Journal of Computational Science 3 (2012) 420–428

Contents lists available at SciVerse ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A multi-mesh finite element method for Lagrange elements of arbitrary degree

A. Voigt, T. Witkowski*

TU Dresden, Department of Mathematics, Helmholtzstrasse 10, 01062 Dresden, Germany

ARTICLE INFO

Article history:
Received 1 November 2011
Received in revised form 4 June 2012
Accepted 6 June 2012
Available online 13 June 2012

Keywords:
FEM
Multi-mesh
PDE

ABSTRACT

We consider within a finite element approach the usage of different adaptively refined meshes for different variables in systems of nonlinear, time-depended PDEs. To resolve different solution behaviors of these variables, the meshes can be independently adapted. The resulting linear systems are usually much smaller, when compared to the usage of a single mesh, and the overall computational runtime can be more than halved in such cases. Our multi-mesh method works for Lagrange finite elements of arbitrary degree and is independent of the spatial dimension. The approach is well defined, and can be implemented in existing adaptive finite element codes with minimal effort. We show computational examples in 2D and 3D ranging from dendritic growth to solid–solid phase-transitions. A further application comes from fluid dynamics where we demonstrate the applicability of the approach for solving the incompressible Navier–Stokes equations with Lagrange finite elements of the same order for velocity and pressure. The approach thus provides an easy way to implement alternative to stabilized finite element schemes, if Lagrange finite elements of the same order are required.

1. Introduction

Nowadays, adaptive mesh refinement methods are a standard technique in finite element codes. They are used to resolve a mesh due to the local behavior of the solution. When solving systems of PDEs, e.g., in multiphysics problems, the mesh has to be adapted to the behavior of all components of the solution. If these behaviors are different, the use of a single mesh, even if it is adaptively refined, may lead to an inefficient numerical method. In this work we propose a multi-mesh finite element method that makes it possible to resolve the local nature of different components independently of each other. This method works for Lagrange elements of arbitrary degree in any dimension. Furthermore, the method works "on top" of standard adaptive finite element methods. Hence, only small changes are necessary to implement the approach in existing finite element codes. We have implemented the multi-mesh method in the finite element software AMDiS (adaptive multidimensional simulations),[1] see [26], for Lagrange finite elements up to fourth degree for 1D, 2D and 3D.

The usage of multiple, independently refined meshes to discretize different components in systems of PDEs is not new. To our best knowledge, Schmidt [18] was the first who has considered a multi-mesh method in the context of adaptive finite elements. Li et al. have introduced a very similar technique and used it to simulate dendritic growth [14,5,9]. Solin et al. [19,20] have introduced an hp-FEM multi-mesh method, that is implemented in the finite element software Hermes. Although the multi-mesh technique is introduced, in none of these publications the method is formally derived. Furthermore implementation issues are not discussed and detailed runtime results, which compare the overall runtime between the single-mesh and the multi-mesh method are missing. In contrast, in this work we will formally show how multiple meshes are used in the context of assembling matrices and vectors in the assembly step of the finite element methods and will discuss issues related to error estimates for each component. Furthermore, we will compare the runtimes of both methods and show that the multi-mesh method is superior to the single-mesh method, when one component in the system of PDEs can locally be resolved on a coarser mesh. We should further mention other approaches which are commonly used to deal with different meshes for different components of coupled systems. Especially in the case of multi-physics applications a need exists to couple independent simulations code. A standard tool which can be used to couple various finite element codes is MpCCI (mesh-based parallel code coupling interface) [10]. In this approach an interpolation between the different solutions from one mesh to the other is performed which for different resolutions of the involved meshes will lead to a loss in information and is thus not the method of choice for the problems to be discussed in this work.

The paper is structured as follows. In the next section we give a brief overview on adaptive meshes, and introduce the terminology

* Corresponding author.
E-mail addresses: axel.voigt@tu-dresden.de (A. Voigt), thomas.witkowski@tu-dresden.de (T. Witkowski).

[1] AMDiS is a free C++library to solve various types of PDEs using the finite element method. For more information see http://www.amdis-fem.org.

1877-7503/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.jocs.2012.06.004

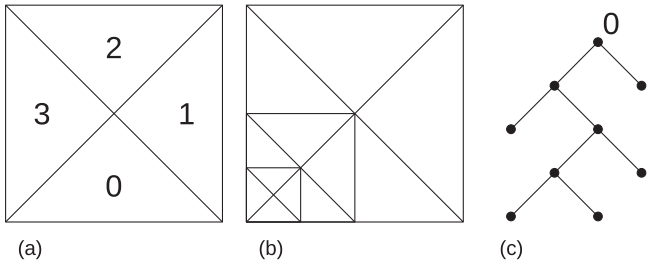

Fig. 1. (a) A two-dimensional macro mesh. (b) Some refinements of it. (c) Binary tree of macro element 0.

used throughout this paper. Section 3 introduces the so-called *virtual mesh assembling*, which is the basis of our multi-mesh method. It is shown, how the coupling meshes are build in a virtual way and how the corresponding coupling operators are assembled on them. In Section 4, we present several numerical experiments in 2D and 3D that show the advantages of the multi-mesh method. The last section summarizes our results.

2. Adaptive meshes

The usage of an adaptive mesh, together with error estimators and a refinement strategy, is becoming a standard in finite element approach for systems of PDEs. For a general overview on this topic see for example [23], and references therein.

2.1. Data structure for adaptive meshes

In this section, we describe how adaptive meshes and the associated algorithms can be implemented. The data structures that are used to store and manipulate adaptive meshes, are the basis for a fast and efficient multi-mesh method, as it is presented in the next section. Even if we concentrate on a specific data structure for mesh representation, other data structures can be used for the multi-mesh concept as well with some simple modifications.

The multi-mesh method as presented here works for both, triangular and quadrilateral elements in 2D and for tetrahedrals and hexahedra in 3D. Furthermore, the method is independent of the refinement strategy, e.g., bisectioning or red-green refinement (4-split). In what follows, we assume, without loss of generality, the meshes to consist of triangles or tetrahedrals, and bisectioning as a refinement strategy. In this case, a mesh can efficiently be stored using a set of binary trees, see Fig. 1. Each binary tree represents the refinement of one element in the coarsest mesh, the so-called macro element. All topological and geometrical information are only stored for the coarse grid representation. The information for refined elements has to be computed from this data. The additional effort to compute the element data multiple times is usually less than 1% of the computational time of the function that process on the mesh and is thus negligible.

2.2. Error estimation and adaptive strategies

The multi-mesh method as presented here does not rely on a specific error estimator. In general, arbitrary error estimator and marker strategies can be used. Because our method refines the meshes independently of each other, the error estimator must also be applied for each component of the PDE. A logical consequence of our approach is the possibility to use also different error estimators for the different components. For all numerical results presented here, we make use of the standard a posterior residual based error estimator as, for example, described by Verfürth [25].

3. Virtual mesh assembling

The basis of our multi-mesh method is the so-called *virtual mesh assembling*. Systems of PDEs usually involve coupling terms. If each component of the system is assembled on a different mesh, special care has to be devoted to these coupling terms. In the next section, we shortly describe this situation. Section 3.2 then introduces the *dual mesh traverse*. This algorithm creates a virtual union of two meshes without creating it explicitly. To the last, we show how to compute integrals, that appear within the assemble procedure, on these virtual meshes.

3.1. Coupling terms in systems of PDEs

To illustrate the techniques, we consider the homogeneous biharmonic equation as a simple example for a system of PDEs. This equation reads:

$$\Delta^2 u = 0 \ \text{in} \ \Omega \quad \text{and} \quad u = \frac{\partial u}{\partial n} = 0 \,\text{on}\, \partial\Omega, \tag{1}$$

with $u \in C^4(\Omega) \cap C^1(\overline{\Omega})$. Using operator splitting, the biharmonic equation can be rewritten as a system of two second order stationary PDEs:

$$\begin{aligned} -\Delta u + v &= 0 \\ \Delta v &= 0 \end{aligned} \tag{2}$$

The standard mixed variational formulation of this system is: find $(u, v) \in H_0^1(\Omega) \times H^1(\Omega)$ such that

$$\begin{aligned} &\int_\Omega \nabla u \nabla \phi \, dx + \int_\Omega v\phi \, dx = 0 \quad \forall \phi \in H^1(\Omega) \\ &\int_\Omega \nabla v \nabla \psi \, dx = 0 \quad \forall \psi \in H_0^1(\Omega) \end{aligned} \tag{3}$$

To discretize these equations, we assume that $\mathcal{T}_h^0$ and $\mathcal{T}_h^1$ are different partitions of the domain Ω into simplices. Then, $V_h^0 = \{v_h \in H^1 : v_h|_T \in P^n \, \forall T \in \mathcal{T}_h^0\}$ and $V_h^1 = \{v_h \in H_0^1 : v_h|_T \in P^n \, \forall T \in \mathcal{T}_h^1\}$ are finite element spaces of globally continuous, piecewise polynomial functions of an arbitrary but fixed degree. We thus obtain: find $(u_h, v_h) \in V_h^0 \times V_h^1$ such that

$$\begin{aligned} &\int_\Omega \nabla u_h \nabla \phi \, dx + \int_\Omega v_h \phi \, dx = 0 \quad \forall \phi \in V_h^0(\Omega) \\ &\int_\Omega \nabla v_h \nabla \psi \, dx = 0 \quad \forall \psi \in V_h^1(\Omega). \end{aligned} \tag{4}$$

Let us define $\{\phi_i | 1 \le i \le n\}$ and $\{\psi_i | 1 \le i \le m\}$ to be the nodal basis of V_h^0 and V_h^1, respectively. Hence, u_h and v_h can be written by the linear combinations $u_h = \sum_{i=1}^n u_i \phi_i$ and $v_h = \sum_{i=1}^m v_i \psi_i$, with u_i and v_i the unknown real coefficients. Using these relations and breaking up the domain in the partitions of Ω, Eq. (4) rewrites to

$$\begin{aligned} &\sum_{j=1}^n u_j \left(\sum_{T \in \mathcal{T}_h^0} \int_T \nabla \phi_j \cdot \nabla \phi_i \right) + \sum_{j=1}^m v_j \left(\sum_{T \in \mathcal{T}_h^0 \cup \mathcal{T}_h^1} \int_T \psi_j \phi_i \right) = 0, \quad i = 1, \ldots, n \\ &\sum_{j=1}^m v_j \left(\sum_{T \in \mathcal{T}_h^1} \int_T \nabla \psi_j \cdot \nabla \psi_i \right) = 0, \quad i = 1, \ldots, m. \end{aligned} \tag{5}$$

To compute the coupling term $\int_T \psi_j \phi_i$, we have to define the union of two different partitions $T_h^0 \cup T_h^1$. For this, we make a restriction on the partitions: any element $T^0 \in \mathcal{T}_h^0$ is either a subelement of an element $T^1 \in \mathcal{T}_h^1$, or vice versa. This restriction is not very strict. It is always fulfilled for the standard refinement algorithms, e.g., bisectioning or red-green refinement, if the initial meshes for all

components share the same macro mesh. Then, $\mathcal{T}_h^0 \cup \mathcal{T}_h^1$ is the union of the locally finest simplices.

The most common way to compute the integrals in (5) is to define local basis functions. We define $\phi_{i,j}$ to be the jth local basis function on an element $T_i \in \mathcal{T}_h^0$. $\psi_{i,j}$ is defined in the same way for elements in the partition $\mathcal{T}_h^1$.

Because the global basis functions ϕ_i and ψ_j are defined on different triangulations of the same domain, it is not straightforward to calculate the coupling term $\int_\Omega \psi_j \phi_i$ in an efficient manner. For evaluating this integral, two different cases may occur: either the integral has to be evaluated on an element from the partition $\mathcal{T}_h^0$ or on an element from $\mathcal{T}_h^1$. For what follows, we fix the first case. In terms of local basis functions we have to evaluate

$$\int_{T_i \in \mathcal{T}_h^0} \psi_{k,l} \phi_{i,j} \tag{6}$$

for some j and l, and there exists an element $T_k \in \mathcal{T}_h^1$, with $T_i \subset T_k$. Our aim is to develop a multi-mesh method that works on the top of existing finite element software. These have implemented specialized methods to evaluate integrals of local basis functions very fast, e.g., precalculated integral tables or fast quadrature rules. All these methods cannot directly be applied to the coupling terms, because $\psi_{k,l}$ in (6) is not a local basis function of the element T_i. The general idea to overcome this problem is to define the basis functions $\psi_{k,l}$ by a linear combination of local basis functions of T_i. Thus,

$$\int_{T_i \in \mathcal{T}_h^0} \psi_{k,l} \phi_{i,j} = \int_{T_i \in \mathcal{T}_h^0} \sum_m (c_{k,m} \phi_{i,m}) \phi_{i,j}, \tag{7}$$

with some real coefficients $c_{k,m}$. For the other case, i.e., the integral in the coupling term is evaluated on an element $T_i \in \mathcal{T}_h^1$, we have

$$\int_{T_i \in \mathcal{T}_h^1} \psi_{k,l} \phi_{i,j} = \int_{T_i \in \mathcal{T}_h^1} \psi_{k,l} \sum_m (c_{i,m} \psi_{k,m}). \tag{8}$$

Summarized, to evaluate the coupling terms, two different techniques have to be defined and implemented. Firstly, the method requires to build a union of two meshes. This leads to an algorithm which we name *dual mesh traverse*. It will be discussed in the next section. Once the union is obtained, we need to calculate the coefficients $c_{i,j}$ and to incorporate them in the finite element assemblage procedure such that the overall change of the standard method is as small as possible.

3.2. Creation of the virtual mesh

The simplest way to obtain the union of two meshes is to employ the data structure they are stored in. Hence, in our case we could explicitly build the union by joining the binary trees of both meshes into a set of new binary trees. Especially when we consider meshes that change in time, this procedure is not only too time-consuming but also requires additional memory to store the joined mesh. To avoid this, we do not directly work on the mesh data but instead use the mesh traverse algorithm, that creates the requested element data on demand. According to this method, we define the *dual mesh traverse* that traverses two meshes in parallel and thus create the union of both meshes in a virtual way.

For the dual mesh traverse the only requirement is that both meshes must share the same macro mesh, but they can be refined independently of each other. Due to this requirement and because of the bisectioning refinement algorithm the following holds: if the intersection of two elements of two different meshes is non empty, then either both elements are equal or one element is a real subelement of the other. To receive the leaf level of the virtual mesh, the dual mesh traverse simultaneously traverses two binary trees, each corresponding to the same macro element in both macro meshes. The algorithm then calls a user defined function, e.g., the element assembling function or an element error estimator, that works on pairs of elements, with both, the larger and the smaller element of the current traverse. The larger of both elements is fixed as long as all smaller subelements in the other mesh are traversed. Fig. 2 shows a simple example for a macro mesh consisting of four macro elements. In the first mesh macro element 0, and in the second mesh macro element 1 are refined once.

3.3. Assembling of element matrices

To speedup the calculation, the form as given by (7) and (8) is not appropriate. To implement these transformations, some changes of the inner assemblage procedure are required. First, we have to distinguish two cases: the smaller of both elements defines either the space of test functions, or it defines the space of trial functions. For the first case, we consider the coupling term $\int_\Omega \psi_i \phi_j$ in (5), with some local basis functions ψ_i and ϕ_j, that has to be assembled on a virtual mesh. Then, for some elements T and T', with $T' \subset T$, the element matrix $M_{T'}$ is given by

$$\begin{aligned} M_{T'} &= \begin{pmatrix} \int_{T'} \psi_0 \phi_0 & \cdots & \int_{T'} \psi_0 \phi_n \\ \vdots & & \vdots \\ \int_{T'} \psi_n \phi_0 & \cdots & \int_{T'} \psi_n \phi_n \end{pmatrix} = \begin{pmatrix} \int_{T'} \sum_i (c_{0i} \phi_i) \phi_0 & \cdots & \int_{T'} \sum_i (c_{0i} \phi_i) \phi_n \\ \vdots & & \vdots \\ \int_{T'} \sum_i (c_{ni} \phi_i) \phi_0 & \cdots & \int_{T'} \sum_i (c_{ni} \phi_i) \phi_n \end{pmatrix} \\ &= \begin{pmatrix} \sum_i c_{0i} \int_{T'} \phi_i \phi_0 & \cdots & \sum_i c_{0i} \int_{T'} \phi_i \phi_n \\ \vdots & & \vdots \\ \sum_i c_{ni} \int_{T'} \phi_i \phi_0 & \cdots & \sum_i c_{ni} \int_{T'} \phi_i \phi_n \end{pmatrix} = \mathcal{C} \cdot \begin{pmatrix} \int_{T'} \phi_0 \phi_0 & \cdots & \int_{T'} \phi_0 \phi_n \\ \vdots & & \vdots \\ \int_{T'} \phi_n \phi_0 & \cdots & \int_{T'} \phi_n \phi_n \end{pmatrix}, \end{aligned} \tag{9}$$

where ϕ_i are the local basis function defined on T', ψ_i are the local basis function defined on T, and $\mathcal{C}$ is the transformation matrix for the local basis function from T to T'. This shows, that to assemble the element matrix of a virtual element, there is no need for larges changes within the assemble procedure. The finite element code needs only to assemble the element matrix of the smaller element T' and multiply the result with the transformation matrix. Hence, if the transformation matrices can be computed easily, the overhead for virtual element assembling can be neglected. A different approach for a multi-mesh hp-FEM is presented by Solin et al. [19,20]. Their method is based on transformating quadrate points which is harder to implement in existing single mesh finite element codes.

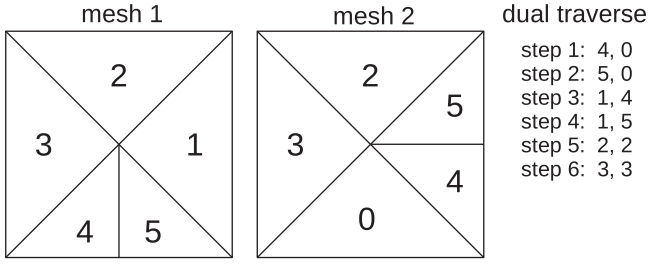

Fig. 2. Dual mesh traverse on two independent refined meshes on the same macro mesh.

The second case, where the smaller element defines the space of trial functions, is very similar. The same calculation as above shows that the following holds:

$$M_{T'} = \begin{pmatrix} \int_{T'} \phi_0\phi_0 & \dots & \int_{T'} \phi_0\phi_n \\ \vdots & & \vdots \\ \int_{T'} \phi_n\phi_0 & \dots & \int_{T'} \phi_n\phi_n \end{pmatrix} \cdot \mathcal{C}^T \tag{10}$$

The above calculations can be immediately generalized for general zero order terms of the form $\int_\Omega \psi_i c \phi_j$ with $c \in L^\infty(\Omega)$, as the transformation matrices $\mathcal{C}$ are independent of c. In a similar way we can also reformulate the element matrices for general first and second order terms. For a general second order term of the form $\int_\Omega \nabla\psi_i \cdot \mathbf{A} \nabla\phi_j$, with $\mathbf{A} : \Omega \mapsto \mathbb{R}^{d\times d}$, the element matrix $M_{T'}$ can be rewritten in the same way as we have done it in (9) for a zero order terms:

$$M_{T'} = \begin{pmatrix} \int_{T'} \nabla\psi_0 \cdot \mathbf{A}\nabla\phi_0 & \dots & \int_{T'} \nabla\psi_0 \cdot \mathbf{A}\nabla\phi_n \\ \vdots & & \vdots \\ \int_{T'} \nabla\psi_n \cdot \mathbf{A}\nabla\phi_0 & \dots & \int_{T'} \nabla\psi_n \cdot \mathbf{A}\nabla\phi_n \end{pmatrix} = \mathcal{C}_\nabla \cdot \begin{pmatrix} \int_{T'} \nabla\phi_0 \cdot \mathbf{A}\nabla\phi_0 & \dots & \int_{T'} \nabla\phi_0 \cdot \mathbf{A}\nabla\phi_n \\ \vdots & & \vdots \\ \int_{T'} \nabla\phi_n \cdot \mathbf{A}\nabla\phi_0 & \dots & \int_{T'} \nabla\phi_n \cdot \mathbf{A}\nabla\phi_n \end{pmatrix} \tag{11}$$

where the coefficients of the matrix $\mathcal{C}_\nabla$ are defined such that

$$\nabla\psi_i = \sum_{j=0}^{n} c_{ij} \nabla\phi_i. \tag{12}$$

If the smaller element defines the space of trial functions, we can establish the same relation as in (10). For general first order terms of the form $\int_\Omega \psi_i \mathbf{b} \cdot \nabla\phi_j$, with $\mathbf{b} \in [L^\infty(\Omega)]^d$, it is simple to check that for the case the smaller element defines the test space, the element matrix $M_{T'}$ can be calculated on the smaller element and multiplied with $\mathcal{C}$ from the left. If the smaller element defines the space of trial function, the element matrix calculated on the smaller element must be multiplied with $\mathcal{C}_\nabla^T$ from the right.

3.4. Calculation of transformation matrices

We have shown that if the transformation matrix is calculated for a given tuple of small and large element, the additional cost for virtual mesh assembling is very small. In this section, we show how to compute these transformation matrices efficiently. We formally define a virtual element pair by the tuple

$$(T, \{\alpha_0, \dots, \alpha_n\}) = (T, \alpha) \quad \text{with } \alpha_i \in \{\mathrm{L}, \mathrm{R}\}, \tag{13}$$

where T is the larger element of the pair and α is an ordered set that is interpreted as the refinement path for element T. Thus, L denotes the "left" and R denotes the "right" children of the element. Furthermore, we define a function *TRA* that uniquely maps a virtual element pair to the smaller element. It is defined recursively by:

$$TRA(T, \emptyset) = T$$

$$TRA(T, \{\alpha_0, \alpha_1, \dots, \alpha_n\}) = \begin{cases} TRA(T_\mathrm{L}, \{\alpha_1, \dots, \alpha_n\}) & \text{if } \alpha_0 = \mathrm{L} \\ TRA(T_\mathrm{R}, \{\alpha_1, \dots, \alpha_n\}) & \text{if } \alpha_0 = \mathrm{R}, \end{cases}$$

where T_L is the left child of the element T, and T_R the right child of the element, respectively. In the same way we can now define transformation matrices as functions on refinement path:

$$\mathcal{C}(\emptyset) = I$$

$$\mathcal{C}(\{\alpha_0, \alpha_1, \dots, \alpha_n\}) = \begin{cases} \mathcal{C}_\mathrm{L} \cdot \mathcal{C}(\{\alpha_1, \dots, \alpha_n\}) & \text{if } \alpha_0 = \mathrm{L} \\ \mathcal{C}_\mathrm{R} \cdot \mathcal{C}(\{\alpha_1, \dots, \alpha_n\}) & \text{if } \alpha_0 = \mathrm{R}, \end{cases}$$

where $\mathcal{C}_\mathrm{L}$ and $\mathcal{C}_\mathrm{R}$ are the transformation matrices for the left child and the right child, respectively, of the reference element.

3.5. Implementation issues

Although the calculation of transformation matrices is quite fast, it can considerably increase the time for assembling the linear system. This is especially the case, if one mesh is much coarsen in some regions than the other mesh. To circumvent this problem, we have implemented a cache, that stores the transformation matrices. In the mesh traverse routine, an 64 bit integer data type stores the refinement path bit-wise, as it is defined by (13). If the bit on the ith position is set, the finer element is a right-refinement of its parent element, otherwise it is a left-refinement of it. Of coarse this limits the level gap between the coarser and the finer level to be less than or equal to 64. But we have not found any practical simulations, where this value is more than 30. Using this data type makes it then easy to define associative array that uniquely maps a refinement path to a transformation matrices. If a transformation matrix was computed for a given refinement path for the first time, it will be stored in this array. To access previously computed matrices using the integer key is then very cheap. In general this data structure should be restricted to a fixed number of matrices to not to spend too much of memory. In all of our simulations, the number of matrices that should be stored in the cache never exceed 100,000. Also in the 3D case with linear elements the overall memory usage is then around 2 MB, and can thus be neglected. Therefore, we have not yet considered to implement an upper limit for the cache. More information about the number and memory usage of the transformation matrices is given in the next section when presenting the numerical results.

All of the algorithms described here can be easily adjusted if other data structure than binary trees are used to represent the mesh. This may be the case if, e.g., a red-green refinement strategy is used, or if the mesh consists of quadrilaterals or cubes. In both case, quadtrees or octrees are employed to represent the adaptive mesh structure. For these data structures, transformation matrices can be calculated in the same way as we have done it in this section for binary trees. Of course, the refinement path, as defined in Section 3.4, cannot be stored in this way. Here, either at least two bits for quadtrees or at least three bits for octrees are required to store the information about the children on the next refinement level.

4. Numerical results

In this section, we present several examples, where the multi-mesh approach is superior in contrast to the standard single-mesh finite element method. Examples to be considered are phase-field equations to study solid–liquid and solid–solid phase transitions. For a recent review we refer to [16]. These equations involve at least one variable, the phase-field variable, which is almost constant in most parts of the domain and thus can be discretized within these parts using a coarse mesh. At the interface region a high resolution is required to resolve the smooth transition between the different phases. A second variable in such systems is typically a diffusion field which in most cases varies smoothly across the whole domain and thus will require a finer mesh outside of the interface and a coarser mesh within the interface. Such problems are therefore well suited for a multi-mesh approach, as has already been demonstrated in Li et al. [14,5] and Schmidt [18], but without detailed runtime comparisons. We will consider dendritic growth in solidification and coarsening phenomena in binary alloys to demonstrate the applicability.

Other examples for which large computational savings due to the use of the multi-mesh approach are expected are diffuse interface and diffuse domain approximations for PDEs to be solved on surfaces are within complicated domains. The approaches introduced in [17,15], respectively, use a phase-field function to implicitly describe the domain the PDE has to be solved on. For the same reason as in phase-transition problems the distinct solution behavior of the different variables will lead to large savings if the multi-mesh approach is applied. The approach has already been used for applications such as chaotic mixing in microfluidics [1], tip splitting of droplets with soluble surfactants [21], and chemotaxis of stem cells in 3D scaffolds [13].

As a further example we demonstrate that the multi-mesh approach can also be used to easily fulfill the inf-sup condition for saddle-point problems if both components are discretized using linear Lagrange elements. We demonstrate this numerically for the incompressible Navier–Stokes equation with piecewise linear elements for velocity and pressure, but a finer mesh used for the velocity. Such an approach might be superior to mixed finite elements of higher order or stabilized schemes in terms of computational efficiency and implementational efforts. For a review on efficient finite element methods for the incompressible Navier–Stokes equation we refer to [22]. We demonstrate the applicability of the multi-mesh approach on the classical driven cavity problem.

4.1. Dendritic growth

We first consider dendritic growth using a phase-field model, which today is the method of choice to simulate microstructure evolution during solidification. For reviews we refer to [3]. A widely used model for quantitative simulations of dendritic structures was introduced by Karma and Rappel [11,12], which reads in non-dimensional from

$$\begin{aligned} A^2(n)\partial_t\phi &= (\phi - \lambda u(1-\phi^2))(1-\phi^2) + \nabla(A^2(n)\nabla\phi) + \sum_{i=1}^{d}\partial_{x_i}\left(|\nabla\phi|^2 A(n)\frac{\partial A(n)}{\partial_{x_i}\phi}\right) \\ \partial_t u &= D\nabla^2 u + \frac{1}{2}\partial_t\phi, \end{aligned} \tag{14}$$

where $d=2, 3$ is the dimension, D is the thermal diffusivity constant, $\lambda = D/a_2$, with $a_2 = 0.6267$ is a coupling term between the phase-field variable ϕ and the thermal field u and A is an anisotropy function. For both, simulation in 2D and 3D, we use the following anisotropy function:

$$A(n) = (1-3\epsilon)\left(1 + \frac{4\epsilon}{1-3\epsilon}\frac{\sum_{i=1}^{d}\phi_{x_i}^4}{|\nabla\phi|^4}\right), \tag{15}$$

where ϵ controls the strength of the anisotropy and $n = (\nabla\phi)/(|\nabla\phi|)$ denotes the normal to the solid–liquid interface. In this setting the phase-field variable is -1 in liquid and 1 in solid, and the melting temperature is set to be zero. As boundary condition we set $u = -\Delta$ to specify an undercooling. For the phase-field variable we use zero-flux boundary conditions.

The time integration is done using a semi-implicit Euler method, which yields a sequence of nonlinear stationary PDEs:

$$\begin{aligned} &\frac{A^2(n_n)}{\tau}\phi_{n+1} + f + g - \nabla(A^2(n_n)\nabla\phi_{n+1}) - \mathcal{L}[A(n_n)] = \frac{A^2(n_n)}{\tau}\phi_n \\ &\frac{u_{n+1}}{\tau} - D\nabla^2 u_{n+1} - \frac{1}{2}\frac{\phi_{n+1}}{\tau} = \frac{u_n}{\tau} - \frac{1}{2}\frac{\phi_n}{\tau}. \end{aligned} \tag{16}$$

with $f = \phi_{n+1}^3 - \phi_{n+1}$, $g = \lambda(1-\phi_{n+1}^2)^2 u_{n+1}$ and

$$\mathcal{L}[A(n_n)] = \sum_{i=1}^{d}\partial_{x_i}\left(|\nabla\phi_{n+1}|^2 A(n_n)\frac{\partial A(n_n)}{\partial_{x_i}\phi_n}\right).$$

We now linearize the involved nonlinear terms f and g:

$$\begin{aligned} f &\approx (3\phi_n^2 - 1)\phi_{n+1} - 2\phi_n^3 \\ g &\approx \lambda(1-\phi_n^2)^2 u_{n+1} \end{aligned} \tag{17}$$

and obtain a linear system for ϕ_{n+1} and u_{n+1} to be solved in each time step.

To compare the multi-mesh method with a standard adaptive finite element approach, we have computed a dendrite using linear finite elements. The following parameters are used: $\Delta = 0.65$, $D = 1.0$, $\epsilon = 0.05$.

We use a constant timestep $\tau = 1.0$ up to time 4000. To speedup the computation we have employed the symmetry of the solution and limited the computation to the upper right quadrant with a domain size of 800 into each direction. The adaptive mesh refinement relies on the residuum based a posteriori error estimate. By setting C_0 to 0 and C_1 to 1, we restrict the estimator to the jump residuum only. We have set the tolerance to be $tol_\phi = 0.5$ and $tol_u = 0.25$. For adaptivity, the equidistribution strategy with parameters $\theta_R = 0.8$ and $\theta_C = 0.2$ was used. Thus, the interface thickness is resolved by around 20 grid points.

The result of both computations coincides at the final timestep. As a quantitative comparison we use the tip velocity of the dendrite. As reported by Karma and Rappel [12], for this parameter set analytical calculations lead to a steady-state tip velocity $V_{\text{tip}} = 0.0469$. In both of our calculations, the tip velocity varies around 1% to this value. Using the single-mesh method, within the final timestep the mesh consists of 429,148 vertices for each component. Thus, the overall system of linear equations has 858,296 unknowns. When our multi-mesh method is used, the same solution can be obtained with a mesh for the phase-field with 401,544 vertices and 46,791 vertices for the temperature field. Here, the system of linear equations consists only of 448,335 unknowns. The gap between the number of vertices required to resolve the phase field and the temperature field increases over time, see Fig. 3 that shows the development of the mesh size over time for both methods. Fig. 4

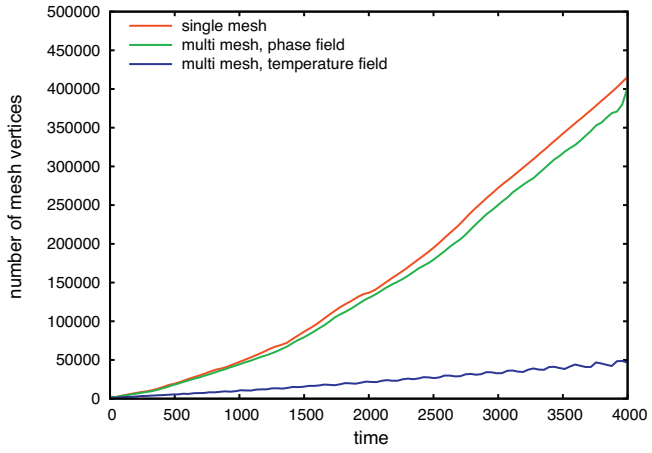

Fig. 3. Evolution of mesh vertices in time for the phase field and the temperature field using single-mesh and multi-mesh method.

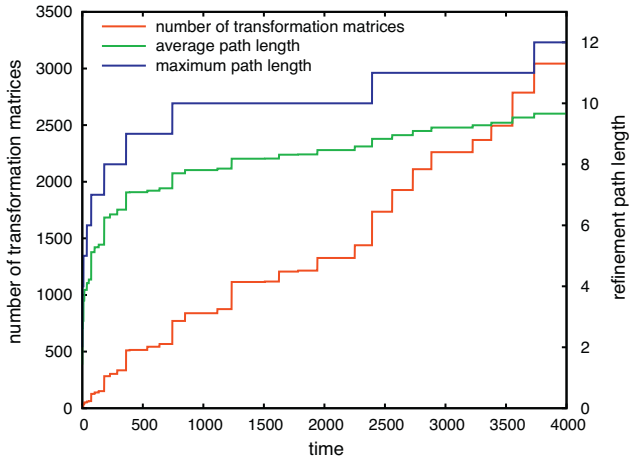

Fig. 5. Evolution of the number of transformation matrices and the refinement path length for the 2D dendritic growth computation.

Table 1
Comparison of runtime when using single-mesh and multi-mesh method for a 2D dendritic growth simulation.

	Single-mesh	Multi-mesh	Speedup
Assembler	11,369 s	10,741 s	5.5%
Solver: UMFPACK	12,799 s	7603 s	40.5%
Solver: BiCGStab(ℓ)	27,436 s	8178 s	70.1%
Estimator	6444 s	3259 s	50.5%
Overall with UMFPACK	30,612 s	21,603 s	29.4%
Overall with BiCGStab(ℓ)	45,249 s	22,178 s	50.9%

Table 2
Matrix related data for the system of linear equations at the final timepoint ($t = 4000$) in a 2D dendritic growth simulation.

	Single-mesh	Multi-mesh
Number of unknowns	858,140	448,335 (401,544 + 46,791)
Matrix non zero values	11,811,959	6,057,416
Matrix size	138.4 MB	71.0 MB
Transformation matrix size	0 MB	0.35 MB
BiCGStab(ℓ) iterations	72	51
Condition number estimate	$4.56 \cdot 10^7$	$8.87 \cdot 10^6$

qualitatively compares the meshes of the phase-field variable and the temperature field which shows the expected finer resolution of the phase-field mesh within the interface and its coarser resolution within the solid and liquid region.

The computational time for both methods is compared in Table 1. The assemblage procedure in the multi-mesh method is around 6% faster, although computing the element matrices is slower due to the extra matrix–matrix multiplication. This is easily explained by the fact that we have much less element matrices to compute and the overall matrix data structure is around 50% smaller with respect to the number of non-zero entries, see also the more detailed data in Table 2. This is also reflected in the solution time for the linear system. We have run all computation twice, with using either UMFPACK, a multifrontal sparse direct solver [4], or the BiCGStab(ℓ) with the parameter $\ell = 2$ and diagonal preconditioning, that is part of the Math Template Library (MTL4), see [8]. When using the first one within the multi-mesh method, the solution time can be reduced by 40% and also the memory usage, which is often the most critical limitation in the usage of direct solvers, is reduced in this magnitude. An even more drastic reduction of the computation time can be achieved when using an iterative solver. Here, the number of iterations is around 30% less with the multi-mesh method and each iteration is faster due to the smaller matrix. This is explained in more detail in Table 2, that shows all relevant matrix data for the very last timestep of the simulation. Not only the overall matrix size is smaller, but also the condition number of the matrices decreases for this PDE when using the multi-mesh method. Hence, the iterative solver needs less iteration to solve the system of linear equations. The memory overhead for the multi-mesh method is quite small. At the very last timestep, only 3215 transformation matrices are stored, requiring 0.35 MB of memory. Fig. 5 shows the evolution of the number of stored transformation matrices in time. Furthermore, it shows the average and maximum length of the refinement paths which are used as keys to find the precomputed transformation matrices.

The time for error estimation is halved, as expected, since it scales linearly with the number of elements in the mesh. Altogether, the time reduction is significant in all parts of the finite element method for this example. In addition the approach also leads to a drastic reduction in the memory usage.

The results are even more significant in 3D. We compared a single-mesh computation with the multi-mesh method using the following model parameters: $\Delta = 0.55$, $D = 1$, $\epsilon = 0.05$. We have run the simulation with a constant timestep $\tau = 1.0$ up to time 2500. The evolution of degrees of freedom over time is quite similar to the 2D

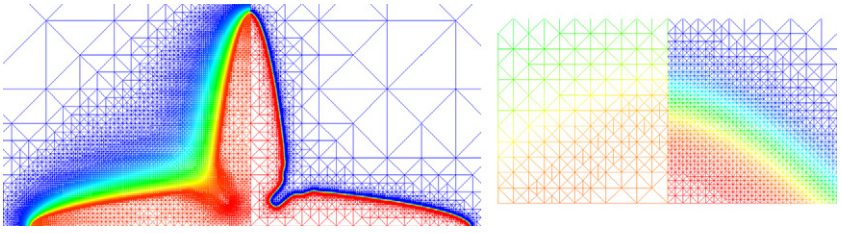

Fig. 4. (a) 2D dendrite computed at $t = 4000$ using the multi-mesh method with the parameters $\Delta = 0.65$, $D = 1.0$, $\epsilon = 0.05$ and a timestep $\tau = 1.0$. Left shows the mesh of the temperature field and right shows the mesh of the phase field. (b) Zoom into the upper tip showing the different resolution of both meshes.

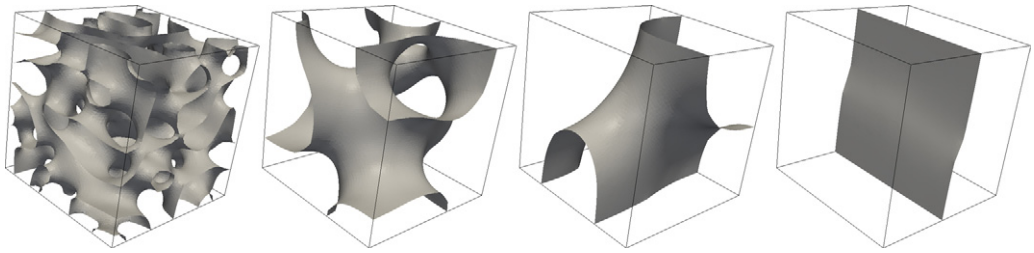

Fig. 6. Solution of the Cahn–Hilliard equation for $t = 0.02$, $t = 1.0$, $t = 5.0$ and $t = 12.50$.

example. When using the multi-mesh method, the time for solving the linear system, again using the BiCGStab(ℓ) solver with diagonal preconditioning, can be reduced by around 60%. The time for error estimation is around half the time needed by the single-mesh method. Because the time for assembling the linear system is more significant in 3D, the overall time reduction with the multi-mesh method is around 24.4%.

4.2. Coarsening

As a second example we consider coarsening of a binary structure using a Cahn–Hilliard equation. We here concentrate only on the phenomenological behavior of the solution and thus consider the simplest isotropic model, which reads

$$\begin{aligned} \partial_t \phi &= \Delta \mu \\ \mu &= -\epsilon\, \Delta \phi + \frac{1}{\epsilon} G'(\phi) \end{aligned} \tag{18}$$

for a phase-field function ϕ and a chemical potential μ. The parameter ϵ again defines a length scale over which the interface is smeared out, and $G(\phi) = 18\phi^2(1-\phi)^2$ defines a double-well potential. To discretize in time we again use a semi-implicit Euler scheme

$$\begin{aligned} \frac{1}{\tau}\phi_{n+1} - \Delta \mu_{n+1} &= \frac{1}{\tau}\phi_n \\ \mu_{n+1} + \epsilon\, \Delta \phi_{n+1} - \frac{1}{\epsilon} G'(\phi_{n+1}) &= 0 \end{aligned} \tag{19}$$

in which we linearize $G'(\phi_{n+1}) \approx G''(\phi_n)\phi_{n+1} + G'(\phi_n) - G''(\phi_n)\phi_n$.

To compare our multi-mesh method with a standard adaptive finite element approach, we have computed the spinodal decomposition and coarsening process using Lagrange finite elements of fourth order. We use $\epsilon = 5 \cdot 10^{-4}$. The adaptive mesh refinement relies on the residuum based a posteriori error estimate. As we have done it in the dendritic growth simulation, also here only the jump residuum is considered, i.e., the constants C_0 and C_1 are set to 0 and 1, respectively. For both methods, the error tolerance are set to $tol_\phi = 2.5 \cdot 10^{-4}$ and $tol_\mu = 5 \cdot 10^{-2}$. For adaptivity, the equidistribution strategy with parameters $\theta_R = 0.8$ and $\theta_C = 0.2$ was used. Using these parameters, the interface thickness is resolved by around 10 grid points.

The simulation was started from noise. The first mesh was globally refined with 196,608 elements. The constant timestep was chosen to be $\tau = 10^{-3}$. We have disabled the adaptivity for the first 10 timesteps, until a very first coarsening in the domain was achieved. Then the simulation was executed up to $t = 13.0$, where both phases are nearly separated. Fig. 6 shows the phase field, i.e., the 0.5 contour of the first solution variable, for four different timesteps. The number of elements and degrees of freedom is linear to the area of the interface that must be resolved on the domain. Indeed, the chemical potential can be resolved on a mush coarser grid, since it is independent of the resolution of the phase field. In the final state, the chemical potential is constant on the whole domain, and the macro mesh (which consists of 6 elements in this simulation) is enough to resolve it. The evolution of the number of

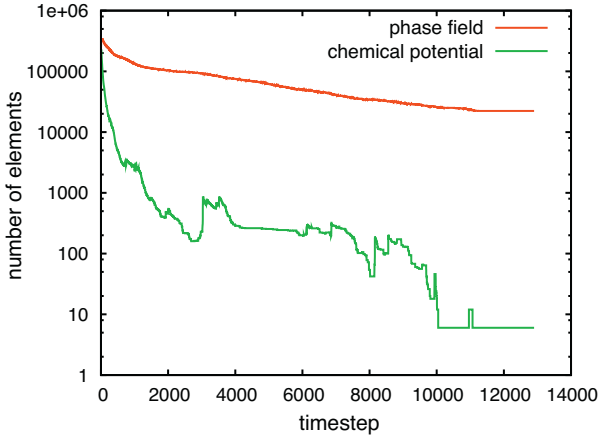

Fig. 7. Evolution of number of elements for both variables of the Cahn–Hilliard equation.

elements for both variable over time is plotted in Fig. 7. As expected, the number of elements for the phase field monotonously decreases as its area shrinks due to the coarsening process. The number of elements for the chemical potential rapidly decreases at the very first beginning, as the initial mesh is over refined to resolve this variable. For most of the simulation, the number of elements of the chemical potential is three orders of magnitude smaller the number of elements for the phase field variable. This gap is also reflected in the computation time for the single-mesh and the multi-mesh method, which are compared in Table 3. The assembling procedure of the multi-mesh method is now 4.7% slower in comparison to the single-mesh method. The main reason therefore is that the transformation matrices are here large due to the 4th order finite element in 3D. They are of size 35×35, and thus slow down the assembling procedure more than in the 2D example before, where the transformation matrices are of size 3×3 for linear finite elements. This small surcharge is paid off when comparing solver and error estimator run times. For solving the linear system of equations we again make use of BiCGStab(ℓ) solver with parameter $\ell = 2$ and diagonal preconditioning. To solve the systems arising in the multi-mesh methods it takes less than 25% of time required for solving the systems when using the single-mesh approach. Here we see again both effects as already described in the numerical example before: each iteration of the solver is faster due to smaller matrices and vectors, and the system of equations are better conditioned in

Table 3
Comparison of runtime when using single-mesh and multi-mesh method for a 3D coarsening simulation using 4th order Lagrange finite elements.

	Single-mesh	Multi-mesh	Speedup
Assembler	19,718 s	20,649 s	−4.7%
Solver: BiCGStab(ℓ)	26,178 s	6312 s	75.8%
Estimator	18,016 s	6967 s	61.3%
Overall	63,912 s	33,928 s	46.9%

Table 4
Comparison of eddy position for $Re = 1000$ in the driven cavity model.

	Eddy 1	Eddy 2	Eddy 3	Eddy 4
Single-mesh	0.5310, 0.5658	0.8633, 0.1116	0.0838, 0.0775	0.9937, 0.0062
Multi-mesh	0.5305, 0.5671	0.8669, 0.1125	0.0813, 0.0750	0.9953, 0.0062
Wall	0.5308, 0.5660	0.8643, 0.1115	0.0832, 0.0775	0.9941, 0.0066
Ghia et al.	0.5313, 0.5625	0.8594, 0.1094	0.0859, 0.0781	0.9922, 0.0078

the case of the multi-mesh method leading to a smaller number of overall iterations. The overall solution time reduces from around 1064 min when using the single-mesh method to 565 min when using the multi-mesh approach.

4.3. Fluid dynamics

For the last example, we demonstrate that the multi-mesh method can also be used to solve problems in fluid dynamics using standard linear finite elements. The inf-sub stability is thereby established by using two different meshes. As an example we construct a problem in 2D. The mesh for the velocity components is refined twice more than the mesh for pressure. In the 3D case, the velocity mesh has to be refined three times to get the corresponding refinement structure. This discretization was introduced by Bercovier and Pironneau [2], and was analyzed and proven to be stable by Verfürth [24]. Although this is not the most efficient technique to ensure the inf-sub stability condition, it is a very simple way if the usage of multiple meshes is supported in the used finite element software. We consider the standard instationary Navier–Stokes equation given by

$$\begin{aligned} &\partial_t u - \nu\nabla^2 u + u\cdot\nabla u + \nabla p = f \\ &\nabla\cdot u = 0, \end{aligned} \tag{20}$$

where u is the velocity, p the pressure and $\nu > 0$ is the kinematic viscosity. The time is discretized by the standard backward Euler method. The nonlinear term in (20) is linearized by $u_n \cdot \nabla u_{n+1}$.

The model problem is the "driven cavity" flow, as described and analyzed in [27,7]. In a unit square, the boundary conditions for the velocity are set to be zero on the left, right and lower part of the domain. On the top, the velocity into x direction is set to be one and into y direction to be zero. In the upper corners, both velocities are set to be zero, which models the so-called non-leaky boundary conditions. The computation was done for several Reynold numbers varying between 50 and 1000. First, we have used the single-mesh method with a standard Taylor–Hood element, i.e., second order Lagrange finite element for the velocity components and linear Lagrange finite element for the pressure. Then, we have compared these results with the multi-mesh method, where for both components linear finite elements were used and the mesh for velocity is refined twice more than the pressure. All computations were done with a fixed timestep $\tau = 0.01$ and aborted, when the relative change in velocity and pressure is less than 10^{-6}. For verification purposes, Table 4 compares all the positions of all eddies in our results with reference values from the work of Ghia et al. [7] and Wall [27].

In both, the single-mesh method and the multi-mesh method, all finite element spaces have the same number of unknowns. This is the reason, why the usage of the latter one is not faster in contrast to the single-mesh method. The time for assembling the linear system growth is from 4.13 s to 5.79 s, which is mainly caused by the multiplication of the element matrices with the transformation matrices. Instead, the average solution time with a BiCGStab(ℓ) solver and ILU preconditioning decreases from 10.18 s to 8.88 s. Although the linear systems have the same number of unknowns, the linear systems resulting from the single-mesh method are denser due to the usage of second order finite elements. The number of non-zero entries decreases around 20% when linear elements are used on both meshes.

5. Conclusion

To further improve efficiency of adaptive finite element simulations we consider the usage of different adaptively refined meshes for different variables in systems of nonlinear, time-depended PDEs. The different variables can have very distinct solution behavior. To resolve this the meshes can be independently adapted for each variable. The multi-mesh method, as defined in this paper, can make use of Lagrange finite elements of arbitrary degree and is independent of the spatial dimension. The approach is well defined, and can be implemented in existing adaptive finite element codes with minimal effort. The additional computational effort for assembling matrices on virtual meshes is very small and can be negligible in most computations. Only small transformation matrices must be multiplied with the matrices assembled on mesh elements. As discussed in Section 3.5, the transformation matrices can be stored in an appropriate data structure to avoid unnecessary recalculations. We have demonstrated for various examples that the resulting linear systems are usually much smaller, when compared to the usage of a single mesh, and the overall computational runtime can be more than halved in various cases. Phase transition problems within a diffuse interface approach are well suited for such an approach. The same holds for saddle-point problems in which the inf-sup condition can be fulfilled for finite elements of the same order.

This work is the very first rigorous derivation of a multi-mesh method. In contrast to existing work [18–20], we have shown that using virtual meshes results in the same matrices when combining the two meshes physically to a union of both. Our approach is very simple to implement in existing finite element codes, as it does not relay on any special data structure. Furthermore, it is not restricted to linear finite elements, but generalizes to Lagrange finite elements of arbitrary degree. Though presented here for triangles/tetrahedrons and bisectioning, it can easily be modified to other elements and other refinement strategies.

Further examples, that may benefit from the multi-mesh method, include general diffuse interface concepts to solve PDEs in complex domain. Here a phase-field function is used to describe the domain implicitly [15], which only requires a fine resolution along the boundaries. The approach might also be used in time stepping schemes to prevent loss of information during coarsening. In a classical approach the solution from the old time step is simply interpolated to the new mesh at the new time step. If the new mesh is coarser information is lost, which can be prevented by using the multi-mesh approach for the solution at different time steps. This aspects has been considered in Dubcova et al. [6]. And also in optimal control problems the approach is very promising, as in many situations the dual solution is much smoother than the primal solution and thus can be discretized on a much coarser mesh. To demonstrate this for parabolic control problems work is in progress.

Acknowledgments

We would like to thank Rainer Backofen for fruitful discussions. The work has been supported by DFG through Vo899/5-1 and Vo899/11-1.

References

[1] S. Aland, J. Lowengrub, A. Voigt, Two-phase flow in complex geometries – a diffuse domain approach, Computer Modeling in Engineering & Sciences 57 (1) (2010) 77–108.
[2] M. Bercovier, O. Pironneau, Error estimates for finite element method solution of the Stokes problem in the primitive variables, Numerische Mathematik 33 (2) (1979) 211–224.
[3] W.J. Boettinger, J.A. Warren, C. Beckermann, A. Karma, Phase-field simulation of solidification, Annual Review of Materials Research 32 (2002).
[4] T.A. Davis, Algorithm 832: UMFPACK v4.3 – an unsymmetric-pattern multifrontal method, ACM Transactions on Mathematical Software 30 (2) (2004) 196–199.
[5] Y. Di, R. Li, Computation of dendritic growth with level set model using a multimesh adaptive finite element method, Journal of Scientific Computing 39 (3) (2009) 441–453.
[6] L. Dubcova, P. Solin, G. Hansen, H. Park, Comparison of multimesh hp-FEM to interpolation and projection methods for spatial coupling of thermal and neutron diffusion calculations, Journal of Computational Physics 230 (4) (2011) 1182–1197, http://dx.doi.org/10.1016/j.jcp.2010.10.034.
[7] U. Ghia, K.N. Ghia, C.T. Shin, High-Re solutions for incompressible flow using the Navier–Stokes equations and a multigrid method, Journal of Computational Physics 48 (3) (1982) 387–411.
[8] P. Gottschling, D.S. Wise, M.D. Adams, Representation-transparent matrix algorithms with scalable performance, in: ICS'07: Proceedings of the 21st Annual International Conference on Supercomputing, New York, NY, USA, ACM, 2007, pp. 116–125.
[9] X. Hu, R. Li, T. Tang, A multi-mesh adaptive finite element approximation to phase field models, Communications in Computational Physics 5 (2009) 1012–1029.
[10] W. Joppich, M. Kürschner, MpCCI – a tool for the simulation of coupled applications, Concurrency and Computation: Practice and Experience 10 (2005) 183–192.
[11] A. Karma, J.W. Rappel, Phase-field method for computationally efficient modeling of solidification with arbitrary interface kinetics, Physical Review E 53 (1996) R3017–R3020.
[12] A. Karma, W. Rappel, Quantitative phase-field modeling of dendritic growth in two and three dimensions, Physical Review E 57 (4) (1998) 4323–4349.
[13] C. Landsberg, F. Stenger, A. Deutsch, M. Gelinsky, A. Rösen-Wolff, A. Voigt, Chemotaxis of mesenchymal stem cells within 3D biomimetic scaffolds – a modeling approach, Journal of Biomechanics 44 (2) (2011) 359–364.
[14] R. Li, On multi-mesh H-adaptive methods, Journal of Scientific Computing 24 (3) (2005) 321–341.
[15] X. Li, J. Lowengrub, A. Rätz, A. Voigt, Solving PDEs in complex geometries, Communications in Mathematical Sciences 7 (2009) 81–107.
[16] N. Provatas, K. Elder, Phase-Field Methods in Materials Science and Engineering, Wiley, 2010.
[17] A. Rätz, A. Voigt, PDEs on surfaces – a diffuse interface approach, Communications in Mathematical Sciences 4 (2006) 575–590.
[18] A. Schmidt, A multi-mesh finite element method for phase field simulations, Lecture Notes in Computational Science and Engineering 32 (2003) 208–217.
[19] P. Solin, J. Cerveny, L. Dubcova, Adaptive multi-mesh hp-FEM for linear thermoelasticity, Research Report No. 2007-08, The University of Texas at El Paso, 2007.
[20] P. Solin, L. Dubcova, J. Kruis, Adaptive hp-FEM with dynamical meshes for transient heat and moisture transfer problems, Journal of Computational and Applied Mathematics 233 (12) (2010) 3103–3112.
[21] K.E. Teigen, A. Rätz, A. Voigt, A diffuse-interface method for two-phase flows with soluble surfactants, Journal of Computational Physics 230 (2) (2011) 375–393.
[22] S. Turek, Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach, Springer, 1999.
[23] R. Verfürt, A Review of a Posteriori Error Estimation and Adaptive Mesh-Refinement Techniques, Teubner, 1996.
[24] R. Verfürth, Error estimates for a mixed finite element approximation of the Stokes equations, RAIRO – Analyse Numèrique 18 (1984) 175–182.
[25] R. Verfürth, A posteriori error estimates for nonlinear problems. Finite element discretizations of elliptic equations, Mathematics of Computation 62 (206) (1994) 445–475.
[26] S. Vey, A. Voigt, AMDiS: adaptive multidimensional simulations, Computing and Visualization in Science 10 (1) (2007) 57–67.
[27] W.A. Wall, Fluid-Struktur-Interaktion mit stabilisierten Finiten Elementen, PhD Thesis, Institut für Baustatistik der Universität Stuttgart, 1999.

Axel Voigt received his doctoral degree in mathematics from Technische Universität München, Germany, in 2001. After a lecture position in the Mathematics Department at the University of Bonn and a group leader position at the research center caeasr in Bonn, Germany, he become full professor in 2007 at the Institut for Scientific Computing at Technische Universität Dresden, Germany. He works on numerical approaches for partial differential equations in various applications including image processing, material science and biophysics.

Thomas Witkowski is a Ph.D. student at the Institut for Scientific Computing, Technische Universität Dresden, since 2007. His research interests are in the area of numerical algorithms for the solution of PDEs, parallelization and high performance computing (HPC).