}$ is called *isolated* if the reference edge $\mathrm{ref}_{\mathcal{T}}(T)$ is shared with another element $\tilde{T} \in \mathcal{T}$ and $\mathrm{ref}_{\mathcal{T}}(\tilde{T}) \neq \mathrm{ref}_{\mathcal{T}}(T)$. A mesh $(\mathcal{T}, \mathrm{ref}_{\mathcal{T}})$ has the *weak BDD-property* if two distinct isolated elements $T_1, T_2 \in \mathcal{T}$ do not share an edge.

Remark 3 In adaptive meshing, it is common to impose conditions on the distribution of reference edges. In fact, the NVB coarsening algorithm of Chen and Zhang relies on a stricter condition on the initial triangulation, namely that there are no isolated elements at all, cf. [7]. This is the same condition as Binev, Dahmen, and DeVore

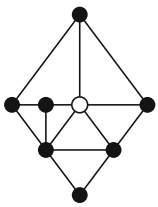


Fig. 14 Impossible situation for which the node (white dot) is the newest node for the upper pattern but not for the lower one. This situation cannot occur because only conforming triangulations are allowed as input parameters, i.e., hanging nodes cannot exist

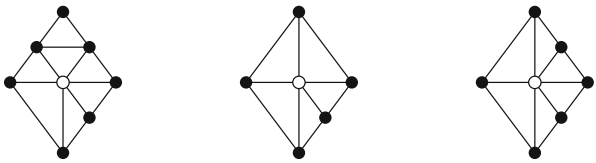


Fig. 15 Blue patterns present at a node v (white dot). In these cases, it holds that $\#\mathcal{R}_v \notin \{2, 4\}$, i.e., the node v is not considered for elimination

(BDD) imposed on the initial triangulation to prove optimal convergence rates for adaptive finite element methods with NVB [3]. Carstensen weakened this condition to the above Definition 2 to prove the H^1-stability of the L^2-projection for RGB refined meshes [4]. Later, Karkulik, Pavlicek, and Praetorius improved these results in the sense that the condition of assignment of reference edges in the initial triangulation was removed [15]. For coarsening, we still need the weakened condition to guarantee the existence of nodes for elimination. Without any assumptions on the initial mesh, a loop of isolated elements may be formed that cannot be eliminated with our coarsening criteria, cf. [7]. This is not restrictive. In fact, Carstensen provides an algorithm that generates an extended conforming triangulation of weak BDD type for an arbitrary conforming triangulation in linear complexity [4]. The results of Chen and Zhang for NVB remain also true under the weak BDD-assumption.

With the weak BDD-property, we can show:

Theorem 2 (Coarsening) *Let $(\mathcal{T}, ref_{\mathcal{T}})$ be an arbitrary RGB refinement of an initial conforming triangulation $(\mathcal{T}_0, ref_{\mathcal{T}_0})$ where $(\mathcal{T}_0, ref_{\mathcal{T}_0})$ is of weak BDD type. Let $(\mathcal{T}^{(i)})_{i=0,1,\dots}$ be a sequence of triangulations generated by Algorithm 1, i.e.,*

$$\mathcal{T}^{(0)} := \mathcal{T} \text{ and } \left(\mathcal{T}^{(i+1)}, ref_{\mathcal{T}^{(i+1)}}\right) := COARSEN\Big(\mathcal{T}^{(i)}, ref_{\mathcal{T}^{(i)}}, \mathcal{N}\big(\mathcal{T}^{(i)}\big)\Big).$$

Then, after a finite number of steps $M \in \mathbb{N}_0$, we obtain:

$$\mathcal{T}^{(M)} = \mathcal{T}_0.$$

Remark 4 In practice, we would like to ensure that $M \leq cN$ for a small $c \geq 1$, where N is the number of adaptive RGB refinement steps performed to obtain $(\mathcal{T}, \mathrm{ref}_{\mathcal{T}})$. In Section 6.2, some numerical experiments to estimate the ratio $\frac{M}{N}$ are performed.

Proof For the proof, we consider a mesh refined by NVB as the coarsening patterns are chosen as if we would trace back the refinement history of a NVB refined mesh with the same nodes. RGB can then be related to NVB via a corresponding mesh function, cf. [15], i.e., the results are also valid for RGB. For a NVB refined mesh,

Fig. 16 Compatible patches as shown in [7]. The nodes v (white squares) have the property $v \in \mathcal{N}_{\text{new}}$ and $\#\mathcal{R}_v \in \{2, 4\}$ and can thus be eliminated in a coarsening step. Reference edges are shown as hatched lines

there holds that $\tilde{\mathcal{R}}_v = \mathcal{R}_v$ for each node v and thus relates to the work of Chen and Zhang [7]. As shown in their work, only compatible patches are eliminated; see Fig. 16.

Let $(\mathcal{T}_0, \text{ref}_{\mathcal{T}_0})$ be of weak BDD type. We first show that any uniform bisec(3)-refinement of a weak BDD triangulation results into a weak BDD triangulation. For this purpose, we first recognize that elements with a reference edge as inner edge are not isolated due to the allowed refinement patterns. Thus, we only have to consider elements that share their reference edge with a triangle of another parent element. As illustrated in Fig. 17, the weak BDD property is inherited at these edges.

Therefore, we can restrict ourselves to a piece of the whole refinement and see if we can eliminate nodes to achieve the uniform refinement of a lower level. Figure 18 shows what happens for a piece of the whole refinement. For the highlighted nodes, $\#\mathcal{R}_v \in \{2, 4\}$ applies; otherwise, two isolated elements would have shared an edge in the initial triangulation, i.e., the mesh would not be of weak BDD type. Therefore, these nodes can be eliminated. In a further coarsening step, we again have a compatible patch that can be coarsened. In a last step, with the same arguments as above, $\#\mathcal{R}_v \in \{2, 4\}$ and can thus be eliminated. As long as $\mathcal{T}^{(i)} \neq \mathcal{T}_0$, this process can be repeated until there are no more newest nodes. In this way, the initial mesh is recovered after a finite number of steps. □

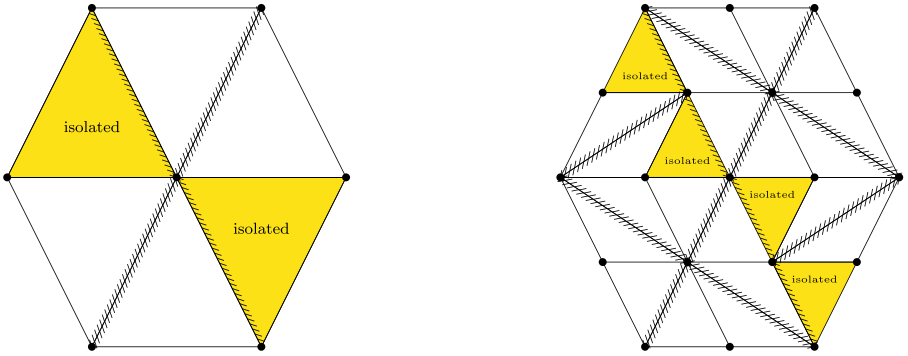


Fig. 17 The weak BDD-property is inherited if the mesh is uniformly bisec(3)- or red-refined. Isolated elements are marked; reference edges are shown as hatched lines. Left: Initial mesh of weak BDD type. Right: Uniform bisec(3)-refinement of the left mesh. The resulting mesh is still of weak BDD type. Isolated elements exist only at the reference edges previously shared with another parent element with a different reference edge. An RGB refined mesh has the same property

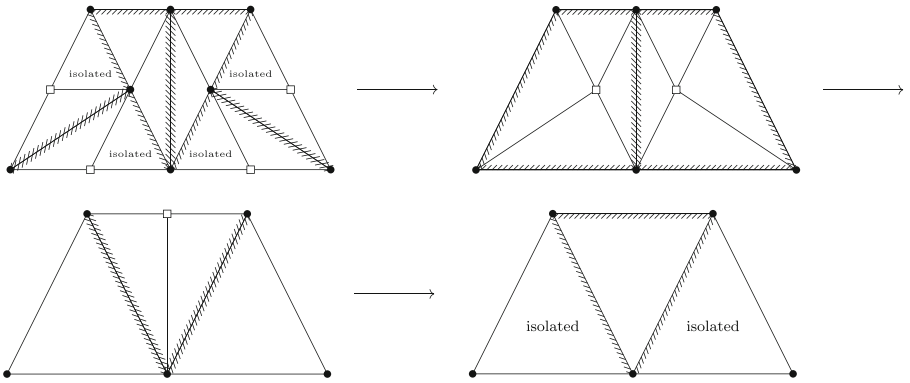


Fig. 18 Piece of a uniformly bisec(3)-refined triangulation of weak BDD type. Here, two isolated elements were connected by one element. From top left to bottom right: Uniformly refined mesh with nodes (white squares) that can be coarsened. Subsequently, coarsened meshes where the nodes marked in the previous mesh have been eliminated and further nodes (white squares) are determined for the next elimination. After three coarsening steps, a uniform refinement of a lower level is reached

Remark 5 At most $M = \#\mathcal{N}(\mathcal{T}) - \#\mathcal{N}_0$ are needed to recover the initial triangulation. Numerical experiments show that this bound is very pessimistic.

5 MATLAB implementation of the coarsening algorithm

In the previous section, we have presented our RGB coarsening algorithm. In this section, we focus on the concrete implementation in MATLAB based on the refinement routine `TrefineRGB.m`; see [11]. We have already discussed the data structure used in the refinement procedure in Section 2.2. This will also play a major role in the implementation of the coarsening routine. Let us recap quickly the main structures: Elements $T \in \mathcal{T}$ are defined by their vertices v_i, $i = 1, 2, 3$ and numbered counterclockwise. The edge in between the first two vertices v_1, v_2 is the reference edge $\mathrm{ref}_{\mathcal{T}}(T) = v_1 v_2$ of T. Refined elements are stored at the former position of the parent element and subsequent positions, cf. Fig. 4. Elements that belong together are therefore listed one after the other. RGB coarsening can then be implemented as follows (see Listing 1):

- Lines 1–4: The function `TcoarsenRGB` expects the number of coordinates N_0 of the initial triangulation $\mathcal{T}_0$, mesh information such as `coordinates` and `elements` and optionally boundary data in `varargin` as input. The last argument in `varargin` is reserved for marked elements (line 4). It is sufficient to pass the number of coordinates N_0 of the initial triangulation, since the coordinates added during the refinement process are appended to the end of the array `coordinates`. This means that the first N_0 entries in `coordinates` are the nodes $\mathcal{N}_0$.
- Lines 5–7: A triangulation is represented by the array `elements` and `coordinates`. The auxiliary functions `provideGeometricData` and

```
function [coordinates,elements,varargout] = ...
                    TcoarsenRGB(N0,coordinates,elements,varargin)
nC = size(coordinates,1); nE = size(elements,1);
marked = varargin{end};
%*** obtain geometric information
[~,element2edges] = provideGeometricData(elements,zeros(0,4));
edge2elements = createEdge2Elements_adv(element2edges);
%*** determine admissible nodes for coarsening
midelement = find(sum(abs(element2edges(4:end,:)-[element2edges(3:end-1,1),...
   element2edges(1:end-3,2),element2edges(2:end-2,3)]),2)==0)+3;
newest_nodes = zeros(nC,1); marked_nodes = zeros(nC,1);
newest_nodes(elements(:,3)) = 1; newest_nodes(1:N0)=0;
marked_nodes(elements(marked,:)) = 1;
%*** define color of nodes
mid_nodes = elements(midelement,:);
red_node = accumarray(reshape(mid_nodes,[],1),1);
valence = accumarray(elements(:),ones(3*nE,1),[nC,1])...
            -[red_node;zeros(nC-size(red_node,1),1)];
newest_marked = elements(marked,3);
green_node = setdiff(newest_marked(newest_marked>N0),find(red_node));
green_val = valence(green_node);
%*** determine arising hanging nodes
blocked_nodes = ~(((valence == 2) | (valence == 4)) & newest_nodes & marked_nodes);
old = -1;
while ~isempty(find(old ~= blocked_nodes))
    old = blocked_nodes;
    [row,~] = find(reshape(blocked_nodes(mid_nodes),[],3));
    blocked_nodes(elements(midelement(row),3)) = 1; % mark reference edge
end
hanging_nodes = blocked_nodes & newest_nodes;
%*** determine coarsening pattern regarding hanging nodes
red_elements = reshape(hanging_nodes(mid_nodes),[],3);
val = sum(red_elements .* [1,2,4],2);
none = find(val==0);
gdx = find(val == 4);
```

Listing 1 Lines 1–35 of `TcoarsenRGB.m`

`createEdge2Elements_adv` provide more geometric information on the mesh. The array `element2edges` specifies the edges of each element, cf. [12], and the array `edge2elements` specifies the elements containing this edge and the position of this edge within an element for all edges. E. g. , for

$$\texttt{element2edges} = \begin{bmatrix} 3 & 1 & 2 \\ 3 & 4 & 5 \end{bmatrix} \text{ holds } \texttt{edge2elements} = \begin{bmatrix} 1 & 2 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 1 & 1 & 2 & 1 \\ 2 & 2 & 0 & 0 \\ 2 & 3 & 0 & 0 \end{bmatrix}.$$

`edge2elements(`ℓ`,1:2)` specifies the position (column, row) of the ℓ-th edge in `element2edges`. If the $\ell-$th edge is a boundary edge, it holds `edge2elements(`ℓ`,3:4)` $= [0, 0]$. Otherwise, it specifies the position (column, row) of the ℓ-th edge's second entry in `element2edges`.

– Lines 9–10: (**R4.**) We determine all red middle elements as depicted in Fig. 3 by comparing the edges of four consecutive entries in `element2edges`. If the second edge of element 1 is equal to the second edge of element 4, the third edge of element 2 is equal to the third edge of element 4 and the first edge of element 3 is equal to the first edge of element 4, then `sum(abs(...))=0` and therefore the index of the red middle element is given by `find(sum(abs(...))=0)+3`, cf. Fig. 19. Note, that this characterizes a red pattern uniquely and thus this criterion to find red middle elements is appropriate.

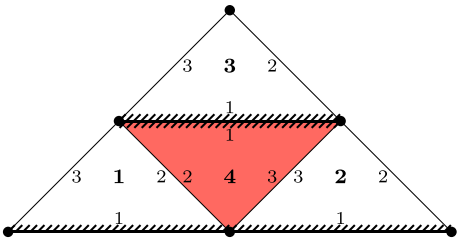


Fig. 19 Numbering of red pattern: Element numbers are in bold, edge numbers per element are shown in a smaller font size

- Lines 11–13: (**R3.**) In this part, the newest node of each element is detected. The newest node of an element is stored on position 3 in `elements`. We only consider nodes for coarsening that are not part of the initial triangulation $\mathcal{T}_0$ (line 12). Note that we make a systematic error for red patterns, as an additional node is detected as newest node even though it is not a new one, cf. Fig. 20. We have to consider this systematic error in the course of our implementation. If an element is marked for coarsening, we mark all nodes of this element for coarsening (line 13).
- Lines 15–21: Let `red_node` be the set of newest nodes of a red pattern. The term `valence` computed in line 17 is the number of elements $\#\tilde{\mathcal{R}}_v$ of the patch $\tilde{\mathcal{R}}_v$ defined in Algorithm 1. Nodes of green patterns are then all new nodes that are not red. Note that the array `green_node` includes the systematic errors.
- Lines 22–29: The admissible set $\mathcal{N}_{\text{adm}} = \left\{v \in \mathcal{N} \mid \#\tilde{\mathcal{R}}_v \in \{2,4\} \text{ and} v \in \mathcal{N}_{\text{new}} \text{ and } v \in \mathcal{N}_{\text{mark}}\right\}$ and $\mathcal{N}_{\text{block}} = \mathcal{N} \setminus \mathcal{N}_{\text{adm}}$ are computed. As the reference edge plays a crucial role, we need to do a CLOSURE step to ensure that the shape regularity still holds when coarsening. We again consider the patterns shown in Fig. 1. We make sure that at least the reference edge of the parent element is marked. Differently said, at least the third vertex of the red middle element needs to be blocked if any other vertex in this element is blocked (lines 27–28). We loop through this process until no further changes are made.
- Lines 30–37: Hanging nodes are then given by $\mathcal{N}_{\text{hang}} = \mathcal{N}_{\text{block}} \cap \mathcal{N}_{\text{new}}$ (line 30). With this, we determine the coarsening pattern regarding to these hanging nodes. For green patterns, this is either coarsen or not coarsen. For red patterns, we have more cases to consider; see Fig. 12. To this end, we form the weighted sum of hanging nodes. A red pattern can then be coarsened to the patterns: none (000), green (001), blue$_r$ (101), blue$_\ell$ (011), and red (111). In the weighted sum computed in line 33, these patterns correspond to the values 0, 4, 5, 6, and 7. The

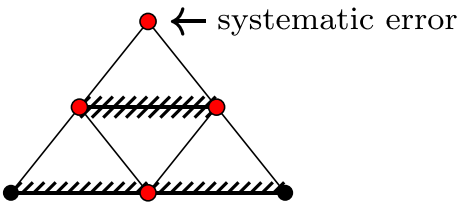


Fig. 20 Systematic error made by taking `elements(:,3)` as newest nodes

value 7 is not considered separately as in this case the elements are kept as they are and are not coarsened.

We omit the presentation of the rest of the code (further 70 lines), as it is a straightforward implementation of element updates. We distinguish between former red patterns, former green patterns neighboring a red pattern, and former green patterns not neighboring red patterns. Note, that for the latter, the corresponding array includes the systematic error made earlier. To this end, we only consider subsequent elements for coarsening, the case shown in Fig. 21 is out of question. The valence is $\#\tilde{\mathcal{R}}_v = 2$ but the elements containing v are not numbered consecutively and thus are not considered. In a next step, the old triangulation is deleted. If provided, boundary data is updated. Again, nodes are eliminated only if they are not blocked and they do not stem from the systematic error shown in Fig. 20. Lastly, surplus nodes are deleted and the new coordinates and elements are updated. The interested reader may download the full code from [10].

5.1 A minimal example

Listing 2 shows an exemplary code of how to embed the coarsening routine into a framework. We start with defining an initial mesh $\mathcal{T}_0$ (lines 1–4). A refined mesh $\tilde{\mathcal{T}}$ is created via `TrefineRGB` (lines 7–15). For a given triangulation $\mathcal{T}$ and a given discrete point set $\mathcal{P}$, the function `point2element` determines the elements of $\mathcal{T}$ that include p for some $p \in \mathcal{P}$. Thus, for the defined discrete point set in lines 16–22, elements in $\tilde{\mathcal{T}}$ are marked according to `point2element`. We coarsen the mesh via the function call `TcoarsenRGB` (lines 26–27) until no further change is made (line 28). Lines 33–35 plot the locally coarsened mesh.

5.2 Examples and demo files

The coarsening routine for RGB meshes is part of the toolbox `ameshcoars` - Efficient Implementation of Adaptive Mesh Coarsening in 2D [10]. Numerical examples and demo files based on the interplay of refinement and coarsening are provided in subdirectories of the `ameshcoars`–toolbox:

- `example1/`: refinement along a moving circle,
- `example2/`: adaptive finite element implementation following [9] for a quasi-stationary partial differential equation,

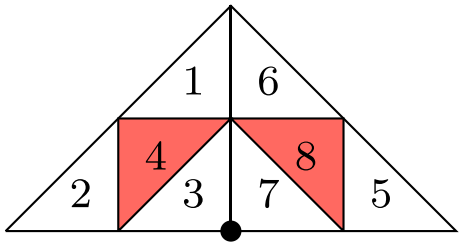


Fig. 21 Due to the systematic error, situations arise where the valence $\#\tilde{\mathcal{R}}_v = 2$ and the elements containing v are not numbered consecutively (3 and 7). Analogously, the same situation can arise for $\#\tilde{\mathcal{R}}_v = 4$

```
%% Define initial mesh
coordinates = [0,0;1,0;1,1;0,1;2,0;2,1];
elements = [3,1,2;1,3,4;2,6,3;6,2,5];
boundary = [1,2;2,5;5,6;6,3;3,4;4,1];
N0 = size(coordinates,1);
c_old = 0;
%% Refine uniformly
while 1
    marked = 1:size(elements,1);
    [coordinates,elements,boundary] ...
        = TrefineRGB(coordinates,elements,boundary,marked);
    if isempty(marked)|| (size(coordinates,1)>1e3)
        break
    end
end
% define discrete points (here a disc)
phi = -pi:pi/50:pi;
r = 0.2:1/50:0.4;
[r,phi] = meshgrid(r,phi);
s = r.*cos(phi);
t = r.*sin(phi);
points = [s(:)+1,t(:)+0.5];
%% Coarsen at discrete points
while 1
    mark3 = point2element(coordinates,elements,points);
    [coordinates,elements,boundary] = ...
                TcoarsenRGB(N0,coordinates,elements,boundary,mark3);
    if size(coordinates,1) == c_old
        break
    end
    c_old = size(coordinates,1);
end
% plot results
clf, patch('Faces',elements,'Vertices',coordinates,'Facecolor','none')
axis equal, axis off
```

Listing 2 A minimal example

– `example3/`: triangulation of a GIF,
– `example4/`: local coarsening of a uniformly refined triangulation.

6 Numerical experiments

In this section, we test our coarsening routine with MATLAB 2018a. We present some results based on `example1/` and `example4/` of the `ameshcoars`-toolbox [10]. In particular, we look at the interplay of refinement and coarsening and how well moving singularities can be captured by this procedure. Furthermore, we take a look at the efficiency of our coarsening algorithm. We know that our coarsening implementation is not inverse to the refinement but that it can fully recover the initial triangulation. To this end, we want to examine what element- and coordinate-ratios we get between each refinement/coarsening step to get a feeling of how efficient our coarsening routine is. Furthermore, we examine our implementation for scalability and give a remark on how the implementation depends on the local refinement. Lastly, we show that coarsening can be done locally.

6.1 Interplay of refinement and coarsening

Let us start with a basic example. Adaptive coarsening is widely used to release degrees of freedom that are not needed anymore as, e.g., a singularity advances. We

imitate the behavior by a moving circle, which is supposed to represent the singularity. Starting off with an initial triangulation $(\mathcal{T}_0, \mathrm{ref}_{\mathcal{T}_0})$, we refine along the circle at time t_0. To capture the singularity in $t_1 > t_0$, we could start off again from $(\mathcal{T}_0, \mathrm{ref}_{\mathcal{T}_0})$ and only use the refinement procedure. However, as the circle progresses steadily, only a few nodes need to be released and only a few nodes need to be added. To this end, we use our coarsening routine to set the corresponding coordinates free and the refinement routine to add further coordinates to capture the shape of the circle. The comparison of number of degrees of freedom between refinement only and refinement combined with coarsening is illustrated in Fig. 22.

We first explain the procedure in this example. We define a circle with center and radius. If this circle intersects an element, we mark this element for refinement. This is done by the function `markcircle` and together with the refinement routine this ensures that we capture the shape of the circle. We do this until a maximal number $N_{\max}$ of coordinates is reached, or the element size becomes too small. We then mark all elements for coarsening, coarsen and repeat this until we fall below a given minimal number $N_{\min}$ of nodes. Subsequently, in the next time step t_1, we again refine along the circle (now moved to another position!), but do not start from the initial mesh but from the coarsened mesh from time step t_0. This is done consecutively, and we get a sequence of triangulations capturing the moving circle; see left column of Fig. 23 and cf. `example1/` in [10].

Note that the triangulations we get are highly sensitive to the choice of the parameter $N_{\min}$. We observe a pollution effect if $N_{\min}$ is chosen too big; see right column of Fig. 23. However, in general, a pollution effect does not falsify the computation. It only means that the shape of the circle is not captured in the best possible way and nodes exist where they are not necessarily needed. In principal, it is not just the parameter $N_{\min}$ that is responsible for a pollution effect. It also depends on how fast the front for refinement advances, how many time steps are considered, how $N_{\max}$ is chosen, etc. However, in most applications, the pollution effect is controlled as error estimators are often used to regulate the error and thus also adapt the mesh, cf. `example2/` and `example3/` in [13].

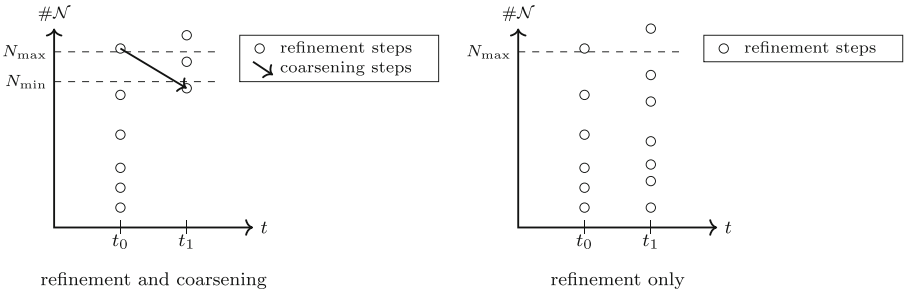


Fig. 22 Left: Interplay between refinement and coarsening to capture moving circle in time step $t_1 > t_0$. Right: Refinement only to capture moving circle in time step t_1

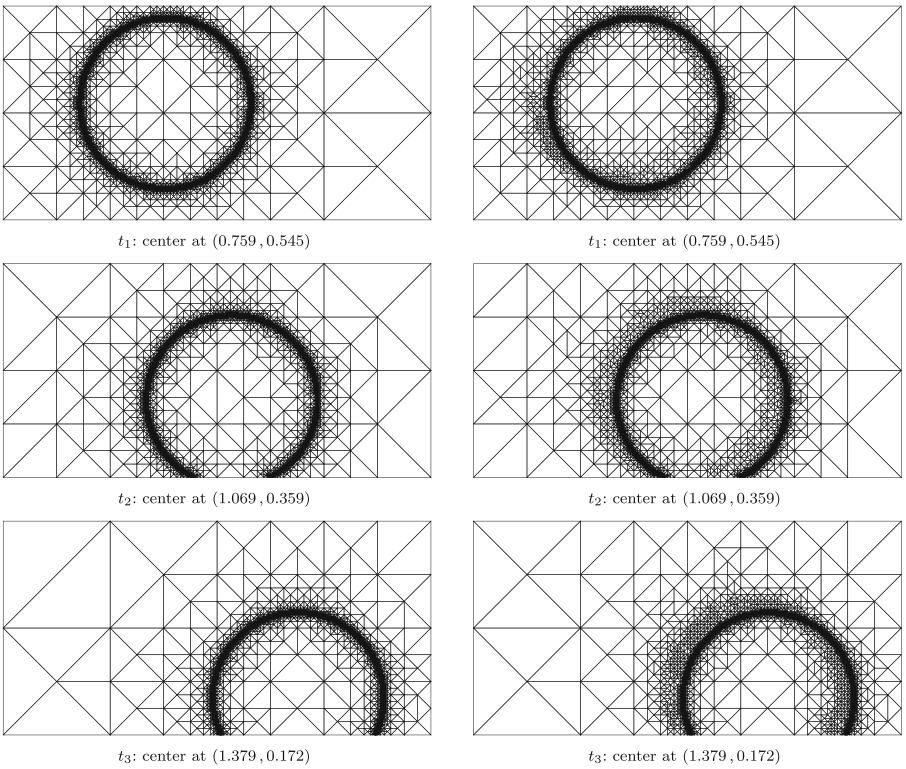


Fig. 23 Moving circle: triangulations at different time frames t_1, t_2, t_3 with $N_{\max} = 10^4$. Left: $N_{\min} = 10^2$. Right: $N_{\min} = 10^3$. On the right, we observe a pollution effect; the circle draws a tail

6.2 Efficiency of the coarsening routine

We want to determine how efficient our coarsening routine is in the sense of element- and coordinate-ratios in between two coarsening steps in comparison with two refinement steps. Let therefore $\mathcal{T}_i$ be the triangulation after i refinement steps and $\mathcal{N}_i$ denotes the set of nodes of the triangulation $\mathcal{T}_i$. We determine the element-ratio

$$\rho^i_{\text{elem}} = \frac{\#\mathcal{T}_{i+1}}{\#\mathcal{T}_i}$$

and the coordinate-ratio

$$\rho^i_{\text{coord}} = \frac{\#\mathcal{N}_{i+1}}{\#\mathcal{N}_i}$$

in each refinement step. Let N be the number refinement steps. We expect $1 \leq \rho^i_{\text{elem}} \leq 4$ for all $i = 1, \ldots, N$ due to the refinement patterns. We compute the actual ratios for an adaptive refinement along a circle. They are presented in Table 2. We see that the geometric means are $\overline{\rho}_{\text{elem}} = 2.40$ and $\overline{\rho}_{\text{coord}} = 2.15$.

Analogously, we determine the ratios for coarsening steps. To this end, let $\hat{\mathcal{T}}_j$ be the triangulation received after j coarsening steps, $j \in \{1, \ldots, M\}$, and $\hat{\mathcal{N}}_j$ is the

Table 2 Element- and coordinate-ratios in between two refinement steps

i	$\#\mathcal{T}_i$	$\#\mathcal{N}_i$	ρ^i_{elem}	ρ^i_{coord}
0	4	6	–	–
1	13	12	3.25	2.00
2	39	28	2.00	2.33
3	123	74	3.15	2.64
4	297	164	2.41	2.22
5	693	365	2.33	2.23
6	1482	762	2.14	2.09
7	3085	1568	2.08	2.06
8	6239	3147	2.02	2.01
9	12,597	6328	2.02	2.01
10	25,221	12642	2.00	2.00

corresponding set of nodes. Note that $\mathcal{T}_N = \hat{\mathcal{T}}_0$, i.e., we start our coarsening routine from the finest mesh and mark all elements for coarsening. As our coarsening routine is not inverse, we need more coarsening steps than refinement steps to recover the initial triangulation. In other words, $M \geq N$. A blue refinement is coarsened in a two-step procedure. Thus, we expect a refinement–coarsening step ratio of about $1 : 2$ and consequently $M \approx 2 \cdot N$. For a better quantification, we compute the element-ratio

$$\hat{\rho}^j_{\text{elem}} = \frac{\#\hat{\mathcal{T}}_j}{\#\hat{\mathcal{T}}_{j+1}}$$

and the coordinate-ratio

$$\hat{\rho}^j_{\text{coord}} = \frac{\#\hat{\mathcal{N}}_j}{\#\hat{\mathcal{N}}_{j+1}}$$

in each coarsening step j. The results are shown in Table 3. The geometric means are given by $\overline{\hat{\rho}}_{\text{elem}} = 1.55$ and $\overline{\hat{\rho}}_{\text{coord}} = 1.47$. In this example, we get $M = 2 \cdot N$ and in other experiments we also observed $M \approx 2 \cdot N$. In terms of efficiency, this means that our coarsening strategy is not as efficient as its refinement counterpart. We need to expect twice as many coarsening steps to undo the refinement as is needed for refining. To get a better feeling about time efficiency, we measured the time for the refinement and coarsening part for this example. The computational time for the refinement part is 0.0665 s while the coarsening part takes 0.2171 s. Since twice as many coarsening steps are required than refinement steps, we conclude that one coarsening step is slightly more time-consuming than one refinement step. However, coarsening from 10^4 degrees of freedom to 6 can be done in about a fifth of a second, which is still very efficient.

6.3 Scalability of the coarsening routine

This leads us to the question of scalability. To this end, we measure 20 times the computational time of the coarsening routine by use of MATLAB's builtin `tic/toc`

Table 3 Element- and coordinate-ratios in between two coarsening steps

j	$\#\hat{\mathcal{T}}_j$	$\#\hat{\mathcal{N}}_j$	$\hat{\rho}^j_{\mathrm{elem}}$	$\hat{\rho}^j_{\mathrm{coord}}$
0	25,221	12642	–	–
1	16,610	8335	1.52	1.52
2	13,454	6756	1.23	1.23
3	8851	4453	1.52	1.52
4	6956	3505	1.27	1.27
5	4484	2268	1.55	1.55
6	3485	1768	1.29	1.28
7	2199	1123	1.58	1.57
8	1684	865	1.31	1.30
9	1052	547	1.60	1.58
10	800	421	1.32	1.30
11	486	261	1.65	1.61
12	360	198	1.35	1.32
13	203	115	1.77	1.72
14	143	85	1.42	1.35
15	70	45	2.04	1.89
16	48	34	1.46	1.32
17	19	16	2.53	2.13
18	12	11	1.58	1.45
19	6	7	2.00	1.57
20	4	6	1.50	1.17

and plot the mean of the measured times above the number of nodes for newest vertex bisection and red-green-blue refinement. The numerical experiments on NVB are based on the implementation in [9]. This is displayed in Fig. 24. The plot shows an almost linear behavior between the number of elements and the computational time in seconds. We see that RGB has some offset which is explained by a more involved determination of red middle elements and the fact that a CLOSURE step is carried out within a while-loop. A priori, it is unclear how often the while-loop iterates. This uncertainty in the CLOSURE step was already examined for the refinement routine in [12]. Due to the structure of the mesh, for an adaptive RGB refined mesh of an initial weak BDD triangulation, at most two iterations are needed. The CLOSURE step is thus predictable and can not cause a huge increase in computational time.

6.4 Local coarsening

So far, we have used coarsening by marking all elements for coarsening. This time, we want to show that our algorithm can be used in an adaptive setting, cf.

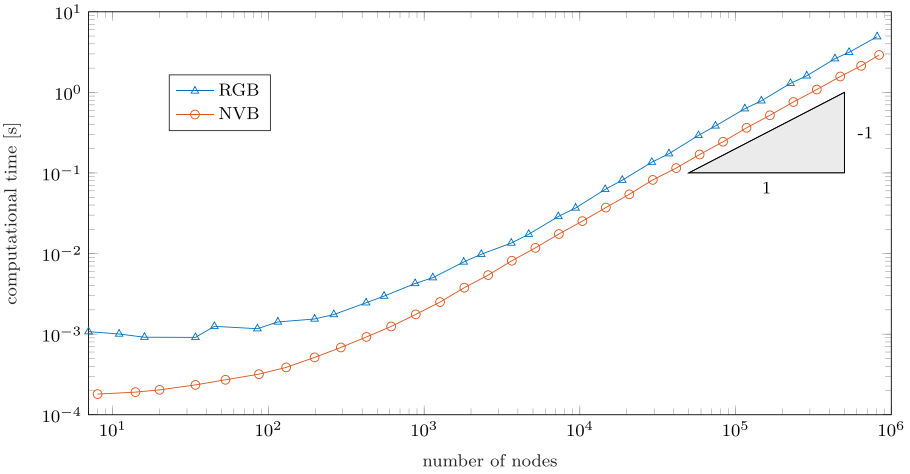


Fig. 24 Scalability of our RGB coarsening algorithm in comparison with the NVB coarsening algorithm implemented in [9]. A nearly linear behavior between the number of nodes and the computational time can be observed

`example2/` or `example3/` or more generally, for local coarsening. To this end, we define a discrete point set and mark elements for coarsening that include this point; see Section 5.1. We start with a fine triangulation and proceed with this marking strategy and our coarsening algorithm. Figure 25 shows possible local coarsened meshes.

In summary, our proposed coarsening algorithm can be used in an interplay of refinement and coarsening, can coarsen locally and recover the initial triangulation-although not quite as efficiently. However, the latter point does usually not play a major role, as only a few coarsening steps are integrated in an adaptive routine.

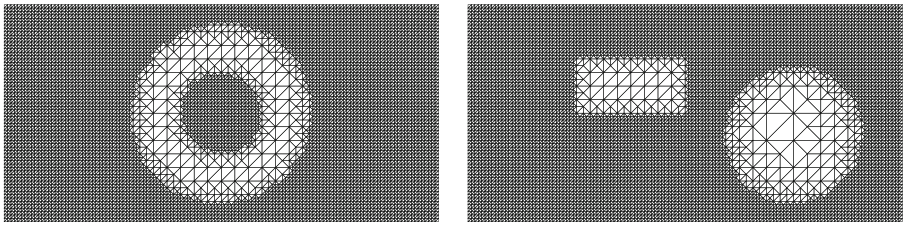


Fig. 25 Local coarsening

Acknowledgments The authors would like to thank Mazen Ali, Stefan Ehard, and Marcus Heitel for comments that greatly improved this work.

Funding Open Access funding provided by Projekt DEAL.

References

1. Bank, R.E., Xu, J.: An algorithm for coarsening unstructured meshes. Numer. Math. **73**(1), 1–36 (1996)
2. Bartels, S., Schreier, P.: Local coarsening of triangulations created by bisections. Univ. Freiburg, SFB 611 (2010)
3. Binev, P., Dahmen, W., DeVore, R.: Adaptive finite element methods with convergence rates. Numer. Math. **97**(2), 219–268 (2004)
4. Carstensen, C.: An adaptive mesh-refining algorithm allowing for an H^1-stable L^2-projection onto Courant finite element spaces. Constr. Approx. **20**(4), 549–564 (2004)
5. Carstensen, C., Feischl, M., Page, M., Praetorius, D.: Axioms of adaptivity. Comput. Math. Appl. **67**(6), 1195–1253 (2014)
6. Cascon, J.M., Kreuzer, C., Nochetto, R.H., Siebert, K.G.: Quasi-optimal convergence rate for an adaptive finite element method. SIAM J. Numer. Anal. **46**(5), 2524–2550 (2008)
7. Chen, L., Zhang, C.: A coarsening algorithm on adaptive grids by newest vertex bisection and its applications. J. Comput. Math. pp 767–789 (2010)
8. Ciarlet, P.G.: The finite element method for elliptic problems. SIAM (2002)
9. Funken, S.A., Praetorius, D., Wissgott, P.: Efficient implementation of adaptive p1-FEM in Matlab. Comput. Methods Appl. Math. **11**(4), 460–490 (2011)
10. Funken, S.A., Schmidt, A.: ameshcoars – Efficient Implementation of Adaptive Mesh Coarsening in 2D. Software download at https://github.com/aschmidtuulm/ameshcoars (2020)
11. Funken, S.A., Schmidt, A.: ameshref – Efficient Implementation of Adaptive Mesh Refinement in 2D. Software download at https://github.com/aschmidtuulm/ameshref (2018)
12. Funken, S.A., Schmidt, A.: Adaptive mesh refinement in 2D–An efficient implementation in matlab. Comput. Methods Appl. Math. (2018)
13. Funken, S.A., Schmidt, A.: Ameshref: a Matlab-Toolbox for adaptive mesh refinement in two dimensions. In: Numerical Geometry, Grid Generation and Scientific Computing, pp. 269–279. Springer (2019)
14. Karkulik, M., Pavlicek, D., Praetorius, D.: On 2d newest vertex bisection: Optimality of mesh-closure and H1-stability of L2-projection. ASC report 10/2012 Vienna University of Technology (2012)
15. Karkulik, M., Pavlicek, D., Praetorius, D.: On 2D newest vertex bisection: optimality of mesh-closure and H 1-stability of L 2-projection. Constr. Approx. **38**(2), 213–234 (2013)
16. Kossaczkỳ, I.: A recursive approach to local mesh refinement in two and three dimensions. J. Comput. Appl. Math. **55**(3), 275–288 (1994)
17. Mavriplis, D., Jameson, A.: Multigrid solution of the Euler equations on unstructured and adaptive meshes. Multigrid methods: Theory, Applications, and Supercomputing, SF McCormick, ed **110**, 413–429 (1988)
18. Nochetto, R.H., Siebert, K.G., Veeser, A.: Theory of adaptive finite element methods: an introduction. In: Multiscale, Nonlinear and Adaptive Approximation, pp. 409–542. Springer (2009)

19. Ollivier-Gooch, C.: Coarsening unstructured meshes by edge contraction. Int. J. Numer. Methods Eng. **57**, 391–414 (2003). https://doi.org/10.1002/nme.682
20. Pavlicek, D.: Optimalität adaptiver FEM, Bachelor Thesis (in German). Institute for Analysis and Scientific Computing Vienna University of Technology (2010)
21. Praetorius, D., Weinmüller, E., Wissgott, P.: A space-time adaptive algorithm for linear parabolic problems. ASC report 07/2008 Vienna University of Technology (2008)
22. Schmidt, A.: Adaptive Mesh Refinement in 2D – an Efficient Implementation in Matlab for Triangular and Quadrilateral Meshes. Master's thesis, Universität Ulm (2018)
23. Schmidt, A., Siebert, K.G.: Design of Adaptive Finite Element Software - The Finite Element Toolbox ALBERTA, Lect. Notes Comput. Sci. Eng., vol. 42, Springer. https://doi.org/10.1007/b138692 (2005)
24. Schneiders, R.: Mesh generation and grid generation on the web. http://www.robertschneiders.de/meshgeneration/meshgeneration.html. Accessed: 2020-01-09
25. Sewell, E.: Automatic Generation of Triangulations for Piecewise Polynomial Approximation. Purdue University. https://books.google.de/books?id=zaJfnQEACAAJ (1972)
26. Shu, Z.Y., Wang, G.Z., Dong, C.S.: Adaptive triangular mesh coarsening with centroidal voronoi tessellations. J. Zhejiang Univ. Sci. A **10**(4), 535–545 (2009). https://doi.org/10.1631/jzus.A0820229
27. Stevenson, R.: The completion of locally refined simplicial partitions created by bisection. Math. Comput. **77**(261), 227–241 (2008)