Comput Visual Sci (2007) 10:57–67
DOI 10.1007/s00791-006-0048-3

REGULAR ARTICLE

AMDiS: adaptive multidimensional simulations

Simon Vey · Axel Voigt

Received: 10 May 2005 / Accepted: 16 May 2006 / Published online: 14 December 2006

Abstract We describe how modular software design and well proven object oriented design patterns can help to implement a flexible software package for the efficient solution of partial differential equations. Today not only efficiency in the numerical solution is of utmost importance for practical use, efficiency in problem setup and interpretation of numerical results is of importance if modeling and computing comes closer and closer together. In order to demonstrate the possibilities of the software, we apply the tool to several non-standard problems.

1 Introduction

Scientific computation has become a major tool in conducting research, playing comparable roles as do experiment and theory. This success of computational methods in scientific and engineering research is next to the enormous improvement of computer hardware to a large extend due to contributions from applied mathematicians, who have developed algorithms which make real life applications feasible. Examples are adaptive methods, high order discretization, fast linear and nonlinear solvers and multi-level methods. The application of these methods in a large class of problems demands for suitable and robust tools for a flexible and efficient implementation. Today there exist several different simulation packages for the numerical solution of partial differential equations by the above mentioned algorithms, which are suitable to solve real world problems. For an example of an adaptive finite element software we refer to [25]. In order to play a crucial role in scientific and engineering research, besides efficiency in the numerical solution, efficiency in problem setup and interpretation of simulation results is of utmost importance too. As modeling and computing comes closer together efficient computational methods need to be applied to new sets of equations. The problems to be addressed by simulation methods become more and more complicated, ranging over different scales, interacting on different dimensions and combining different physics. Such problems need to be implemented in a short period of time, solved on complicated domains and visualized with respect to the demand of the user. Only a modular abstract simulation environment will fulfill these requirements and allow to setup, solve and visualize real-world problems appropriately. We would like to show, how abstract data structures and modern software concepts can help to design user-friendly finite element software which provide large flexibility in problem definition while on the other hand solve these problems efficiently. Attempts to modularize finite element codes have recently intensified, see e.g. [2,6]. The basic principles and advantages of modularization are commonly known and acknowledged. However a widely accepted modular design of finite element software is not yet developed. In this paper we motivate and describe the design of AMDiS and demonstrate its possibilities. AMDiS extends some of the mathematical concepts of the adaptive finite element C-library ALBERTA [25] and realizes them in a modular object

Communicated by K. Mikula.

S. Vey · A. Voigt (✉)
Crystal Growth group, Research Center caesar,
Ludwig-Erhard-Allee 2, 53175 Bonn, Germany
e-mail: voigt@caesar.de

S.Vey
e-mail: vey@caesar.de

oriented design. Besides a much more modern software design, which allows for a more flexible use of the software, the main additional feature is the extension of several concepts to systems of PDEs and coupling of problems over different dimensions, which is not possible in ALBERTA.

In Sect. 2 we show for a linear model problem of second order which part of the solution process is problem specific and what can be handled by general exchangeable library functions. Section 3 describes implementation aspects related to this separation. In Sect. 4 features of AMDiS are explained and in Sect. 5 several nonstandard applications are shown. Finally conclusions are drawn in Sect. 6.

2 Abstract FEM

We consider the following partial differential equation, which serves as a model problem and describes a general second order parabolic equation

$$u_t - \nabla \cdot A\nabla u + b \cdot \nabla u + cu = f \quad \text{in } \Omega, \text{ for } t > 0 \tag{1}$$
$$u = g \quad \text{on } \Gamma_D, \text{ for } t > 0 \tag{2}$$
$$\mathbf{n} \cdot A\nabla u = h \quad \text{on } \Gamma_N, \text{ for } t > 0 \tag{3}$$
$$u(\cdot, 0) = u_0 \quad \text{in } \Omega, \tag{4}$$

with problem specific parameters $A = A(u, \nabla u, \mathbf{x}, t)$, $b = b(u, \nabla u, \mathbf{x}, t)$ and $c = c(u, \nabla u, \mathbf{x}, t)$. The right hand side, Dirichlet and Neumann boundary values are given by $f = f(\mathbf{x}, t)$, $g = g(u, \nabla u, \mathbf{x}, t)$ and $h = h(u, \nabla u, \mathbf{x}, t)$. And the initial condition is given by $u_0 = u_0(\mathbf{x})$. For all these functions we assume appropriate regularity. By $\mathbf{n}$ we denote the unit normal on $\partial\Omega$. With $X = H^1(\Omega)$ and $Y = \{v \in X : v = 0 \text{ on } \Gamma_D\}$ the weak form reads

$$\int_\Omega u_t\phi + \int_\Omega A\nabla u \cdot \nabla\phi + \int_\Omega b \cdot \nabla u\phi + \int_\Omega cu\phi - \int_{\Gamma_N} h\phi = \int_\Omega f\phi \tag{5}$$

for all $\phi \in Y$. Now we split the time interval by discrete time instants $0 = t_0 < t_1 < \cdots$ and define the time steps $\tau^n = t_{n+1} - t_n$. We consider at time instant t_n finite dimensional subspaces $X^n_{h,i} \subset X$ with $N^n_i = \dim X^n_{h,i}$ and set $Y^n_{h,i} = X^n_{h,i} \cap Y$ with $M^n_i = \dim Y^n_{h,i}$. For simplicity lets apply an implicit Euler discretization in time. The discrete solution of (5) is then given by: Find $u^{n+1}_h \in X^{n+1}_{h,1}$ such that $u^{n+1}_h \in g_h + Y^{n+1}_{h,1}$ and

$$\int_\Omega \frac{u^{n+1}_h - u^n_h}{\tau^n}\phi_h + \int_\Omega A\nabla u^{n+1}_h \cdot \nabla\phi_h + \int_\Omega b \cdot \nabla u^{n+1}_h\phi_h + \int_\Omega cu^{n+1}_h\phi_h - \int_{\Gamma_N} h_h\phi_h = \int_\Omega f\phi_h \tag{6}$$

for all $\phi_h \in Y_{h,2}$, with g_h and h_h approximation of g and h, respectively. If we treat all dependencies of the parameters on u explicitly, the resulting equation becomes linear. Now choosing basis $\{\phi_1, \ldots, \phi_{N^n_1}\}$ and $\{\psi_1, \ldots, \psi_{N^n_2}\}$ of $X^n_{h,1}$ and $X^n_{h,2}$, such that $\{\phi_1, \ldots, \phi_{M^n_1}\}$ and $\{\psi_1, \ldots, \psi_{M^n_2}\}$ is a basis of $Y^n_{h,1}$ and $Y^n_{h,2}$, respectively. For $v^n_h \in X^n_{h_1}$ we denote by $V^n = (V^n_1, \ldots, V^n_{N^n_1})$ the coefficient vector of v^n_h, with respect to the basis $\{\phi_1, \ldots, \phi_{N^n_1}\}$, i.e.

$$v^n_h = \sum_{j=1}^{N^n_1} V^n_j\phi_j. \tag{7}$$

Using (6) with test functions ψ_i, $i = 1, \ldots, M^{n+1}_2$, we get the following equations for the coefficient vector U^{n+1} of u^{n+1}_h

$$\frac{1}{\tau^n}\sum_{j=1}^{N^{n+1}} U^{n+1}_j \int_\Omega \phi_j\psi_i + \sum_{j=1}^{N^{n+1}} U^{n+1}_j \int_\Omega A\nabla\phi_j \cdot \nabla\psi_i + \sum_{j=1}^{N^{n+1}} U^{n+1}_j \int_\Omega b \cdot \nabla\phi_j\psi_i + \sum_{j=1}^{N^{n+1}} U^{n+1}_j \int_\Omega c\phi_j\psi_i - \sum_{j=1}^{N^{n+1}} U^{n+1}_j \int_{\Gamma_N} \tilde{h}_h\phi_j\psi_i = \int_\Omega \tilde{f}\psi_i + \frac{1}{\tau^n}\sum_{j=1}^{N^n} U^n_j \int_\Omega \phi_j\psi_i$$

for $i = 1, \ldots, M^{n+1}_2$ and $U^{n+1}_i = G_i$, with G_i the coefficients of g_h, for $i = M^{n+1}_2 + 1, \ldots, N^{n+1}_2$. The functions $\tilde{h}_h$ and $\tilde{f}_h$ result from a linearization of h_h. The resulting system has to be assembled, which is done by looping over all grid elements and using quadrature formulas for each element to compute the integrals. If $\mathcal{T}^{n+1}$ is a triangulation of Ω with elements T^{n+1} the integrals read

$$\int_\Omega \phi_j\psi_i \, \mathrm{d}\mathbf{x} = \sum_{T^{n+1} \in \mathcal{T}^{n+1}} \int_{T^{n+1}} \phi_j\psi_i \, \mathrm{d}\mathbf{x}, \quad \ldots \tag{8}$$

where the sum can be restricted to $T^{n+1} \subset \text{supp } \phi_j \cap \text{supp } \psi_i$. Now instead of performing the integration on the elements T^{n+1}, a parameterization $F_T : \hat{T} \to T^{n+1}$ is used, with $\hat{T} = \text{conv hull}\{0, \mathbf{e}_1, \ldots, \mathbf{e}_d\}$ the *standard element* in $\mathbb{R}^d$. Furthermore the basis functions are defined on a *reference element* $\overline{T} = \{(\lambda_0, \ldots, \lambda_d) \in \mathbb{R}^{d+1}; \lambda_k \geq 0, \sum_{k=0}^d \lambda_k = 1\}$ using barycentric coordinates, see Fig. 1. These allow to rewrite the integrals as

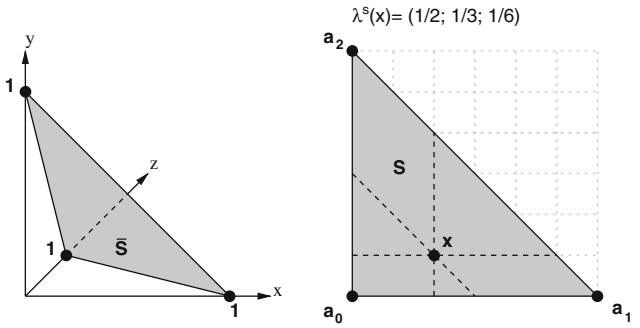

Fig. 1 2d reference simplex $\bar{S}$ and a point x in barycentric coordinates of an element S

$$\int_{T^{n+1}} A\nabla\phi_j \cdot \nabla\psi_i \,\mathrm{d}\mathbf{x} = \int_{\hat{T}} \bar{A}\nabla_\lambda\phi_j(\lambda(\hat{\mathbf{x}})) \cdot \nabla_\lambda\psi_i(\lambda(\hat{\mathbf{x}})) \,\mathrm{d}\hat{\mathbf{x}}$$

$$\int_{T^{n+1}} b \cdot \nabla\phi_j\psi_i \,\mathrm{d}\mathbf{x} = \int_{\hat{T}} \bar{b}\nabla_\lambda\phi_j(\lambda(\hat{\mathbf{x}})) \cdot \psi_i(\lambda(\hat{\mathbf{x}})) \,\mathrm{d}\hat{\mathbf{x}}$$

$$\int_{T^{n+1}} c\phi_j\psi_i \,\mathrm{d}\mathbf{x} = \int_{\hat{T}} \bar{c}\phi_j(\lambda(\hat{\mathbf{x}}))\psi_i(\lambda(\hat{\mathbf{x}})) \,\mathrm{d}\hat{\mathbf{x}}$$

and so on, with $\bar{A} = \Lambda(F_T(\hat{\mathbf{x}}))A\Lambda^t(F_T(\hat{\mathbf{x}}))|\det DF_T(\hat{\mathbf{x}})|$, $\bar{b} = \Lambda(F_T(\hat{\mathbf{x}}))b|\det DF_T(\hat{\mathbf{x}})|$, $\bar{c} = c|\det DF_T(\hat{\mathbf{x}})|$ and Λ the Jacobian of the barycentric coordinates on T^{n+1}. Barrycentric coordinates can be used with simplical elements only but have the advantage that basis functions have to be constructed and evaluated only once at the reference element. In this way we transform all integrations onto the *standard element* and have the definition of basis functions on the *reference element* at hand. The parameterization F_T is given by the coordinates of the mesh elements, thus everything which is needed to assemble the system is knowledge of the operators on the *standard element*, a triangulation and a set of basis functions on the *reference element*. The assembled system has then to be solved and the error of the numerically solution afterwards estimated. The error estimation can be performed with the data at hand. These procedure determine the core aspects of the implementation, which will be described in the next section.

3 Implementation

AMDiS is written as an object oriented library in the programming language C++, using a modular design and well proven object oriented design patterns. In Fig. 2 the main AMDiS components with the adaptation loop as highest abstraction level are shown. In the following sections these components and their implementation are explained in more detail.

3.1 Adaptation loop

The adaptation loop builds the highest abstraction level in the simulation. Here the decisions are made, when a problem has to assemble its equation system, when to solve the system, when to estimate the error indicators and when to adapt the underlying mesh. In order to keep the implementation of these single steps transparent at adaptation loop level, it is delegated to the problem classes, that implement an abstract problem interface. The problems are known to the adaptation loop only by this simple interface. This allows a very easy and straightforward formulation of the adaptation loop by using the problem as black box component, what in turn leads to a high reusability of the adaptation loop for different problem types. So the adaptation strategy, which can become complicated in time and space adaptive problems, can be used independently of problem implementations, and on the other hand problem implementations can be reused with different adaptation strategies. In AMDiS some adaptation strategies are implemented already but of course the adaptation loop can be user defined also. Beside the problem the current adaptation state, which is stored in an *AdaptInfo* object, must be known to the adaptation loop. Here information about the current simulation time, the current timestep size and the current iteration number are stored and whether the desired tolerances or the maximal iteration numbers are reached.

To illustrate the interactions between adaptation loops and problems, in Algorithm 1 we show an adaptation loop of a time dependent problem solved with an explicit time strategy. Before the time loop starts an initial solution must be calculated for the start time. After that in each timestep the problem is solved on the mesh of the last timestep and after that error indicators are estimated and the mesh will be adapted according to these indicators. This procedure is not iterated within one timestep.

Algorithm 1 Explicit time strategy

```
adaptInfo→time = adaptInfo→startTime;
adaptInitial→adaptationLoop();
problem→estimate(adaptInfo→timestep);
while adaptInfo→time < adaptInfo→endTime do
  adaptInfo→time += adaptInfo→timestep;
  problem→initTimestep();
  problem→setTime(adaptInfo→time);
  problem→buildAndAdapt();
  problem→solve();
  problem→estimate(adaptInfo→timestep);
  problem→closeTimestep();
end while
```

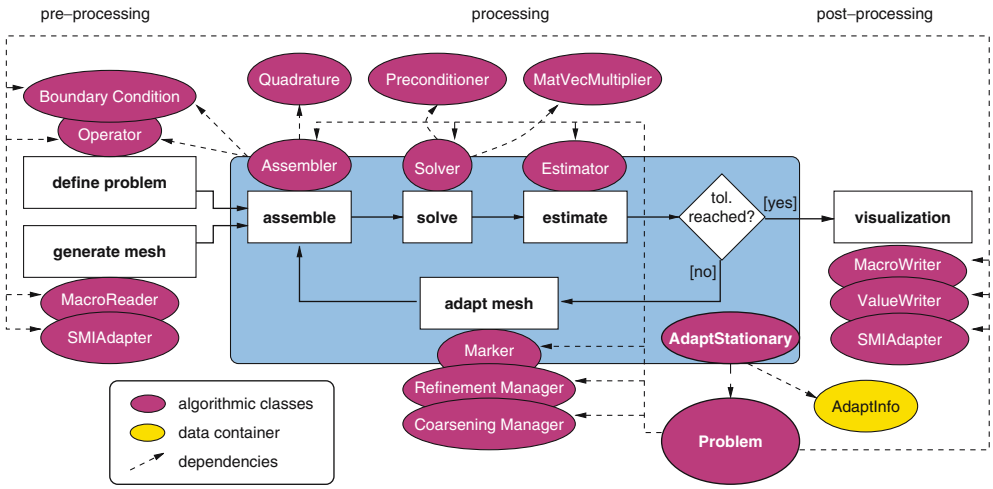

Fig. 2 Adaptation loop for a stationary problem and the needed software components

The problem method *buildAndAdapt()* is an abbreviation for a nested assemble and mesh adaptation strategy, where partial assembling before refinement, between refinement and coarsening, and after coarsening is allowed. Note that no assumptions about the problem implementation are made in the adaptation loop. The calculation of the initial solution is delegated to another adaptation loop, whose implementation also is transparent for the calling adaptation loop. In Algorithm 2 we show, how two stationary problems can be coupled at adaptation loop level. Problem two here needs the solution of problem one for the assemblage of its equation system. So first problem one must assemble and solve its system, before problem two can start the assemblage. To simplify matters, we assume that no mesh adaptation is necessary (even if the name *adaptation loop* then is somehow misleading) and that the problems do all their assembling in *buildAfterCoarsen()*.

Algorithm 2 Two coupled problems

```
problem1→buildAfterCoarsen();
problem1→solve();
problem2→buildAfterCoarsen();
problem2→solve();
```

To keep the adaptation loop independent of problem implementations, no knowledge about the problem dependencies is assumed here, but only an order of execution is defined. Therefore it must be assured outside the adaptation loop, that problem two can access the solution of problem one. This is not part of the abstract problem interface.

3.2 Finite element spaces

While at adaptation loop level no knowledge about the implementation is assumed, we now descent into deeper abstraction levels and describe the concrete data structures and algorithms which fill the single adaptation steps with life. The first step of a solid object oriented software design is a clean separation between data and algorithms. Linking those two too close together will produce strong algorithmic dependencies and reduce extensibility and maintainability of the software extremely. Fortunately finite element software decomposes in its components in a very natural way. All needed algorithms operate on one or more *finite element spaces*. Like mentioned in Sect. 2 a finite element space consists of a domain discretization (Sect. 3.2.1), local and global basis functions (Sect. 3.2.2), and degrees of freedom (DOFs) which build the connection between the local and the global basis functions (Sect. 3.2.3).

3.2.1 The hierarchical mesh

In AMDiS the domain is discretized by a mesh consisting of simplicial elements. The simplex is the simplest possible polytope in any given space (a line in 1d, a triangle in 2d, a tetrahedron in 3d). Choosing simplices as mesh elements allows to define local basis functions on a single reference element in barycentric coordinates (Sect. 3.2.2) and to implement simple mesh refinement by bisection (Sect. 3.6). The mesh is stored in a hierarchical way. In the preprocessing phase an initial triangulation (*macro triangulation*) consisting of *macro elements* is created. In these macro elements all needed topological and geometrical information are stored. Additionally to these information the reference to an *element*

object is stored which builds the root of a binary tree resulting from consecutive bisections of the element. Therefore each refined element holds the references to its two children. To avoid redundancy of element information and so to reduce the amount of needed memory, geometrical and topological data aren't stored at each element, but generated from the macro elements while mesh traversal only for the current element and stored in an *element info* object. The hierarchical storage of the mesh allows an easy coarsening and can be utilized for multi level methods. In order to keep the mesh implementation independent from the dimension the *prototype design pattern* is applied to the mesh (see [13]). So no knowledge about the elements is assumed in mesh implementation, but only a reference to an *element prototype* is stored. This prototype can deliver all needed information at runtime and is able to create a copy (*clone*) of itself. Thus new elements can be created during refinement. Sometimes it is necessary to store data directly at the elements because they can't be generated from macro data while mesh traversal. But not all elements may need these information and at some elements more than one information type has to be stored. Furthermore the data assignment to the elements may change during runtime. Therefore a flexible *decoration* of the elements is needed which is provided by the *decorator pattern* (see [13]). To decide what to do with the element data while refinement and coarsening a *chain of responsibility* is used, which allows element data specific implementations that are transparent to the mesh.

3.2.2 Basis functions

Using barycentric coordinates it is easy to construct local basis functions ϕ_i which hold the condition $\phi_i(\lambda_j) = \delta_{ij}$, $i,j \in \{1,\dots,N\}$ where N is the number of basis functions defined at the element and the jth basis function is located at barycentric coordinates λ_j. This condition ensures that the calculated coefficient of each basis function is equal to the finite element solution at the coordinates this function is located at. In AMDiS so far Lagrange elements up to degree four are implemented in one, two and three space dimensions.

3.2.3 Degrees of freedom

As described in Sect. 3.2.2 the basis functions are defined locally at the elements of the triangulation. To be able to assemble and solve the global equation system, each basis function must be also accessible by a global index. On each local position where basis functions can be located, a pointer to an array containing DOF indices is stored. In Fig. 3 two cubic Lagrange triangles, which

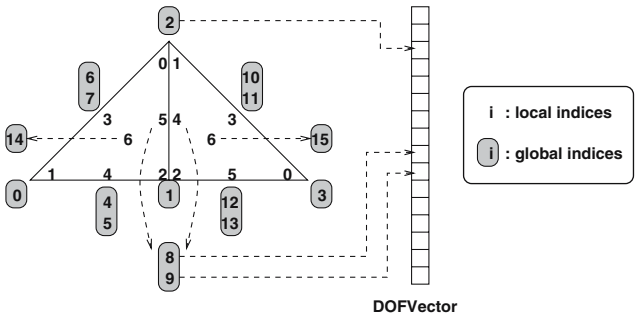

Fig. 3 Assignment between local and global DOF indices for cubic Lagrange elements in 2d

have one basis function located at each vertex and in the center and two basis functions located at each edge, are shown. At their common vertices and the common edge the elements share the same DOFs by pointing to the same arrays of indices which now can be used for indexing *DOF indexed* objects, which can be *DOF vectors* or *DOF matrices*. When the corresponding mesh is refined or coarsened, a central *DOF administration* dynamically adapts the size of each *DOF indexed* object and generates new DOFs or removes DOFs, that are not longer needed. The removal of DOFs can lead to holes in the sequence of valid DOF indices, which in turn can lead to unused entries in *DOF vectors* and *DOF matrices*. Therefore *DOF iterators* can be configured to iterate over all used DOFs, all free DOFs or over every DOF of a *DOF indexed* object.

3.3 Assembling

Now we describe, how the matrix and the right hand side vector of the linear equation system is build in the assembling step. If we consider a single equation the procedure is already described in Sect. 2, we need only one mesh and one finite element space. In coupled problems it may be, that different finite element spaces are used for the different components and that they might live on different meshes. So when assembling is done for a term, that couples two components, this must be taken into consideration. We describe the contribution of a second order term, that couples finite element space $X^n_{h,1}$ with finite element space $X^n_{h,2}$, to the entry (j,i) of the element matrix of an element T_1 of finite element space $X^n_{h,1}$

$$\sum_{T_2 \in \operatorname{supp} T_1} \int_{\hat{T}} \bar{A} \nabla_{\lambda_1} \phi_j(\lambda_1(\hat{x})) \cdot \nabla_{\lambda_2} \psi_i(\lambda_2(\hat{x})) \, \mathrm{d}\hat{x}$$

with $\bar{A} = \Lambda_{T_1}(F_{T_1}(\hat{x})) A \Lambda^t_{T_2}(F_{T_2}(\hat{x}))| \det \mathrm{DF}_{T_1}(\hat{x})|$. If both finite element spaces share the same mesh the sum

disappears and $T_2 = T_1, \lambda_2 = \lambda_1$ and $\Lambda_{T_2} = \Lambda_{T_1}$. If even the same basis functions are applied, the finite element spaces are equivalent and $\phi_j = \psi_j$. Note that in this formulation only the matrix A is problem dependent. Therefore in AMDiS a separation between *operators* and *assemblers* is done. Each operator consists of *operator terms* which can be of different order. A *second order operator term* e.g. calculates the term $\Lambda_{T_1}(x) A \Lambda_{T_2}{}^t(x)$. The fact that not only A but this whole product is defined by the operator term, allows to consider special properties and dependencies of A in a more efficient way. The integral now is computed by an *assembler* consisting of up to three *sub assemblers* for the different order terms using numerical quadrature. Because the integration is done at the standard simplex $\hat{T}$, basis functions often have to be evaluated at the same coordinates. Therefore the values of the basis functions are calculated only once at each quadrature point and stored in *fast quadrature* objects. To keep the shape of element matrices and vectors transparent to other program components it is encapsulated in *element matrix* and *element vector* objects.

3.4 Solvers

After the equation system is assembled, the finite element solution can be calculated by solving this system. Therefore different iterative methods with different demands on the equation system are available. To solve the system, a right hand side vector, an initial guess for the solution and a *matrix–vector multiplier* must be given to the solver. The vector types are specified by a template parameter of the solver class. Thereby the solver implementations are independent of the vector type. This is used for the solution of vector valued systems where a vector consists of multiple DOF vectors (Sect. 3.7). The matrix is not directly known to the solver, but it is encapsulated within the matrix–vector multiplier, which performs the matrix–vector multiplication for an arbitrary vector and returns the result vector. So the multiplication can be specified without changing the solver code. Furthermore several *preconditioners* are provided and for the solution of nonlinear equation systems several Newton methods are available.

3.5 Estimators

Error estimators build the foundation for mesh adaptivity. On the one hand a global error indicator determines whether the actual solution has reached the wanted accuracy and therefore the adaptation loop can be finished. On the other hand local indicators are used to decide, which elements have to be refined and which to be coarsened. The fact that problem formulations are encapsulated in *operator* objects enables a general estimator implementation for arbitrary operators. So estimators can be generated, at least for linear problems, automatically for given operators. Estimators for vector valued problems can be derived automatically from corresponding scalar estimators. In AMDiS so far residual based error estimators and recovery estimators [1,30] are available.

3.6 Mesh adaptation

Mesh adaptation can be divided into three steps: element marking, mesh refinement and mesh coarsening. Following the *visitor design pattern* (see [13]) each of these tasks is delegated to specialized components (*marker, refinement manager, coarsening manager*), which are accessible by abstract interfaces. This separation between data and algorithmic classes increases the flexibility and maintainability of the software by reducing algorithmic dependencies. In the marking step the decision is made, which elements have to be refined or coarsened depending on the local indicators calculated in the estimation step. Here several marking strategies are available. Mesh refinement is done by bisection of the simplicial elements by the refinement manager, coarsening is done by removing the children of an element by the coarsening manager.

3.7 Systems of coupled PDEs

In AMDiS systems of coupled PDEs can be solved in one equation system. To illustrate this we use model problem

$$-\Delta u = f$$
$$u - v = 0$$

To define this problem we formulate a matrix of operators for the left hand side and a vector of operators for the right hand side of the equation system which results in the following equation system after assemblage

$$\begin{pmatrix} L_{-\Delta u} & 0 \\ L_u & L_{-v} \end{pmatrix} \begin{pmatrix} u_h \\ v_h \end{pmatrix} = \begin{pmatrix} f_h \\ 0 \end{pmatrix}.$$

This gives an equation system consisting of a matrix of DOF matrices and two vectors of DOF vectors. In the general case of l coupled PDEs we have a matrix of $l \times l$ operators where the entry (i, j) contains the operator which couples the ith equation of the system with the jth variable. If equation i and equation j live on the same finite element space the assemblage can be done in the standard way described in Sect. 3.3. If the mesh is the

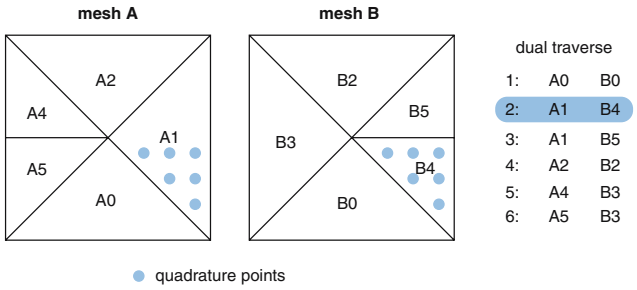

Fig. 4 Dual traverse of two independently refined meshes

same but the basis functions differ, the finite element space of equation *i* must be considered as *row space* and that of equation *j* as *column space* in the assembling routines (see Sect. 3.3). If even different meshes are used for the different components, for one element T of the mesh of component *i* all elements of the *j*th components mesh must be considered, that have an overlap with element T.

Similar to the method described in [24] in AMDiS meshes of different components share one initial triangulation but can be adapted independently of each other. The assemblage then is done within a parallel traverse of the two involved meshes (*dual traverse*), where each traverse step returns one element of each mesh. If a leaf element of one mesh has further refinements in the other mesh, it is returned in several dual traverse steps, until all corresponding leaf elements of the other mesh were traversed as well. The integration now is done over the smaller element using a parameterized quadrature for the basis functions of the bigger one. Figure 4 shows an example of a dual traverse for two triangular meshes. The resulting linear equation system now is solved by one of the iterative solvers described in Sect. 3.4, with *system vector* (vector of DOF vectors) as vector type and a specialized matrix–vector multiplication. Error estimation and element marking can be deduced from the scalar case in a general way.

4 Additional features

4.1 Parametric elements

Parametric elements allow to solve problems on arbitrary manifolds and to implement moving meshes. As mentioned above, geometrical information are not stored in the elements but generated during mesh traversal and stored in an *element info* object for the current traverse element. To be able to parameterize its elements a mesh can hold a *parametric* object which will get every *element info* object before it is returned

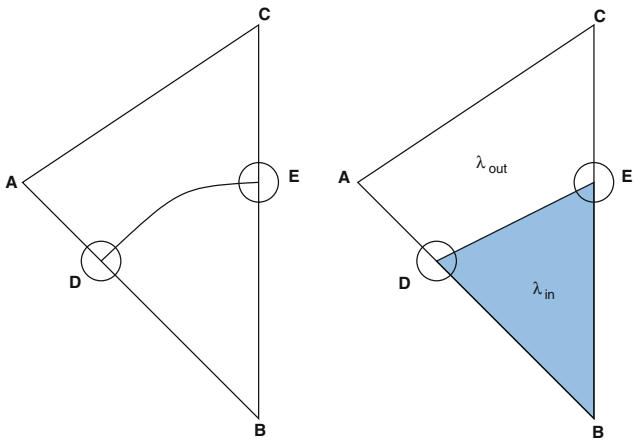

Fig. 5 Element T, boundary $\mathcal{M}_0$, and definition of λ

to the user. This parametric object know parameterizes the element information by changing its variable state or it can return a completely new object, derived from the element information class, which e.g. has the knowledge, how to transform world coordinates in barycentric coordinates and vice versa. This is necessary, if the parameterization is not affine. In both cases the parameterization is transparent for the user, who receives element information and doesn't have to care whether they are parameterized.

4.2 Composite finite elements

Composite finite elements provide the ability to circumvent the meshing of complex domains [14]. We use a *signed distance function* $d(x,y,z)$ which is negative inside the region, to describe our domain. The numerical grid now is always generated from a regular tetrahedral grid in a larger box containing the domain, which is adaptively refined according to $d(x,y,z)$. The signed distance function can be given analytically, be computed for implicitly given boundaries, or provided in a discrete form by values on the grid, which is common in *level set* applications.

The geometry is adaptively resolved with increasing refinement, but generally at no level can be represented by the nodes of the grid. If parameters vary across the boundary, integrals of the form $\int_T \lambda\phi$ with λ a discontinuous function have to be calculated. The method used is explained in Fig. 5 for 2d

$$\int_T \lambda\,\phi \approx \int_{\triangle(DBE)} \lambda_{\text{in}}\phi + \int_{\square(ADEC)} \lambda_{\text{out}}\phi$$
$$= \int_{\triangle(DBE)} \lambda_{\text{in}}\phi + \int_T \lambda_{\text{out}}\phi - \int_{\triangle(DBE)} \lambda_{\text{out}}\phi.$$

To enforce boundary conditions at the domain boundary a penalty method is used. For Dirichlet boundary condition $u = g$ the penalty term reads

$$\frac{1}{\epsilon(h)} \int_{\partial\Omega} (g - u_h)\phi \, \mathrm{d}\mathbf{x},$$

whereas for Neumann boundary conditions $\mathbf{n} \cdot A\nabla u = h$

$$\frac{1}{\epsilon(h)} \int_{\partial\Omega} y_h \mathbf{n} \cdot \nabla\phi \, \mathrm{d}\mathbf{x}, \quad \Delta_S y_h = \mathbf{n} \cdot A\nabla u_h - h \quad \text{on } \partial\Omega$$

is used to ensure the needed regularity. Δ_S is the surface Laplacian. Following [7] the additional equation on $\partial\Omega$ can be transformed into an equation on Ω.

4.3 Shared mesh interface (SMI)

We developed SMI to provide an unified and distributed management for arbitrary meshes. It can be used to share one ore more meshes between different programs. In AMDiS SMI is used to couple the simulation with a specialized visualization software in the post-processing step and to allow the integration of external mesh generators in the pre-processing step. Furthermore SMI can be used to turn the hierarchical mesh structure of AMDiS into a flat representation of the mesh, which allows a flexible iteration over elements and nodes, what is useful for some algorithms. On the one hand SMI provides an abstract interface which can handle any kind of unstructured meshes consisting of arbitrary and even mixed element types, on the other hand the client–server architecture of SMI allows to share meshes between different application running at even different computers. Figure 6 illustrates this client–server structure.

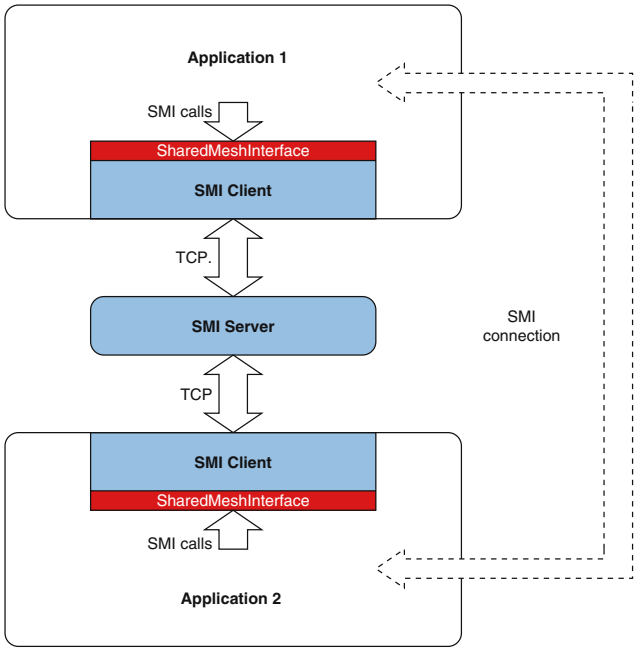

Fig. 6 Client-server architecture of SMI

5 Applications

Here we do not intend to solve a specific application problem. For that purpose, especially for problems related to materials science we refer to [3,16,22]. In order to demonstrate the possibilities of AMDiS, we rather work on an artificial problems. Our object of interest is the Stanford Bunny. This bunny model is most commonly used in testing computer graphics techniques and it is complicated enough as a test object for non-standard numerical simulations. With the techniques developed in [29] a polygonal mesh with 69.451 triangles of a bunny surface was created which after some modifications is here used as a macro triangulation for computations on parametrically defined domains. Besides this polygonal mesh the bunny is also described implicitly by a signed distance function and used as a computational domain to demonstrate the applicability of the level set techniques.

5.1 Higher order models on polygonal meshes

Applications for higher order partial differential equations on surfaces are abundant. For some special problems related to materials science, biology and image processing we refer to [15,19,20]. Here we will concentrate on a classical model for spinodal decomposition of a binary alloy, the Cahn–Hilliard equation. The model is applicable to describe coarsening dynamics in phase separation processes, which occur in quenched alloys. For numerical approaches we refer to [5,10,18]. We here solve such an equation on a general surface S. The equation reads

$$u_t = \Delta_S \left(-\epsilon \Delta_S u + \epsilon^{-1} G'(u) \right) \tag{9}$$

with Δ_S the surface Laplacian, $G(u) = 18u^2(1-u^2)$ a double well potential and ϵ a small parameter. $u = 0$ and $u = 1$ are the two stable steady states, representing the two phases. As initial conditions we use a small zero mean perturbation of $u = 0.5$. Our numerical approach is the same as for Euclidean geometries, see [22]. We write (9) as a system of two second order equations, discretize in space by linear finite elements, linearize the derivative of the double-well potential, apply a semi-implicit time–discretization and solve the resulting

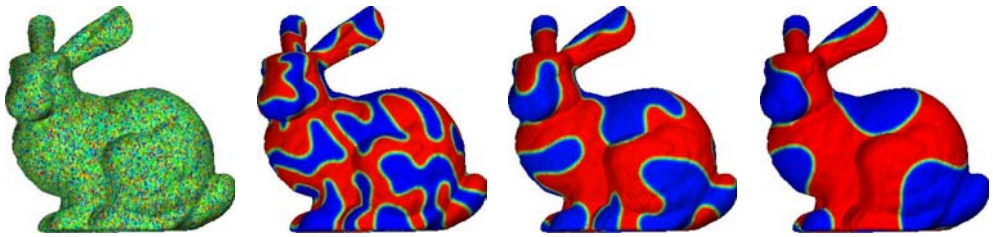

Fig. 7 Coarsening in the Cahn–Hilliard equation. Initial condition $u = 0.5$ (slightly perturbed), blue denotes $u = 0$ and red denotes $u = 1$. The time steps are $t = 0, 0.001, 0.00463, 0.01163$. The simulations are performed by A. Rätz

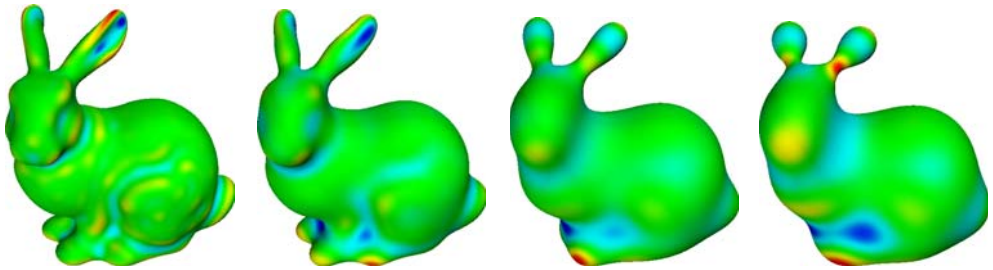

Fig. 8 Surface evolution due to surface diffusion. The time steps are $t = 10^{-7}, 10^{-6}, 5 \cdot 10^{-6}, 10^{-5}$. Red indicates a normal velocity inwards and *blue* outwards. The simulations are performed by F. Haußer

linear system by an iterative solver. The finite element representation reads

$$\int_S \frac{u_h^{n+1} - u_h^n}{\tau^n} \phi = \int_S \nabla_S w_h^{n+1} \cdot \nabla_S \phi \tag{10}$$

$$\int_S w_h^{n+1} \phi - \epsilon \int_S \nabla_S u_h^{n+1} \cdot \nabla_S \phi - \epsilon^{-1} \int_S G''(u_h^n) u_h^{n+1} \phi = \epsilon^{-1} \int_S (G'(u_h^n) - G''(u_h^n) u_h^n \phi) \tag{11}$$

with ϕ test functions from the space of piecewise linear, globally continuous elements. To solve this problem in AMDiS an implementation on an Euclidean grid can be used. Their are no changes in the code needed, only the parametric mesh has to be provided. Figure 7 shows the evolution of u at different time steps. The solution quickly separates the surface S into two regions S_0 and S_1, where u takes the values of 0 and 1, respectively. The remaining part of S lies on an interface of width $O(\epsilon)$ between the two regions. In later stages S_0 and S_1 change shape so that the length of the interface between the two regions decreases while maintaining the area of S_0 and S_1.

5.2 Geometric evolution by parametric finite elements

The evolution of surfaces plays a major role in various applications, such as fluid dynamics, materials science and image processing [11,23], and become even more important if smaller and smaller length scales are reached. Here we will concentrate on surface diffusion, as an important mass transport mechanism in epitaxial growth.

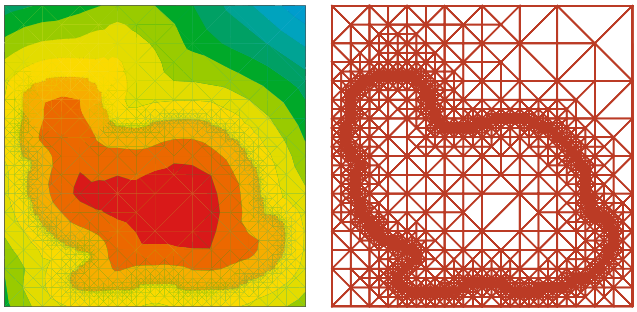

Fig. 9 Signed distance function and adaptive mesh on cut through the bunny

$$v = -\Delta_S \kappa \tag{12}$$

with v the normal velocity and κ the curvature. For numerical approaches we refer to [4] which we closely follow. We rewrite the forth order equation into a system of second order equations, discretize in space by linear parametric elements and semi-implicitly in time by treating the nonlinear operators and the surface normal explicitly but all other quantities implicitly. The resulting system for the vector and scalar valued unknowns

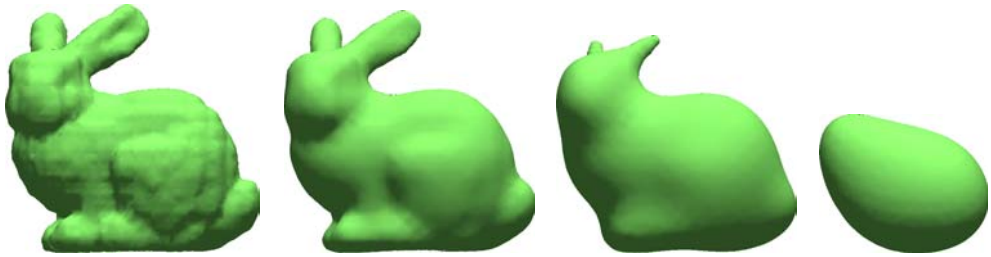

Fig. 10 Surface evolution due to mean curvature flow. The time steps are $t = 0, 0.006, 0.030, 0.200$. The simulations are performed by C. Stöcker

$\boldsymbol{\kappa} = \boldsymbol{\kappa}_h^{n+1}, \kappa = \kappa_h^{n+1}, \mathbf{v} = \mathbf{v}_h^{n+1}$ and $v = v_h^{n+1}$ reads

$$\int_{S^n} \boldsymbol{\kappa}\boldsymbol{\phi} - \tau^n \int_{S^n} \nabla_{S^n}\mathbf{v} \cdot \nabla_{S^n}\boldsymbol{\phi} = \int_{S^n} \nabla_{S^n}\mathbf{x}^n \cdot \nabla_{S^n}\boldsymbol{\phi} \tag{13}$$

$$\int_{S^n} \kappa\boldsymbol{\phi} = \int_{S^n} \kappa \cdot \mathbf{n}^n\phi \tag{14}$$

$$\int_{S^n} v\phi = \int_{S^n} \nabla_{S^n}\kappa \cdot \nabla_{S^n}\phi \tag{15}$$

$$\int_{S^n} \mathbf{v}\boldsymbol{\phi} = \int_{S^n} v\mathbf{n}^n\boldsymbol{\phi} \tag{16}$$

with ϕ and $\boldsymbol{\phi}$ scalar and vector valued piecewise linear test functions. The first equation results from the geometric expression $\boldsymbol{\kappa}^{n+1} = -\Delta_S\mathbf{x}^{n+1}$, with $\mathbf{x}^{n+1} = \mathbf{x}^n + \tau^n\mathbf{v}^{n+1}$ the updated position vector. The resulting linear system is solved by a Schur-complement ansatz. The main difference to the implementation of (10) and (11) lies in the fact, that the parameterization of the surface changes in time, which is accounted for in treating the position vector $\mathbf{x}^n$ as an unknown. For further numerical details and an extension to anisotropic situations we refer to [17]. Figure 8 shows the evolution of the surface at different time steps. The normal velocity is plotted on the surface to visualize the smoothing properties in detail. The evolution quickly smoothes small surface features and afterwards evolves towards its equilibrium shape. The configuration of the bunny however will lead to a pinch-off off the left ear, which cannot be handled within the described method. For the demonstrated isotropic situation the surface area decreases, while keeping the volume constant.

5.3 Implicit description of surfaces

In several applications, today mainly related to volumetric medical imaging, surfaces are not given in parametric form. They are only defined through implicit functions. On the other hand due to the success of level set methods [21,26] in a wide range of applications, including computer vision, fluid dynamics, optimal design, and others where the simulation of moving interfaces plays a key role, an implicit description of domains and the solution of partial differential equations on such implicitly described domains is of importance. In constructing signed distance functions, as well as in solving problems by level set methods, efficient numerical methods of Hamilton-Jacobi equations are necessary. To derive such methods on unstructured meshes we follow a proposed finite-element discretization of [8]. Here a linear finite element solution is constructed through a simplified local equation, which is solved by the Hopf–Lax formula. The proposed adaptive Gauss–Seidel iteration for the solution of the nonlinear system is modified and extended to three dimensional situations in [28] and used here to construct a signed distance function of the Stanford bunny from a given implicit representation. Figure 9 shows on a cut through the bunny, the computed signed distance function and the adaptively refined mesh. The computed data is used in the following sections as an initial description of the surface.

5.4 Geometric evolution by level sets

Again we are concerned with surface evolution, but now concentrate on mean curvature flow, important for example to describe the motion of grain boundaries. For an isotropic situation the equation has the simple form

$$v = \kappa \tag{17}$$

We solve this equation within a level set context. For related work we refer to [9,12,27]. Starting from the L^2-gradient flow of the surface energy $e(S_0) = \int_{S_0} 1ds$ and representing the surface $S_c = \{\mathbf{x} \in \Omega | u(\mathbf{x}) = c\}$ through the level set of u with value c, we can define a global energy $\mathcal{E}(u) = \int_R e(S_c)\mathrm{d}c = \int_\Omega \|\nabla u\|\mathrm{d}\mathbf{x}$. Following [9] this can be used to derive a finite element formulation for isotropic mean curvature flow, which in a semi-implicit time discretization reads

$$\int_\Omega \frac{u_h^{n+1} - u_h^n}{\tau^n} \frac{1}{||\nabla u_h^n||} \phi = \int_\Omega \frac{\nabla u_h^{n+1}}{\|\nabla u_h^n\|} \cdot \nabla \phi \tag{18}$$

with ϕ test functions from the space of piecewise linear, globally continuous elements. For further numerical details and an extension to anisotropic situations we refer to [9]. Figure 10 shows the evolution of the surface at different time steps. Again the evolution quickly smoothes small surface features and afterwards evolves towards its equlibrium shape, while the volume is shrinking. As expected the time scale is different than for the surface diffusion case in Sect. 5.2.

6 Conclusion

We described the software concepts of the adaptive finite element toolbox AMDiS and applied the toolbox to several non-standard problems. AMDiS is freely available for research and teaching purposes and can be downloaded at *http://www.caesar.de/cg*.

Acknowledgments We would like to thank J. Greer and G. Sapiro for providing an implicit representation of the bunny and M. Droske and M. Rumpf for helpful discussions.

References

1. Ainsworth, M., Oden, J.: A posteriori error estimation in finite element analysis. Wiley, New York (2000)
2. Banas, K.: On a modular architecture for finite element systems. i sequential codes. Comput. Vis. Sci. **8**, 35–47 (2005)
3. Bänsch, E., Haußer, F., Lakkis, O., Li, B., Voigt, A.: Finite element method for epitaxial growth with attachment-etachment kinetics. J. Comput. Phys. **194**, 409–434 (2004)
4. Bänsch, E., Morin, P., Nochetto, R.: A finite element method for surface diffusion: the parametric case. J. Comput. Phys. **203**, 321–343 (2005)
5. Barrett, J., Blowey, J.: Finite element approximation of the Cahn-Hilliard equation with concentration dependent mobility. Math. Comp. **68**, 487–517 (1999)
6. Bastian, P., Droske, M., Engwer, C., Klöfkorn, R., Neubauer, T., Ohlberger, M., Rumpf, M.: Towards a Unified Framework for Scientific Computing. LNCSE. Springer, Berlin Heidelberg New York (2005, Accepted for publication)
7. Bertalmio, M., Cheng, L.T., Osher, S., Sapiro, G.: Variational problems and partial differential equations on implicit surfaces. J. Comput. Phys. **174**, 759–780 (2001)
8. Bornemann, F., Rasch, C.: Finite-element discretization of static Hamilton–Jacobi equations based on a local variational principle. Comput. Vis. Sci. **9**, 57–69 (2006)
9. Clarenz U., Haußer F., Rumpf M., Voigt A., Weickard U. (2006) On level set formulations for anisotropic mean curvature flow and surface diffusion. In: Multiscale Modeling in epitaxial growth, ISNM, vol. 149, pp. 227–237. Birkhäuser, Boston (2005)
10. Feng, X., Prohl, A.: Numerical analysis of the Cahn–Hilliard equation and approximation for the Hele–Shaw problem. Interf. Free Bound. **7**(1), 1–28 (2005)
11. Fried, E., Gurtin, M.: A unified treatment of evolving interfaces accounting for small deformations and atomic transport with emphasis on grain-boundaries and epitaxy. Adv. Appl. Mech. **40**, 1–177 (2004)
12. Fried, M.: A level set based finite element algorithm for the simulation of dendritic growth. Comput. Vis. Sci. **7**(2), 97–110 (2004)
13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Addison-Wesley, Reading (1996)
14. Hackbusch, W., Sauter, S.: Composite finite elements for the approximation of PDEs on domains with complicated microstructures. Numer. Math. **75**, 447–472 (1997)
15. Halpern, D., Jensen, O., Grotberg, J.: A theoretical study of surfactant and liquid delivery into the lung. J. Appl. Physiol. **85**, 333–352 (1998)
16. Haußer, F., Voigt, A.: Facet formation and coarsening modeled by a geometric evolution law for epitaxial growth. J. Cryst. Growth **275**, e47–e51 (2005)
17. Haußer, F., Voigt, A.: Anisotropic surface diffusion, a numerical approach by parametric finite elements. J. Sci. Comput. (2007) DOI 10.1007/s10915-005-9064-6
18. Kim, J., Kang, K., Lowengrub, J.: Conservative multigrid methods for Cahn–Hilliard fluids. J. Comput. Phys. **193**(2), 511–543 (2004)
19. Memoli, F., Sapiro, G., Thompson, P.: Implicit brain imaging. Human Brain Mapping **23**, 179–188 (2004)
20. Myers, T., Charpin, J.: A mathematical model for atmosheric ice accretion and water flow on a cold surface. Int. J. Heat Mass Trans. **47**(25), 5483–5500 (2004)
21. Osher, S., Fedkiw, R.: Level Set Methods and Dynamic Implicit Surfaces. Springer, Berlin Heidelberg New York (2003)
22. Rätz, A., Ribalta, A., Voigt, A.: Surface evolution of elastically stressed films under deposition. J. Comput. Phys. **214**, 187–208 (2006)
23. Sapiro, G.: Geometric partial differential equations and image analysis. Cambridge University Press, Cambridge (2001)
24. Schmidt, A.: A multi-mesh finite element method for phase-field simulations. In: Interface and Transport Dynamics, LNCSE, vol. 32, pp. 209–217 (2003)
25. Schmidt, A., Siebert, K.: Design of Adaptive Finite Element Software, LNCSE, vol. 42. Springer, Berlin Heidelberg New York (2005)
26. Sethian, J.: Level Set Methods and Fast Marching Methods. Cambridge University Press, Cambridge (1999)
27. Smereka, P.: A level set based finite element algorithm for the simulation of dendritic growth. J. Sci. Comput. **19**, 439–456 (2003)
28. Stöcker, C., Vey, S., Voigt, A.: AMDiS-adaptive multidimensional simulation: composite finite elements and signed distance functions. WSEAS Trans. Circ. Syst. **4**, 111–116 (2005)
29. Turk, G., Levoy, M.: Zipped polygon meshes from range images. In: SIGGRAPH'94, pp. 311–318 (1994)
30. Verfürth, R.: A review of a posteriori error estimation and adaptive mesh refinement techniques. Wiley-Teubner, Chichester (1996)