Numerical Analysis and Practical Aspects of a Robust and Efficient Grid Deformation Method in the Finite Element Context

Matthias Grajewski*[†] Michael Köster[‡] Susanne Kilian[§] Stefan Turek [¶]

August 31, 2005

Abstract

Among a variety of grid deformation methods, the method proposed by Liao [4, 7, 17] seems to be one of the most favourables. In this article, we introduce a generalisation of Liao's method which allows for generating a desired mesh size distribution for quite arbitrary grids without giving rise to mesh tangling. Furthermore, we tackle the efficient implementation of the underlying steps and we discuss accuracy aspects like convergence behaviour as well as the robustness of the presented technique. The benefits of our new method are demonstrated for several applications and examples.

Keywords: mesh generation, deformation method, a posteriori error estimation, mesh adaption, fictitious boundary method

AMS classification: 65N15, 65N30, 76D05, 76D55

1 Introduction

Grid deformation, i.e. the redistribution of the mesh points of a given grid preserving the mesh topology, is a topic of research since many years and, consequently, a variety of methods is available today (see [5, 8, 10, 26] among many others). Most of the grid deformation approaches can be divided into two groups: the group of *static* methods and the group of *dynamic* approaches. Static methods obtain the mapping to the deformed grid by minimising certain functionals which usually leads to the (often difficult and expensive) solution of non-linear PDEs. In contrast to this, dynamic methods (e.g. [9, 16]) use time stepping or pseudo-time stepping approaches to construct the desired

*corresponding author

[†]Institute of Applied Mathematics, University of Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany, `Matthias.Grajewski@mathematik.uni-dortmund.de`

[‡]Institute of Applied Mathematics, University of Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany, `Michael.Koester@mathematik.uni-dortmund.de`

[§]Department of Mathematics, Humboldt-Universtität zu Berlin, Unter den Linden 6, D-10099 Berlin, Germany, `kilian@math.hu-berlin.de`

[¶]Institute of Applied Mathematics, University of Dortmund, Vogelpothsweg 87, D-44227 Dortmund, Germany, `Stefan.Turek@mathematik.uni-dortmund.de`

transformation. Being a member of the latter group, the method developed by Liao and his coworkers, based on the work of Moser [11], requires the solution of a single Poisson equation and a decoupled system of ODEs only. Furthermore, it has been shown that tangled elements cannot occur [19].

There are a couple of reasons to investigate and apply grid deformation in order to adapt a given computational mesh (*r-adaptivity*):

A numerical simulation using FEM starts with creating a computational grid. In the case of 2D, this process can be automatised by available grid generation software, but in the more interesting and demanding case of a full 3D simulation, the problem of efficient and robust grid generation is not solved in a satisfying manner yet. Consequently, 3D grid generation is often one of the most time consuming parts of an FEM simulation. In this context, fictitious boundary methods seem especially advantageous [22, 25]. Using such a method, it is not necessary anymore to generate a computational grid whose geometry is exactly adjusted to the physical boundary, i.e. the boundary of the simulated object. Instead, one can start with a non-adapted (semi-) structured mesh, because the physical boundary is implicitly enforced during the calculation. Unfortunately, the boundary approximation in these methods is of first order only. If the grid is adjusted and concentrated at the physical boundary by a grid deformation process, the boundary approximation error can be significantly decreased.

In many applications, local and anisotropic phenomena like shock fronts occur. It was shown [1, 12, 13] that by anisotropic refinement and alignment according to such phenomena, the accuracy of the calculation can be vastly improved. Refining the region of a shock by using hanging nodes solely (*h-adaptivity*) may suffer from the fact that the given grid is not aligned with the shock. Therefore, anisotropic refinement in such a situation requires either remeshing or grid deformation.

In FEM simulation it has turned out that grid adaptivity governed by a posteriori error estimation is mandatory for reliable and efficient computations. The widely used method of grid adaptation by allowing hanging nodes on element level, however, has severe impact on the speed of computation. Recent research [2] shows that in typical adaptive FEM codes using this method of grid adaptation only a small fraction of the available processor performance of several GFlop/s can be typically used. One of the reasons for this behaviour is the extensive usage of indirect adressing in such codes which is necessary to handle the unstructured grids emerging from the adaptation procedure. On the other hand, by using local generalised tensor product meshes and thereby avoiding indirect adressing a very significant speed up can be achieved. This has been successfully implemented in our new FEM package FEAST [2]. In this context, grid deformation is an ideal tool to grant the geometric flexibility necessary for the grid adaptation process according to a posteriori error estimators while maintaining logical tensor product structures of the grid.

In this article, we proceed as follows: In the next section, we describe in detail our deformation method and its numerical realisation. In section 3, the focus is placed on the numerical analysis of the grid deformation method. This includes robustness and accuracy considerations. The application of the deformation method to more sophisticated simulations is tackled in section 4.

2 Description of our Deformation Method

We first introduce some notations. A computational domain $\Omega \subset \mathbb{R}^2$ is triangulated by a mesh $\mathbb{T}$ consisting of N quadrilateral elements T of size h_T. The mesh is supposed to be conforming, i.e. no hanging nodes are allowed. The area of an element T is denoted by $m(T)$. For the common Lebesgue and Sobolev spaces on a domain D we use the abbreviations $L^2(D)$ and $H^k(D)$. In the special case $D = \Omega$, we write L^2 and H^k instead. The function space of k-fold continuously differentiable functions is referenced by $\mathcal{C}^k(D)$; for an interval I, $\mathcal{C}^{k,\alpha}(I)$, $0 < \alpha < 1$, denotes the space of functions with Hölder-continuous kth derivatives. A domain is said to have an $\mathcal{C}^{k,\alpha}$-smooth boundary if the boundary can be parameterised by a function in $\mathcal{C}^{k,\alpha}(I)$. The Jacobian matrix of a smooth mapping $\Phi : \Omega \to \Omega$ is denoted by $J\Phi$, its determinant by $|J\Phi|$.

To formalise the deformation process, we introduce a weighting function $g \in \mathcal{C}^1(\Omega)$ and a *monitor function* $f \in \mathcal{C}^1(\Omega)$. Both functions must be strictly positive in $\bar{\Omega}$. The reason why f is called monitor function will become clear below.

The theoretical background of our approach – like Liao's approach [4, 7, 17, 18] – is based on Moser's work [11]. The aim of the numerical grid deformation algorithm described below is to construct a bijective transformation $\Phi : \Omega \to \Omega$ which satifies

$$g(x)|J\Phi(x)| = f(\Phi(x)), \quad x \in \Omega \tag{1}$$

as well as

$$\Phi : \partial\Omega \to \partial\Omega. \tag{2}$$

If such a transformation Φ has been found, the new coordinates ξ of a grid point x are computed by

$$\xi := \Phi(x). \tag{3}$$

Applying the area formula to an element T yields

$$m(\Phi(T)) := \int_{\Phi(T)} 1\, dx = \int_T |J\Phi(x)| dx, \tag{4}$$

and using the 1×1-Gauss quadrature rule in formula (1), we obtain

$$g(x_c)\frac{m(\Phi(T))}{m(T)} = f(\Phi(x_c)) + \mathcal{O}(h^2). \tag{5}$$

Here, x_c stands for the center of T. If the function g represents the distribution of the element area in the mesh, i.e. $g(x) = m(T) + \mathcal{O}(h^2)$, $x \in T$, then we have

$$m(\Phi(T)) = f(\Phi(x_c)) + \mathcal{O}(h^2), \tag{6}$$

thus by prescribing the monitor function f, the element T will get – up to a spatially fixed scaling constant – the size defined by the value of f in the position of the image of T in the deformed grid.

In the special case $g \equiv 1$ investigated by Liao, the monitor function f determines *the relative growth or shrinkage of the elements with respect to the previous mesh*, i.e., the mesh on which the deformation takes place. In Liao's methods, the monitor function f does in general *not describe the absolute distribution of the element size in space.* If and only if the starting mesh has equidistributed element sizes, the monitor function does control the absolute element size.

In our new method, considering g to be the area distribution on the undeformed mesh, f in contrast describes the absolute mesh size distribution of the target grid, which is clearly *independent of the starting grid.* Note that condition (2) ensures that boundary points can move along the boundary only.

Then, based upon [4, 7, 17, 18], the transformation Φ is computed in four steps.

1. Scale the monitor function f or the area function g such that

$$\int_\Omega \frac{1}{f(x)} dx = \int_\Omega \frac{1}{g(x)} dx. \tag{7}$$

For the sake of simplicity, we will assume that (7) is fulfilled from now on. Let $\tilde{f}$ and $\tilde{g}$ denote the reciprocals of the scaled functions f and g.

2. Compute a grid-velocity vector field $v : \Omega \to \mathbb{R}^n$ satisfying

$$-\mathrm{div}(v(x)) = \tilde{f}(x) - \tilde{g}(x),\ x \in \Omega, \quad \text{and} \quad v(x) \cdot \mathfrak{n} = 0,\ x \in \partial\Omega, \tag{8}$$

with $\mathfrak{n}$ being the outer normal vector of the domain boundary $\partial\Omega$, which may consist of several boundary components.

3. For each grid point x, solve the initial value problem

$$\frac{\partial \varphi(x,t)}{\partial t} = \eta(\varphi(x,t),t), \quad 0 \le t \le 1, \quad \varphi(x,0) = x \tag{9}$$

with

$$\eta(y,s) := \frac{v(y)}{s\tilde{f}(y) + (1-s)\tilde{g}(y)}, \quad y \in \Omega, s \in [0,1]. \tag{10}$$

4. Define

$$\Phi(x) := \varphi(x,1). \tag{11}$$

By this method, we are able to construct a mapping Φ satisfying conditions (1) and (2).

Theorem 2.1. *Let the boundary of Ω be $\mathcal{C}^{3,\alpha}$-smooth and let $f, g \in \mathcal{C}^1(\Omega)$ be strictly positive in $\bar{\Omega}$. Then, if the mapping $\Phi : \Omega \to \Omega$ constructed above exists, it fulfills conditions (1) and (2).*

Proof: Define the auxiliary function

$$H(x,t) := |J\varphi(x,t)| \left[t\tilde{f}(\varphi(x,t)) + (1-t)\tilde{g}(\varphi(x,t)) \right]. \tag{12}$$

A direct, but tedious calculation (see Appendix) shows that

$$\frac{\partial H(x,t)}{\partial t} = 0, \tag{13}$$

and therefore we obtain

$$\begin{aligned}\frac{1}{g(x)} = \tilde{g}(x) &= |J\varphi(x,0)|\,\tilde{g}(\varphi(x,0)) \\ &= H(x,0) \\ &= H(x,1) \\ &= |J\varphi(x,1)|\,\tilde{f}(\varphi(x,1)) \\ &= |J\Phi(x)|\,\frac{1}{f(\Phi(x))}. \qquad (14)\end{aligned}$$

As by (8) the normal component of v vanishes for a boundary point x, it follows due to formula (9) that $\partial_{\mathfrak{n}}\varphi(x,t) = 0\ \forall\, t \in [0,1]$. Therefore, boundary points are moved in tangential direction only and by this the proof is finished. $\square$

The existence of such a mapping Φ is guaranteed by the following theorem from [11]. However, the mapping Φ is *not unique.*

Theorem 2.2 (Moser). *Let $0 < k \in \mathbb{N}$, $\alpha > 0$. Let $\Omega \subset \mathbb{R}^n$ be a domain with $\mathcal{C}^{3+k,\alpha}$-smooth boundary. Suppose $f, g \in \mathcal{C}^{k,\alpha}(\bar{\Omega})$ with $\int_\Omega f = \int_\Omega g$. Then there exists a $\mathcal{C}^{k+1}$-diffeomorphism Φ which fulfills*

$$g(x)|J\Phi(x)| = f(\Phi(x)) \quad \forall x \in \Omega$$

and

$$\Phi(x) = x \quad \forall x \in \partial\Omega.$$

Remark 2.3. *Note that the derivation of our new deformation method does not depend on the dimension. Although we restrict to the two-dimensional case in this article for the sake of implementational simplicity, our new deformation method is applicable to three-dimensional meshes without any modification.*

3 Numerical Realisation

This section is devoted to the numerical implementation of the four steps described above. Although the construction of the mapping Φ can be performed in any dimension, we restrict ourselves to the two-dimensional case. To prove the existence of a smooth diffeomorphism Φ, Moser uses the smoothness conditions of f, g and the domain Ω stated above. In practical computations, we relax these conditions. In the examples presented in this article, the functions f and g are strictly positive and continuous, the domain Ω may have a Lipschitz boundary.

The first step in constructing Φ is to obtain the functions f and g. In our computations, the monitor function f is defined to be the bilinear interpolant of an analytically given function. To compute g, we first determine the element size in a grid point, which is set as the arithmetic mean of the area of the elements surrounding the grid point. Then, we define g as the bilinear interpolant of these node values.

The vector field v is computed by solving the pure Neumann problem

$$-\Delta w = \tilde{f} - \tilde{g}, \quad \partial_{\mathfrak{n}} w = 0 \text{ on } \partial\Omega \qquad (15)$$

and setting $v := \nabla w$. To maintain a high degree of flexibility with respect to the underlying mesh, we compute problem (15) using its discrete weak formulation

$$(\nabla w_h, \nabla \varphi_h) = (\tilde{f} - \tilde{g}, \varphi_h) \quad \forall \varphi_h \in \mathcal{Q}_1(\mathbb{T}) \tag{16}$$

by the finite element method on the given mesh $\mathbb{T}$. Here $\mathcal{Q}_1$ denotes the function space created by continuous and elementwise bilinear functions on $\mathbb{T}$. In the following, every FEM calculation is performed with bilinear conforming finite elements unless stated differently.

The solution of the corresponding algebraic systems however requires special care, as the solution of the Neumann problem (15) is unique up to an additive constant only. To solve (15), we use a modified multigrid method in which after every iteration the side condition $\int_\Omega w_h = 0$ is imposed by adding a suitable constant [14].

Afterwards, the vector field v is approximated by the recovered gradient v_h of the finite element solution w_h. For the reconstruction of the gradient we employ the polynomial preserving recovery (PPR)-method (cf. [27]), although standard 1st-order interpolation techniques also performed well in our tests.

In the next step, we approximate the initial value problem (9) by replacing v by its discrete counterpart v_h. This leads to the initial value problem

$$\frac{\partial \varphi(x,t)}{\partial t} = \eta_h(\varphi(x,t),t), \quad 0 \leq t \leq 1, \quad \varphi(x,0) = x \tag{17}$$

with

$$\eta_h(y,s) := \frac{v_h(y)}{s\tilde{f}(y) + (1-s)\tilde{g}(y)}, \quad y \in \Omega, s \in [0,1]. \tag{18}$$

Note that this ODE system decouples into 1D-ODEs for every coordinate, which can be solved by standard ODE methods. All numerical solution methods for ODEs require one or more evaluations of the right hand side per time step. In the proposed grid deformation method, evaluating the vector field as well as $\tilde{f}$ and $\tilde{g}$ is rather expensive, as every evaluation requires searching through the grid: To evaluate a finite element function in a given point in real coordinates, the element this point belongs to has to be known. Unfortunately, we have to evaluate (from the 2nd time step on) at points which possibly have been moved to an entirely different region of the grid during the time steps already performed. Hence, the element the moved point is inside of has to be found. Note that the evaluations of v_h, $\tilde{f}$ and $\tilde{g}$ have to take place on the original grid, where these functions have been computed. It is important to realise that the grid deformation process described here needs to search the whole grid (at least one time) *per time step* and *per grid point*. Consequently, searching the grid just by "brute force", i.e. without clever search algorithms, leads to unreasonable computational costs (compare Example 3.2) with quadratic complexity, as we will show.

To perform the "brute force" approach, we loop over all elements in the grid. The search is finished and the loop terminated if the element containing the point is found. To improve this strategy, we introduce the "improved brute force" approach: Here, we store the index of the element T_{old} the point has been inside of in the previous time step. If the point is not inside T_{old} after the current time step, we perform a "brute force" search. For both search methods, the whole grid has to be searched in the worst case. Therefore, for a grid consisting of N grid points the search time is $\mathcal{O}(N^2)$.

Our raytracing search adopted from computer graphics avoids searching the entire grid. At first, it is tested if the grid point is inside the element T_{old} where it was in the previous time step. If this is true, the search is finished. Otherwise, we take the center of T_{old} and connect it with the moved point. Detecting the element edge e which intersects with this ray, we move to the element T_{new} which shares e with T_{old}. Then, setting $T_{\text{old}} := T_{\text{new}}$, we proceed as before.

Special care, however, must be taken to avoid that the ray intersects the element in an element vertex as the choice of e is not unique in this case. If this occurs, we simply modify the starting point of the ray heuristically by disturbing the coordinates of the element center.

Another case requiring special treatment may occur when searching in non-convex domains. It may happen that, although the grid point before and after the current time step is located in the domain, the ray between these points leaves the domain. In this case, we find the second intersection of this ray with the domain boundary by looping over all boundary elements. In the element containing this second intersection, the search is continued.

The maximum length of the search path and therefore the maximum amount for a single search process is $\mathcal{O}(N)$. Thus, the total search time behaves like $\mathcal{O}(N^2)$ in the worst case, too. But on reasonable grids which have roughly the same resolution in x- and y-direction, the maximum length of the search path is $\mathcal{O}(\sqrt{N})$ and thus the total search time grows like $\mathcal{O}(N^{3/2})$ in contrast to the brute force approaches, where the search time still shows quadratic growth.

Potentially, hierarchical search methods based on spatial partitions feature even higher speed, as the search time per evaluation needs only $\mathcal{O}(\log N)$ operations yielding a total search time of $\mathcal{O}(N \log N)$. In contrast to the raytracing search, these methods do not benefit from the fact that the movements per deformation time step are usually small. Therefore, we render the raytracing approach a well suited search method in grid deformation as compromise.

Test Problem 3.1. *As a first test problem, we consider the unit square* $\Omega = [0,1]^2$ *triangulated by a tensor product mesh with* N *grid points. As monitor function, we choose*

$$f(x) = \min\left\{1, \max\left\{\frac{|d-0.25|}{0.25}, \epsilon\right\}\right\}, \quad d := \sqrt{(x_1-0.5)^2+(x_2-0.5)^2}. \tag{19}$$

The parameter ϵ *is set to 0.1. This setting implies that on the deformed grid the largest cell has 10 times the area of the smallest one.*

Example 3.2. *To evaluate the time needed for searching the grid during the solution of the ODE (17), we consider Test Problem 3.1. The computations are performed on a dual Opteron 250. In this example, ten Runge-Kutta-3 steps with step size fixed to 0.1 are performed yielding 30 evaluations per grid point. We compare the total search time in seconds, i.e. the search time needed for all evaluations performed in the deformation process, of the three search algorithms described above on various levels of (regular) refinement. Due to the natural inaccuracy in the measurement of time, all measurements have been repeated until the relative error of the mean value of the measured times decreased below 1%.*

The results are collected in Table 3.1. It is obvious that the raytracing search is by far faster than the brute force approach, which by its quadratic search time leads to inappropriate search times (≈ 9 h in our example) for still quite small grids. In contrast to this, the search amount of the raytracing search is almost $\mathcal{O}(N)$ in this example. The same calculations were additionally performed with 50 instead of 10 Runge-Kutta steps and a step size of 0.02. The comparison of the search times in this case (Table 3.2) with the ones discussed above shows that the total search times for the brute force approaches grows by a factor of 5 corresponding to the number of times steps. Instead of this, the search time in the case of raytracing search grows only by a factor of 3 for the calculation with 65536 grid points. This phenomenon stems from the smaller time steps in the latter calculations resulting in shorter search paths.

To summarise this section, we give a short description of our basic grid deformation method in algorithmic form.

Algorithm 3.3 (Basic grid deformation).

```
input:    • $f$ : monitor function
          • $GRID$ : computational grid

output:   • $GRID$ : deformed grid

function Deformation($f$, $GRID$) : $GRID$

    compute $\tilde{f} - \tilde{g}$, $\tilde{g} = \tilde{g}(GRID)$

    solve  $(\nabla w_h, \nabla \varphi_h) = (\tilde{f} - \tilde{g}, \varphi_h) \quad \forall \varphi_h \in \mathcal{Q}_1(\mathbb{T})$

    $v_h :=$ recovered_gradient($w_h$)

    DO FORALL $x \in GRID$

        solve $\frac{\partial \varphi(x,t)}{\partial t} = \eta_h(\varphi(x,t),t), \quad 0 \le t \le 1, \quad \varphi(x,0) = x$
        $\Phi(x) := \varphi(x,1)$

    ENDDO

    RETURN $\Phi(GRID)$

END Deformation
```

4 Numerical Analysis of the Deformation Method

To investigate the quality of the obtained grids, we consider the fraction

$$\frac{f(x)}{area(x)} \tag{20}$$

where $area(x)$, the element area distribution of the deformed mesh, is obtained in the way like g on the undeformed grid. If the desired area distribution is achieved, then the fraction (20) is spatially constant (note that f describes the spatial distribution of the

N	raytracing	imp. brute force	brute force
256	$9.36 \cdot 10^{-3}$	$8.56 \cdot 10^{-2}$	$5.65 \cdot 10^{-2}$
1024	$3.96 \cdot 10^{-2}$	$2.23 \cdot 10^{0}$	$8.39 \cdot 10^{0}$
4096	$1.70 \cdot 10^{-1}$	$6.20 \cdot 10^{1}$	$1.30 \cdot 10^{2}$
16384	$8.22 \cdot 10^{-1}$	$1.39 \cdot 10^{3}$	$2.05 \cdot 10^{3}$
65536	$4.28 \cdot 10^{0}$	$2.69 \cdot 10^{4}$	$3.25 \cdot 10^{4}$
262144	$2.42 \cdot 10^{1}$	-	-

Table 3.1: Total search time needed by the different search methods in the case of Test Problem 3.1, 10 time steps

N	raytracing	imp. brute force	brute force
256	$4.73 \cdot 10^{-2}$	$1.33 \cdot 10^{-1}$	$2.82 \cdot 10^{-1}$
1024	$1.83 \cdot 10^{-1}$	$2.61 \cdot 10^{0}$	$4.21 \cdot 10^{0}$
4096	$7.32 \cdot 10^{-1}$	$7.58 \cdot 10^{1}$	$6.49 \cdot 10^{2}$
16384	$3.03 \cdot 10^{0}$	$2.27 \cdot 10^{3}$	$1.02 \cdot 10^{4}$
65536	$1.32 \cdot 10^{1}$	$6.61 \cdot 10^{4}$	$1.62 \cdot 10^{5}$
262144	$6.10 \cdot 10^{1}$	-	-

Table 3.2: Total search time needed by the different search methods in the case of Test Problem 3.1, 50 time steps

cell sizes up to a spatially fixed constant). To force this constant to be 1, we scale the function $area(x)$ with a multiplicative constant c to achieve

$$\int_\Omega a(x)\,dx = \int_\Omega f(x)\,dx, \tag{21}$$

where $a(x) := c \cdot area(x)$. Now, we define the quality function

$$q(x) := \frac{f(x)}{a(x)}. \tag{22}$$

The overall quality of the grid adaptation according to the desired cell size can be measured by the deviation of $q(x)$ from the constant function 1 leading to the quality measure Q defined by

$$Q := ||q - 1||_{l^2}. \tag{23}$$

In our tests, the discrete l^2 norm has been chosen for convenience.

As above, we consider Test Problem 3.1. Now, we investigate how the error induced by the approximate solution of the ODEs influences the quality parameter Q. To do so, we compare Q after one deformation cycle on a tensor product grid with 4096 grid points. As ODE solver, we apply explicit Euler's method (EE) and several Runge-Kutta type methods: Heun's method (HEUN), the classical Runge method of third order (RK3) and the standard Runge-Kutta method of fourth order (RK4). Furthermore, we employ Adams-Bashforth methods of order two and three (AB2, AB3) as representatives of linear multistep methods. The starting values are obtained by Heun's method. These computations are performed with fixed step size. The results are shown in Table 4.1 and Table 4.2. Note that one Runge-Kutta step requires several costly evaluations of the right hand side of the ODE per step in contrast to the linear multistep methods. Therefore, in a corresponding Figure 4.1 we compare Q in relation to the number of vector field evaluations per grid point instead of the number of time steps. It turns out that the high order Runge-Kutta methods perform best, even when only few large time steps are applied.

steps	EE	AB2	AB3	HEUN	RK3	RK4
3	$4.96 \cdot 10^{-1}$	-	$1.98 \cdot 10^{0}$	$8.43 \cdot 10^{-2}$	$4.54 \cdot 10^{-2}$	$4.47 \cdot 10^{-2}$
4	$2.62 \cdot 10^{-1}$	$1.67 \cdot 10^{-1}$	$2.41 \cdot 10^{-1}$	$6.32 \cdot 10^{-2}$	$4.44 \cdot 10^{-2}$	$4.42 \cdot 10^{-2}$
5	$1.93 \cdot 10^{-1}$	$1.06 \cdot 10^{-1}$	$1.05 \cdot 10^{-1}$	$5.46 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
7	$1.38 \cdot 10^{-1}$	$6.96 \cdot 10^{-2}$	$5.79 \cdot 10^{-2}$	$4.85 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
10	$1.04 \cdot 10^{-1}$	$5.37 \cdot 10^{-2}$	$4.69 \cdot 10^{-2}$	$4.59 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
20	$6.81 \cdot 10^{-2}$	$4.56 \cdot 10^{-2}$	$4.41 \cdot 10^{-2}$	$4.44 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
30	$5.66 \cdot 10^{-2}$	$4.46 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.41 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
40	$5.32 \cdot 10^{-2}$	$4.43 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.41 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
50	$5.07 \cdot 10^{-2}$	$4.42 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
70	$4.83 \cdot 10^{-2}$	$4.41 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$
100	$4.67 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$	$4.40 \cdot 10^{-2}$

Table 4.1: Quality measure Q for different step sizes and ODE solvers for Test Problem 3.1, 4096 grid points

steps	EE	AB2	AB3	HEUN	RK3	RK4
3	$6.60 \cdot 10^{-1}$	-	-	$8.15 \cdot 10^{-2}$	$3.11 \cdot 10^{-2}$	$1.89 \cdot 10^{-2}$
4	$3.12 \cdot 10^{-1}$	$2.31 \cdot 10^{-1}$	$2.84 \cdot 10^{-1}$	$4.86 \cdot 10^{-2}$	$1.81 \cdot 10^{-2}$	$1.08 \cdot 10^{-2}$
5	$2.18 \cdot 10^{-1}$	$1.21 \cdot 10^{-1}$	$1.25 \cdot 10^{-1}$	$3.20 \cdot 10^{-2}$	$1.52 \cdot 10^{-2}$	$8.17 \cdot 10^{-3}$
7	$1.44 \cdot 10^{-1}$	$6.13 \cdot 10^{-2}$	$5.81 \cdot 10^{-2}$	$1.76 \cdot 10^{-2}$	$9.07 \cdot 10^{-3}$	$6.12 \cdot 10^{-3}$
10	$9.96 \cdot 10^{-1}$	$3.09 \cdot 10^{-2}$	$2.62 \cdot 10^{-2}$	$9.97 \cdot 10^{-3}$	$5.66 \cdot 10^{-3}$	$5.38 \cdot 10^{-3}$
20	$5.08 \cdot 10^{-2}$	$9.84 \cdot 10^{-3}$	$7.01 \cdot 10^{-3}$	$5.92 \cdot 10^{-3}$	$5.34 \cdot 10^{-3}$	$5.24 \cdot 10^{-3}$
30	$3.45 \cdot 10^{-2}$	$6.63 \cdot 10^{-3}$	$5.44 \cdot 10^{-3}$	$5.46 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$
40	$2.63 \cdot 10^{-2}$	$5.82 \cdot 10^{-3}$	$5.26 \cdot 10^{-3}$	$5.35 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$
50	$2.14 \cdot 10^{-2}$	$5.54 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.30 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$
70	$1.59 \cdot 10^{-2}$	$5.36 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.26 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$
100	$1.19 \cdot 10^{-2}$	$5.28 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.25 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$	$5.23 \cdot 10^{-3}$

Table 4.2: Quality measure Q for different step sizes and ODE solvers for Test Problem 3.1, 65536 grid points

Remark 4.1. *Due to numerical errors, it may happen that some of the intermediate evaluation points of the Runge-Kutta methods are not inside of the domain. As the functions to evaluate are not defined in these points, the intermediate points are projected onto the domain in this case and the calculation proceeds as usual. For more profound investigations of this phenomenon and of the different ODE solvers, see [14].*

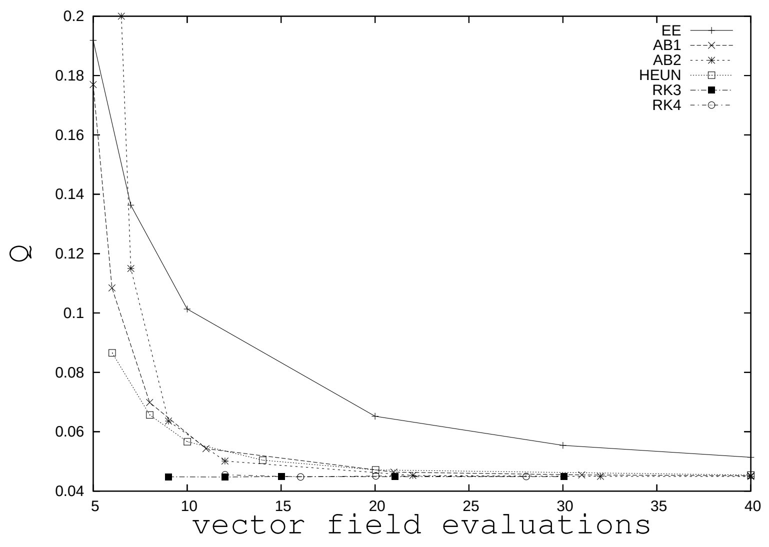

Figure 4.1: Quality measure Q vs. number of vector field evaluations for Test Problem 3.1, 4096 grid points

In practical computations, however, we cannot expect the resulting grid to feature exactly the cell size distribution prescribed by f. This is due to the fact that we have to rely on an approximation v_h of the vector field v during the computation. Additional error is induced by the numerical solution of the ODE (17). To further improve the cell size adjustment, we repeat the deformation algorithm 3.3 on the newly deformed grid until $Q < TOL$ is reached. We refer to this kind of iterated deformation as *correction iteration.*

Algorithm 4.2 (Iterated grid deformation).

input:
- *f: monitor function*
- *$GRID$: computational grid*
- *$NCORR$: maximal number of correction steps*
- *TOL : tolerance for Q*

output:
- *$GRID$: deformed grid*

```
function IteratedDeformation($f$, $GRID$, $NCORR$, $TOL$) : $GRID$
    DO i = 1, NCORR
        $GRID$ := Deformation($f$, $GRID$)
        Q := Q(GRID)
        IF (Q < TOL) EXIT LOOP
    ENDDO
    RETURN GRID
END IteratedDeformation
```

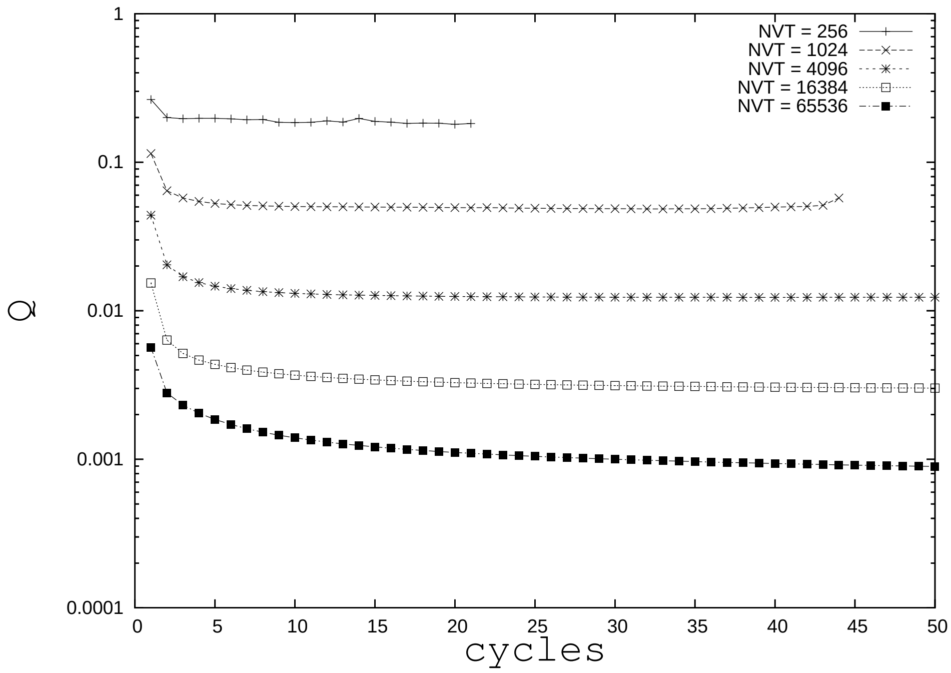

Figure 4.2: Quality measure Q vs. number of correction iterations for Test Problem 3.1

The possibility of defining such an algorithm is one of the biggest advantages of our method over Liao's method, which requires the starting grid to have elements with equal area and therefore prevents any iteration process. Unfortunately, even with iterating the deformation process, it is by principle reasons impossible to achieve $Q = 0$ exactly, which is caused by the fact that in reality the area distribution is a piecewise constant function. Therefore, the interpolation error in computing g remains and prevents Q from converging to 0. But for practical applications, $Q \approx 0.05$ has turned out to be sufficient (see below).

Example 4.3. *To verify the statements made above, we return to Test Problem 3.1. From Figure 4.2 depicting the quality parameter Q vs. the number of deformation cycles for different levels of refinement (NVT = number of vertices), it becomes evident that the error caused by the approximate computation of Φ can be eliminated by iteration, but the deviation of Q from 0 caused by the interpolation of the area function remains. Similar to the computations in example 3.2, we choose RK3 as ODE-solver with time step size 1/10.*

In practical computations, it turns out to be sufficient to perform at most two correction steps. In the case of Test Problem 3.1, when computed on a tensor product grid with 1024 grid points and using 10 RK3-steps, there is visually merely a small difference between the grid obtained without correction and the one produced by the first correction step; the grids created using one and two correction steps do not exhibit any visual difference at all (see Figure 4.3). In contrast to this, the quality measure Q decays from $1.15 \cdot 10^{-1}$ (without correction) to $6.80 \cdot 10^{-2}$ (first correction step) and finally $6.10 \cdot 10^{-2}$

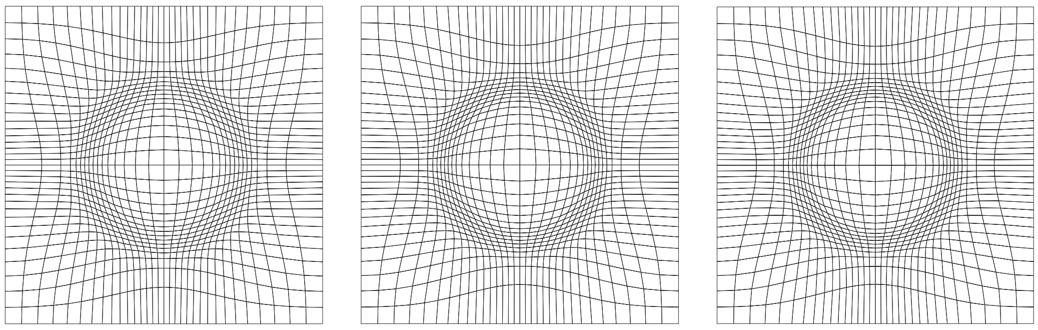

Figure 4.3: Computational grid after zero, one and two correction steps (Test Problem 3.1, 1024 grid points)

(second correction step).

When applying very harsh deformation, e.g. in the case of monitor functions with extremely steep gradients or monitor functions implying extreme element size differences, the corresponding vector field v cannot be resolved properly on the undeformed grid. Thereby, the numerical grid deformation process can be disturbed to that extent that the deformed elements are no longer convex. This is not due to theoretical limitations of the method, but due to the numerical error induced by the approximate solution of the Poisson problem (16). In this context, the error produced by the numerical solution of the ODE (17) seems to be far less critical. Imposing $\epsilon = 1/100$ instead of $1/10$ in formula (19) of Test problem 3.1, for instance, leads to non-convex elements, regardless of the number of time steps chosen in the ODE solver. As a remedy, one can solve the Poisson problem (16) in a more accurate manner, e.g. by using higher order finite elements or a highly refined mesh. These aproaches require demanding calculations and render therefore incompetetive. To overcome the limitation described here, we iterate algorithm **IteratedDeformation** using the *blended monitor function* f_s defined by

$$f_s(x) := sf(x) + (1-s)g(x), \quad s \in [0,1] \tag{24}$$

instead of the monitor function itself. For compatibility reasons, we require f and g to have the same integral mean value here.

Algorithm 4.4 (Grid deformation with combined correction).

input:
- *$NADAP$: number of adaptation steps*
- *$GRID$: computational grid*
- *$NCORR(i)$: maximal number of correction steps in i-th adaptation step*
- *$TOL(i)$: tolerance for Q in i-th adaptation step*
- *$S(i)$: blending parameter in i-th adaptation step, $S(NADAP) = 1$*

output:
- *$GRID$: deformed grid*

```
function EnhancedDeformation($f$, $GRID$, $NCORR$, $NADAP$, $TOL$, $S$) : $GRID$
    DO i = 1, NADAP
        $GRID$ := IteratedDeformation($f_{S(i)}$, $GRID$, $NCORR(i)$, $TOL(i)$)
    ENDDO
    RETURN GRID
END EnhancedDeformation
```

To demonstrate the gain of robustness by performing the adaptation iteration, we investigate the following test problem.

Test Problem 4.5. *We consider the domain $\Omega = [0,1] \times [0,6]$ with the grid shown on top of Figure 4.4. As monitor function, we choose*

$$f(x) = \min\left\{1, \max\left\{\frac{|d-0.25|}{0.25}, \epsilon\right\}\right\}, \quad d := \sqrt{(x_1 - 4.5)^2 + (x_2 - 0.4)^2}, \quad \epsilon = 1/10. \tag{25}$$

Example 4.6. *Using bilinear elements to compute v_h, it is not possible to perform the deformation prescribed in Test Problem 4.5 in one step as inverted elements emerge due to numerical inaccuracies. This phenomenon occurs regardless of the ODE solver chosen and the step size prescribed. The failure in this test problem can be cured by applying two adaptation steps instead of one. Here, we set $NCORR(i) = 0$ for $i < NADAP$ and $NADAP = 2$ in algorithm 4.4. The blending parameter $S(i)$ is computed by*

$$S(i) = \sqrt{\frac{i}{NADAP}}. \tag{26}$$

The computational grids consist of 2016 grid points. The grids obtained after both adaptation steps and one correction step are shown in Figure 4.4. After the adaptation iterations, a quality measure of $Q = 8.42 \cdot 10^{-2}$ is reached. The correction step improves this to $4.41 \cdot 10^{-2}$.

5 Examples and Applications

In this section, the emphasis is put on applications of the deformation method described in the former sections. As mentioned in the introduction, one field of application for

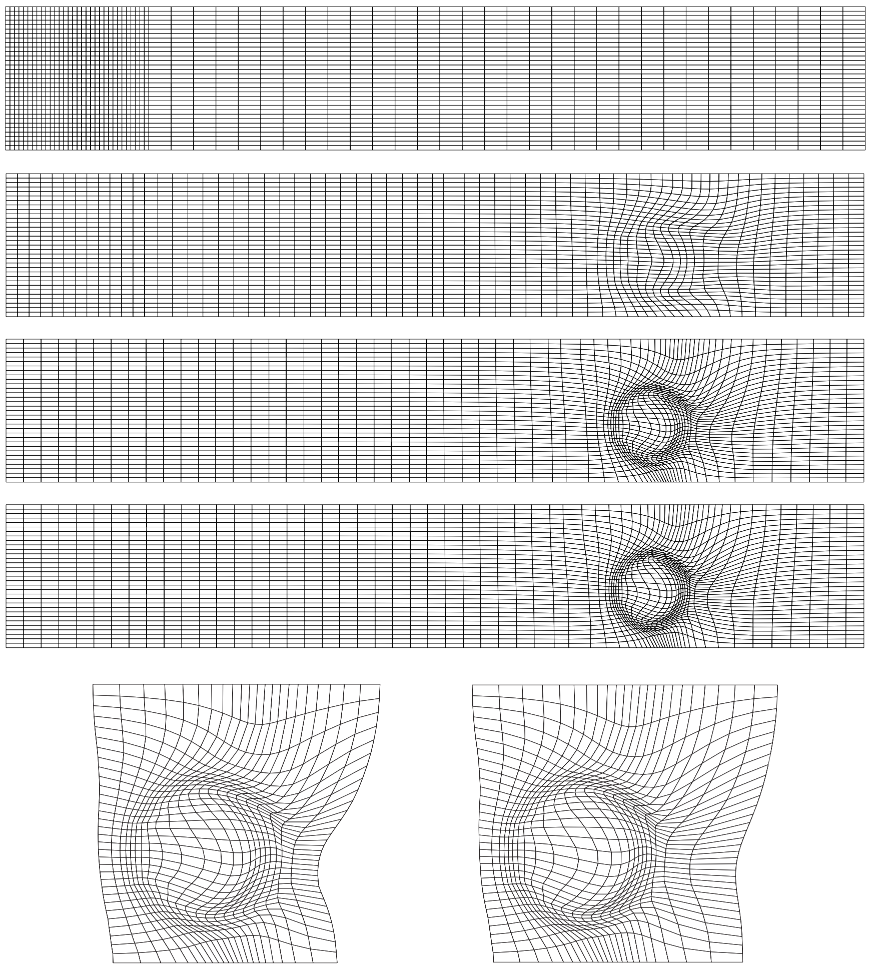

Figure 4.4: Test Problem 4.5: initial grid, grid after the first and second adaptation step, and the grid after one correction step (from top to bottom). Zoom into the region around the circle after the two adaptation steps (left) and a further correction step (right).

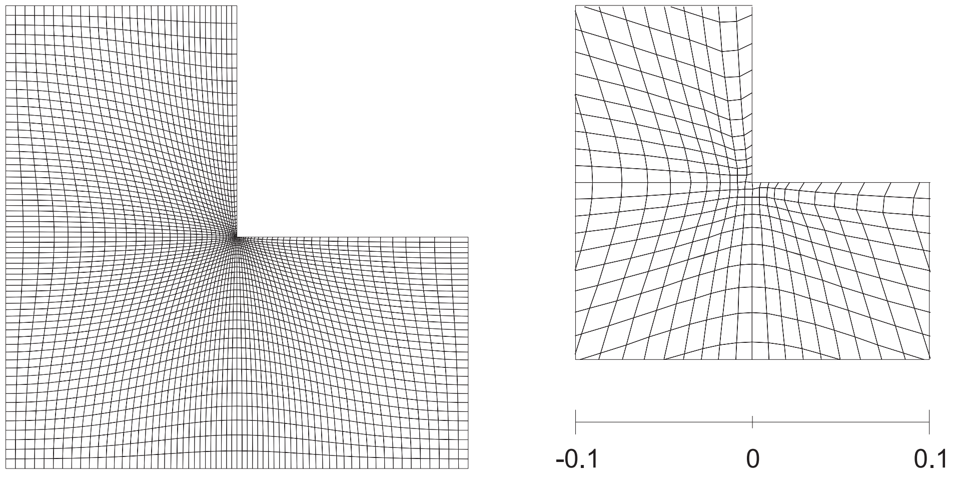

Figure 5.1: Computational grid for the L-domain (left: whole domain, right: zoom into region of reentrant corner)

grid deformation is a posteriori error control and adaptivity. A classical example to show the superiority of adaptive FEM is the Laplace equation on the L-domain $[-1,1]^2 \setminus [0,1]^2$. Choosing appropriate Dirichlet boundary conditions, the (harmonic) solution is not H^2-regular, but features an edge singularity in the reentrant corner. When adapting according to the H^1-norm, the grid has to be concentrated extremely in this corner to balance the singularity.

Example 5.1. *To demonstrate the benefit of grid deformation in this context, we start with an equidistant computational grid and deform it using the monitor function*

$$f_L(x) = \min\left(1, \max(|x|, h \cdot c_0)\right). \tag{27}$$

In this equation, h denotes the mesh width on the undeformed grid, c_0 is set to $1 \cdot 10^{-2}$. The resulting grid is displayed in Figure 5.1. In Figure 5.2, we show the decay of the error on the deformed and the undeformed mesh, respectively, vs. the number of grid points. The adjusted grid is obtained using four adaptation and two correction steps. To solve the ODEs, five $RK3$-steps with step size fixed to 0.2 have been employed. Like in all other examples in this paper, the computation was performed using conforming bilinear elements.

Figure 5.2 shows that by adapting the mesh via grid deformation convergence of almost optimal order (0.493 vs. 0.5) can be achieved, while without deformation the expected detoriation of the convergence order takes place.

In the following, we want to figure out whether grid deformation helps to improve the accuracy of the computation of derived quantities in practical calculations for the incompressible Navier-Stokes equations [21]. For this purpose, we consider an example based on a well known CFD benchmark, which was investigated in [23].

Example 5.2. *Consider a channel of length 2.2 and height 0.41 (cf. Figure 5.3). The upper and lower boundary parts of the channel are treated as rigid walls. On the left*

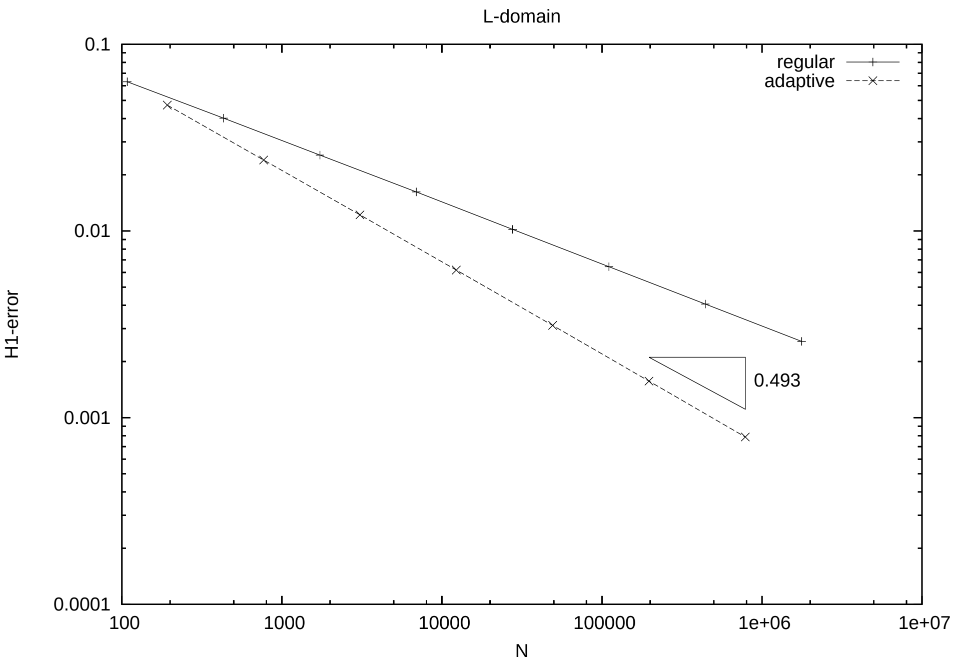

Figure 5.2: Decay of the gradient error for the L-domain with and without grid deformation vs. number of grid points N

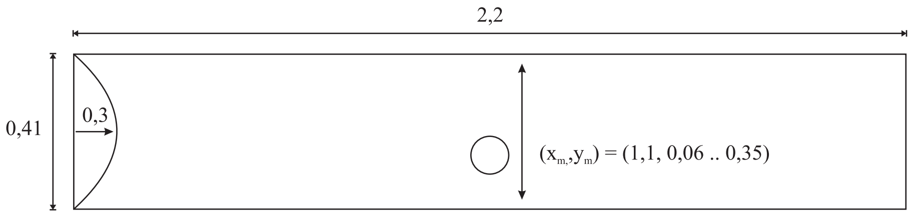

Figure 5.3: Modified CFD benchmark geometry for Example 5.2

a parabolic inflow profile with maximum speed of 0.3 *is prescribed, while on the right boundary segment natural boundary conditions describe the outflow. A rigid body with the shape of a cylinder with radius* 0.05 *is put into the flow. The midpoint* $\vec{p}_m := (x_m, y_m)$ *of this cylinder is set to* $x_m = 1.1$*, while* y_m *is modified in the range* $[0.06, 0.35]$*. In this example, we plot the drag and lift coefficients, depending on the y-position of the cylinder.*

For the calculation, a finite element approach is used that bases on the rotated bilinear nonconforming quadrilateral finite element $\tilde{\mathcal{Q}}_1$ for the velocity whereas the pressure is approximated by $\mathcal{Q}_0$ finite elements. The boundary conditions on the walls of the channel are implemented the usual way. For the boundary conditions on the cylinder we use, in contrast, the fictitious boundary method that was introduced in [22]. This enables us to use our grid deformation technique:

We start with an equally distributed computational mesh which consists of two rectangular cells in the coarse grid, forming the whole channel. This coarse grid (labelled as *level 1*) is refined 4-7 times with regular refinement, resulting in grids labeled as *level 5* to *level 8*. Table 5.1 summarises information about the grids resulting from this procedure. Level 8, for instance, consists of ~ 66000 edges and ~ 32000 elements, which gives ~ 164000 degrees of freedom for the $\tilde{\mathcal{Q}}_1/\mathcal{Q}_0$ approach in $2D$.

Level	#vertices	#edges	#elements	#DOF
5	561	1072	512	2656
6	2145	4192	2048	10432
7	8385	16576	8192	41344
8	33153	65920	32768	164608
10	525825	1050112	524288	2624512

Table 5.1: Statistics about the computational grids for Example 5.2

The resulting grid is now deformed with the help of an appropriate monitor function (see below) using 3 adaptation and 2 correction steps. After the deformation process, we use the fictitious boundary method to impose boundary conditions for all cells in the *inner* of the cylinder. A steady state simulation is performed based on this setting to obtain drag and lift coefficients on the cylinder. Afterwards the position of the cylinder is moved vertically and the grid is re-adapted like above, based on the previous mesh. Then, the calculation is repeated to obtain the next pair of results for drag and lift. We start with $y_m = 0.06$ and increase y in steps of 0.005 until we reach $y_m = 0.35$.

Remark 5.3. *In theory, the grid deformation method formulated above should not produce tangled elements, and all examples shown so far produced appropriate results when adapting and re-adapting the grid. Nevertheless, when performing more and more re-adaptation steps and additionally moving the center of the object like in this example, the grid gets more and more disturbed due to unavoidable numerical errors arising from anisotropic cells. As fast and effective remedy, we simply perform several Laplacian grid smoothing steps (cf. [20]) before each adaptation/correction step. This greatly helps to keep the quality of the mesh appropriate throughout the whole simulation.*

To obtain reference values for drag and lift with this configuration, we perform simulations for all these y-positions at level 10 without using any grid deformation. We repeated this reference simulation with different discretisation techniques (upwind, streamline-diffusion) and found no significant numerical difference. The simulations with (and without) grid deformation on lower levels are all performed using the upwind stabilisation technique (cf. [21]).

Test Problem 5.4. *For a point $\vec{x} = (x, y)$ consider the following functions:*

$$\begin{aligned} h_1(\vec{x}) &:= \max\left\{\frac{1}{40},\, 4\cdot dist(\vec{x})\right\} \\ h_2(\vec{x}) &:= \max\left\{\frac{1}{8},\, \frac{7}{8}x - \frac{37}{40}\right\} \end{aligned}$$

with

$$dist(\vec{x}) := |\ |\vec{x} - \vec{p}_m| - 0.05\ |$$

describing the minimum distance of $\vec{x}$ to the boundary of the cylinder. So, h_1 describes simply the distance of $\vec{x}$ to the cylinder, while h_2 is $1/8$ in the upstream part of the channel *and increasing linearly behind the cylinder, resulting in a value of* $h_2(2.2) = 1$ *at the end of the channel. Using these functions we create now three different test configurations:*

(A) Calculation of drag and lift without any grid deformation.

(B) Calculation of drag and lift with the monitor function

$$f_1(\vec{x}) := \min\{1, h_1(\vec{x})\}.$$

(C) Calculation of drag and lift with the monitor function

$$f_2(\vec{x}) := \min\{1, h_1(\vec{x}), h_2(\vec{x})\}.$$

Test configuration (A) is obviously the simplest one without any deformation. Test configuration (B) refines the mesh only around the cylinder, while test configuration (C) refines not only around the cylinder, but also in the upstream part of the channel (cf. Figure 5.4). The last test configuration therefore introduces the additional a priori knowledge about the test problem that additional error is to be expected in the inflow (cf. [3]).

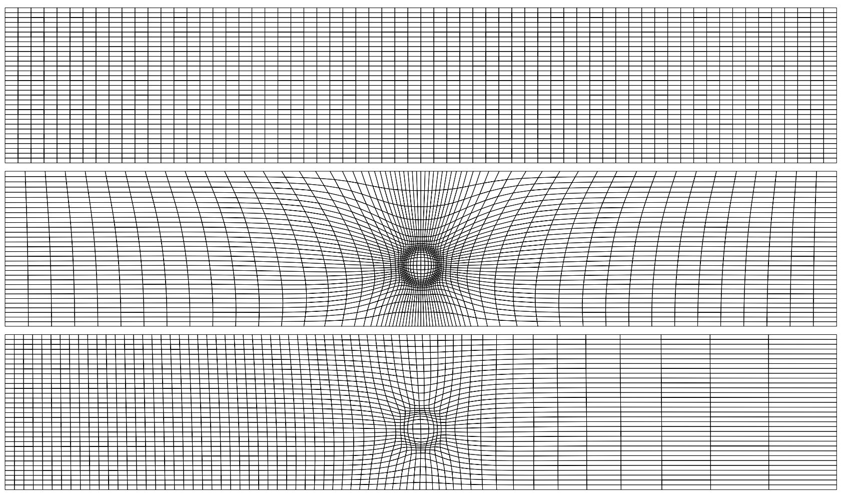
Figure 5.4: Appearance of the mesh (here level 6) for test configuration (A), (B) and (C) (from top to bottom)

Figure 5.5 to 5.7 show the calculated drag and lift coefficients for level 6-8 for the different test configurations. The calculated values obtained with the help of the grid adaptation are obviously much closer to the reference solution than in the case of no grid adaptation at all. Furthermore, it can be seen that with additional adaptation in the upstream part the quality of the calculated values increases even more: At level 7, there is merely a visual difference between the drag coefficients obtained by grid adaptation to those calculated on level 10 without.

To quantify these visual impressions with numerical values we also measured the difference of the calculated drag and lift coefficients with the reference values. Table 5.2 depicts the error on the different levels, comparing the drag and lift coefficients of

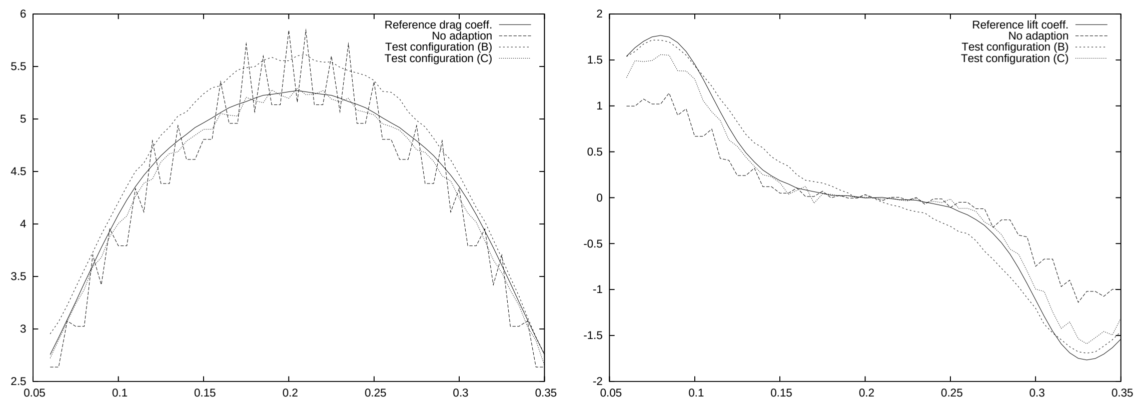

Figure 5.5: Comparison of the drag coefficients (left) and lift coefficients (right) for y-position 0.06..0.35 on level 6. The pictures show the reference curve calculated on level 10 and the coefficients obtained by test configuration (A), (B) and (C).

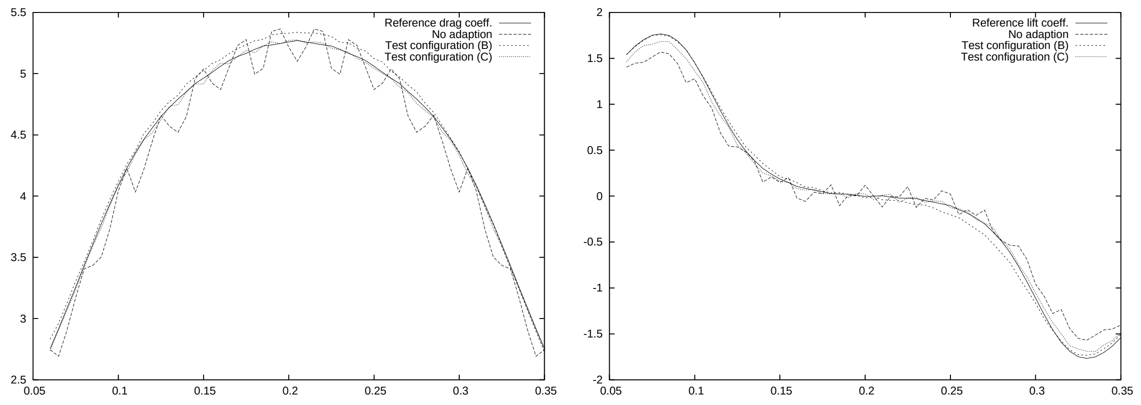

Figure 5.6: Same as Figure 5.5, but for level 7.

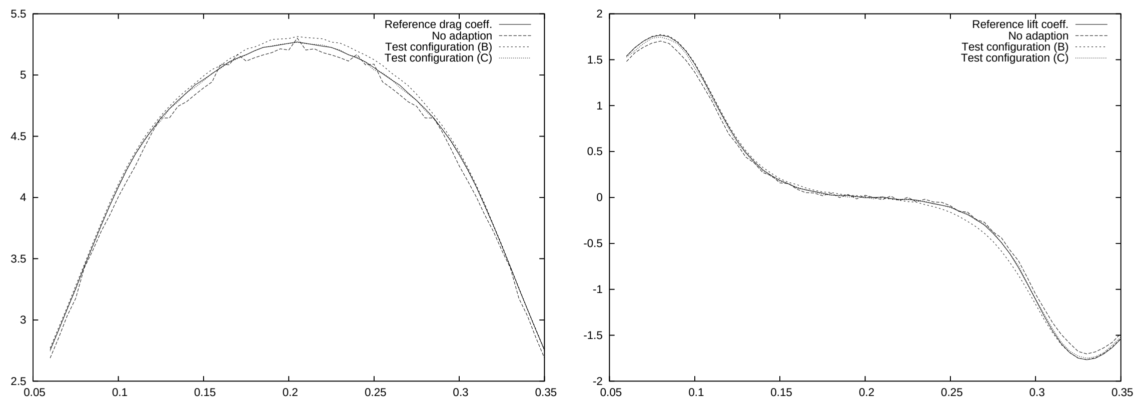

Figure 5.7: Same as Figure 5.5, but for level 8.

Lv.	$\|S_d - D_A\|_{L_B}$	$\|S_d - D_B\|_{L_B}$	$\|S_d - D_C\|_{L_B}$
5	1.027E-01	9.311E-02	5.447E-02
6	3.635E-02	2.917E-02	**8.960E-03**
7	2.063E-02	**6.896E-03**	2.744E-03
8	**7.800E-03**	4.608E-03	1.003E-03
Lv.	$\|S_l - L_A\|_{L_B}$	$\|S_l - L_B\|_{L_B}$	$\|S_l - L_C\|_{L_B}$
5	6.454E-02	2.398E-02	4.895E-02
6	5.564E-02	1.849E-02	1.864E-02
7	2.076E-02	**7.249E-03**	**6.727E-03**
8	**6.986E-03**	4.807E-03	2.085E-03

Table 5.2: $\| \cdot \|_{l_2}$-norm of the difference between the reference drag/lift vector and the drag/lift vector calculated with the three different grid deformation configurations.

the reference calculation on level 10 with those of the three test configurations. S_d and S_l describe the vector that consists of the 59 reference drag (and lift) values for all the y-coordinates 0.06, 0.11,... D_A, D_B and D_C represent the drag coefficient vectors for the three test configurations, while L_A, L_B and L_C denote the lift coefficient vectors.

Table 5.2 shows clearly the advantage of the adapted grids. The error on lower levels, especially level 6 and 7, are mostly by orders better than on the undeformed grid. The additional a priori knowledge used in test configuration (C) increases the accuracy of all drag coefficients even more while improving the lift coefficients significantly starting at level 7. Here it can be seen that the drag and lift coefficients of test configuration (C) are roughly comparable with those calculated without grid deformation on a grid that is refined 1-2 times more. It can be expected that using a sophisticated error indicator, even higher accuracy will be within reach.

As a final remark we also want to highlight another aspect that can be seen when analysing these results. The values calculated with the grid deformation method are not only more accurate, the resulting curves are also much *smoother*. While on low levels the drag (and also the lift) curves are oscillating heavily around the reference values, those calculated with the grid deformation show a by far less oscillating behaviour. The reason for this *zig-zagging* on low levels without using grid deformation quickly becomes clear when analysing the method: For the drag and lift evaluation, the fictitious boundary method performs a volume integration about all cells inside of the cylinder. When moving the cylinder to another position on the uniform grid, large cells "jump" inside or outside of the cylinder, thus disturbing the calculation method. These disturbances nearly vanish or at least are damped very much when there are only "small" cells around the boundary of the cylinder, as it happens on higher levels of refinement or when concentrating cells around the interface.

This observation gives rise to usability of the grid deformation method in combination with optimisation algorithms. There are a number of optimisation algorithms in the context of *PDE constrained optimisation* which rely on the accurate evaluation of functionals (like drag or lift) and/or their derivatives (see e.g. [15, 6]). Here, the grid deformation method can clearly help to dampen numerical noise arising from an inappropriate triangulation on lower levels, which otherwise could lead to non-optimal behaviour of those algorithms (cf. [15, 24]).

6 Conclusion and outlook

In this paper, we presented a new mesh deformation method which relies on solving only Poisson problems for the deformation and 1D-IVPs for every grid point in every dimension. The deformation process is controlled by a monitor function which quantitatively prescribes the cell size distribution of the resulting mesh. We described in detail implementational aspects and showed the accuracy and robustness of our new method on several examples. In this context, we analysed the effects of the deformation method on the accuracy of a flow simulation, governed by the Navier-Stokes equation. We have seen that solutions calculated on properly deformed meshes exhibit similar quality as solutions on undeformed meshes refined once or twice more.

The deformation method proposed in this article, although presented in the 2D context, is not restricted to two-dimensional meshes, but can be applied in any dimensions. Therefore, the investigation of our deformation method in three dimensions has been recently started. Another matter of future investigation will be the embedding of a posteriori error control. Here, the monitor function is not given analytically like in this paper, but has to be derived from a computed error distribution. Furthermore, to increase the flexiblity of the adaptation process a combination of the deformation method with h-adaptivity is currently being investigated, which will lead to a full rh-adaptivity approach. This algorithm will become part of the new hardware-oriented FEM package FEAST.

7 Acknowledgements

We want to thank Reinhard Zwirnmann for a first implementation of Liao's dynamic method. Last, but not least, we want to thank Christian Becker, as he developed the core of the FEAST package.

8 Appendix

Lemma 8.1. *Formula (13) holds.*

Proof: From equation (10), we have

$$v(x) = \eta(x,t)\left(t\tilde{f}(x) + (1-t)\tilde{g}(x)\right). \tag{28}$$

Applying the chain rule and the div-Operator, we find

$$\begin{aligned}\operatorname{div}(v(x)) &= \operatorname{div}\left[\eta(x,t)\cdot\left(t\tilde{f}(x) + (1-t)\tilde{g}(x)\right)\right] \\ &= \left[t\tilde{f}(x) + (1-t)\tilde{g}(x)\right]\operatorname{div}\eta(x,t) \\ &\quad + \left(t\nabla\tilde{f}(x) + (1-t)\nabla\tilde{g}(x)\,,\ \eta(x,t)\right).\end{aligned}$$

Therefore it follows

$$\begin{aligned}\operatorname{div}\ (v(\varphi(x,t))) &= \operatorname{div}\left(\eta(\varphi(x,t),t)\right)\left[t\tilde{f}(\varphi(x,t)) + (1-t)\tilde{g}(\varphi(x,t))\right] \\ &\quad + \left(t\nabla\tilde{f}(\varphi(x,t)) + (1-t)\nabla\tilde{g}(\varphi(x,t))\,,\ \eta(\varphi(x,t))\right). \end{aligned} \tag{29}$$

Starting from the ODE (9), we have by Abel's formula

$$\begin{aligned} |J\varphi(x,t)| &= \exp \int_0^t \mathrm{tr}(J\eta(\varphi(x,s),s))ds \\ &= \exp \int_0^t \mathrm{div}\, \eta(\varphi(x,s),s)ds \end{aligned} \tag{30}$$

and by differentiation of (30) we obtain

$$\frac{\partial}{\partial t}|J\varphi(x,t)| = |J\varphi(x,t)|\mathrm{div}\eta(\varphi(x,t),t)). \tag{31}$$

Therefore, we obtain

$$\begin{aligned} \frac{\partial}{\partial t}H(x,t) &= \left(\frac{\partial}{\partial t}|J\varphi(x,t)|\right) \cdot \left[t\tilde{f}(\varphi(x,t)) + (1-t)\tilde{g}(\varphi(x,t))\right] \\ &\quad +|J\varphi(x,t)| \cdot \Bigg[\tilde{f}(\varphi(x,t)) + t\left((\nabla\tilde{f})(\varphi(x,t)), \frac{\partial}{\partial t}\varphi(x,t)\right) \\ &\qquad -\tilde{g}(\varphi(x,t)) + (1-t)\left((\nabla\tilde{g})(\varphi(x,t)), \frac{\partial}{\partial t}\varphi(x,t)\right)\Bigg] \\ &\overset{(31)}{=} |J\varphi(x,t)| \cdot \mathrm{div}\, \eta(\varphi(x,t),t) \cdot \left[t\tilde{f}(\varphi(x,t)) + (1-t)\tilde{g}(\varphi(x,t))\right] \\ &\quad +|J\varphi(x,t)| \cdot \Bigg[\tilde{f}(\varphi(x,t)) - \tilde{g}(\varphi(x,t)) + \\ &\qquad \left(t\nabla\tilde{f}(\varphi(x,t)) + (1-t)\nabla\tilde{g}(\varphi(x,t))\,,\, \frac{\partial}{\partial t}\varphi(x,t)\right)\Bigg] \\ &\overset{(9)}{=} |J\varphi(x,t)| \cdot \Bigg[\mathrm{div}\, \eta(\varphi(x,t),t)\left[t\tilde{f}(\varphi(x,t)) + (1-t)\tilde{g}(\varphi(x,t))\right] \\ &\qquad +\tilde{f}(\varphi(x,t)) - \tilde{g}(\varphi(x,t)) \\ &\qquad + \left(t\nabla\tilde{f}(\varphi(x,t)) + (1-t)\nabla\tilde{g}(\varphi(x,t))\,, \eta(\varphi(x,t),t)\right)\Bigg] \\ &\overset{(29)}{=} |J\varphi(x,t)|\left[\mathrm{div}\, v(\varphi(x,t)) + \tilde{f}(\varphi(x,t)) - \tilde{g}(\varphi(x,t))\right] \overset{(8)}{=} 0 \end{aligned}$$

□

References

[1] Th. Apel, S. Grosman, P. K. Jimack, and A. Meyer. A new methodology for anisotropic mesh refinement based upon error gradients. Preprint, Fak. für Mathematik, Techn. Universität Chemnitz, 09107 Chemnitz, Germany, 2001.

[2] Ch. Becker, S. Kilian, and S. Turek. Hardware–oriented numerics and concepts for pde software. In *FUTURE 1095*, pages 1–23. Elsevier, 2003. International Conference on Computational Science ICCS2002, Amsterdam.

[3] R. Becker and R. Rannacher. An optimal control approach to a posteriori error estimates in finite element methods. In *Acta Numerica*, pages 1–102. Cambridge University Press, 2001.

[4] P. B. Bochev, G. Liao, and G. C. de la Pena. Analysis and computation of adaptive moving grids by deformation. *Numerical Methods for Partial Differential Equations*, 12:489ff, 1996.

[5] J. U. Brackbill and J. S. Saltzman. Adaptive zoning for singular problems in two dimensions. *J. Comput. Phys.*, 46:342–368, 1982.

[6] J. Burkardt, M. D. Gunzburger, and J. Peterson. Insensitive functionals, inconsistent gradients, spurious minima , and regularized functionals in flow optimization problems. *International Journal of Computational Fluid Dynamics*, 16:171–185, 2002. DOI: 10.1080/10618560290034663.

[7] X.-X. Cai, D. Fleitas, B. Jiang, and G. Liao. Adaptive grid generation based on least–squares finite–element method. *Computers and Mathematics with Applications*, 48(7-8):1077–1086, 2004.

[8] T. Cao, Huang, and Russell. A study of monitor functions for two–dimensional adaptive mesh generation. *SIAM Journal on Scientific Computing*, 20(6):1978–1994, 1999.

[9] W. Cao, W. Huang, and R. D. Russell. Approaches for generating moving adaptive meshes: location versus velocity. *Applied Numerical Mathematics*, 47:121–138, 2003. DOI: 10.1016/S0168-9276(03)00061-8.

[10] G. F. Carey. *Computational Grids: Generation, Adaptation, and Solution Strategies.* Taylor und Francis, 1997.

[11] B. Dacorogna and J. Moser. On a partial differential equation involving jacobian determinant. *Annales de le Institut Henri Poincare*, 7:1–26, 1990.

[12] L. Formaggia, S. Micheletti, and S. Perotto. Anisotropic mesh adaption in computational fluid dynamics: Application to the advection–diffusion–reaction and the Stokes problems. Preprint, MOX, Modellistica e Calcolo Scientifico, Dipartimento di Mathematica "F. Brioschi", Politecnico di Milano, via Bornadi 9, I-20133 Milano, Italy, 2003.

[13] L. Formaggia and S. Perotto. Anisotropic error estimates for elliptic problems. *Numerische Mathematik*, 94:67–92, 2003. DOI 10.1007/s00211-002-0415-z.

[14] M. Grajewski. PhD thesis, Universität Dortmund, to appear 2006.

[15] T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003.

[16] J. Lang. *Adaptive Multilevel Solution of Nonlinear Parabolic PDE Systems.* Springer, Berlin, 2000. ISBN 3-540-67900-6.

[17] G. Liao and D. Anderson. A new approach to grid generation. *Applicable Analysis*, 44:285–298, 1992.

[18] G. Liao and B. Semper. A moving grid finite–element method using grid deformation. *Numerical Methods for Partial Differential Equations*, 11:603–615, 1995.

[19] F. Liu, S. Ji, and G. Liao. An adaptive grid method and its application to steady euler flow calculations. *SIAM Journal on Scientific Computing*, 20(3):811–825, 1998.

[20] Mukherjee. A hybrid, variational 3d smoother for orphaned shell meshes. In *Proceedings of the 11th International Roundtable, Sandia National Laboratories*, pages 379–390, `http://imr.sandia.gov`, 2002.

[21] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach.* Springer, Berlin, 1999.

[22] S. Turek, W. Decheng, and L. Rivkind. The fictitious boundary method for the implicit treatment of dirichlet boundary conditions with applications to incompressible flow simulation. In E. Bänsch, editor, *Challenges in Scientific Computing CISC 2002*, LNCSE, pages 37–68, Berlin, 2002. Springer, Berlin.

[23] S. Turek and M. Schäfer. Benchmark computations of laminar flow around cylinder. In E.H. Hirschel, editor, *Flow Simulation with High–Performance Computers II*, volume 52 of *Notes on Numerical Fluid Mechanics*, pages 547–566. Vieweg, 1996. co. F. Durst, E. Krause, R. Rannacher.

[24] K. E. Vugrin. *On the Effects of Noise on Parameter Identification Optimization Problems.* Phd thesis, Faculty of the Virginia Polytechnic Institute and State University, 2005.

[25] D. Wan and S. Turek. Direct numerical simulation of particulate flow via multigrid fem techniques and the fictitious boundary method. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 275, FB Mathematik, Universität Dortmund, November 2004. submitted to IGNMF 2005.

[26] A. Winslow. Numerical solution of the quasi–linear poisson–equation in a nonuniform triangle mesh. *J. Comput. Phys.*, 1:149–172, 1967.

[27] Z. Zhang. Polynomial preserving recovery for anisotropic and irregular grids. *J. Comput. Math.*, 22(2):331–340, 2004.