Journal of Computational and Applied Mathematics 36 (1991) 3–28
North-Holland

An adaptive finite-element strategy for the three-dimensional time-dependent Navier–Stokes equations

Eberhard Bänsch

Institut für Angewandte Mathematik, Universität Freiburg, Hermann-Herder-Straße 10, W-7800 Freiburg, Germany

Received 9 November 1990

Abstract

Bänsch, E., An adaptive finite-element strategy for the three-dimensional time-dependent Navier–Stokes equations, Journal of Computational and Applied Mathematics 36 (1991) 3–28.

An adaptive strategy for three-dimensional time-dependent problems in the context of the FEM is presented. The basic tools are a mechanism for local refinement and coarsening of simplicial meshes and an unexpensive error-estimator. The algorithm for local grid modification is based on bisecting tetrahedra. The method is applied to the Navier–Stokes equations.

Keywords: Adaptivity, local mesh refinement, Navier–Stokes equations.

Notation

$n = 2$ or 3 denotes the space dimension.

A *triangulation* $\mathcal{T}$ is a set of (nondegenerate) simplices in $\mathbb{R}^n$. A triangulation $\mathcal{T}$ is called *conforming*, if the intersection of two nondisjoint, nonidentical simplices consists either of a common vertex or a common edge or a common face.

$T \in \mathcal{T}$ is said to have a *nonconforming node*, if there is a vertex P of the triangulation which is not a vertex of T but $P \in T$.

Define a relation "$\leqslant$" for triangulations: $\mathcal{T}_1 \leqslant \mathcal{T}_2$ iff $\mathcal{T}_2$ is obtained by refinement of $\mathcal{T}_1$.

We call a sequence $\mathcal{T}_1, \mathcal{T}_2, \ldots$ *stable*, if all angles inside T are uniformly bounded away from 0 and π for all $T \in \bigcup_k \mathcal{T}_k$.

Throughout this paper we will assume $\Omega = \text{interior}(\bigcup \mathcal{T})$ and that the connected components of Ω are distinct.

$$L_0^2(\Omega) = L^2(\Omega) \cap \left\{ u \mid \int_\Omega u(x)\, \mathrm{d}x = 0 \right\},$$

$\overset{\circ}{H}{}^{1,2}(\Omega)$ is the usual Sobolev space with zero boundary values, $(\cdot, \cdot)$ is the inner product of $L^2(\Omega)$, $\langle \cdot, \cdot \rangle$ is the $H^{-1} \times H^1$ pairing, $\| \cdot \|$ is the L^2-norm, $\| \cdot \|_T$ as $\| \cdot \|$ but restricted to $T \subset \Omega$.

0. Introduction

Since the late 70s adaptive methods have been studied for different classes of PDEs (see, e.g., [1,3,4,10,12,13]). Especially in complex situations, such as flow simulation, these methods prove to be powerful tools for reducing computational cost dramatically.

The FEM fits well into the concept of adaptive techniques. Its great flexibility admits a wide range of different types of grids and function spaces. Modification of the local grid size (h-method), the local approximation order (p-method) as well as combinations of both are the most common approaches. Lately much attention was paid on applications for time-dependent problems.

Contrary to two-dimensional problems adaptive methods in three dimensions have not received adequate interest, although especially in this situation much gain can be expected. For instance, there are hardly any algorithms for local refinement of tetrahedral meshes.

In this paper some techniques for applying the h-method to transient laminar flow simulation in three dimensions are presented, in particular we develop a mechanism for local grid modification. The governing equations are the instationary, incompressible Navier–Stokes equations (NVS). The grid modification is based on simplicial grids which can approximate complex geometries very well. Furthermore tetrahedra turn out to be well suited in the adaptive context.

The described techniques were implemented as a FORTRAN code. In Section 5 numerical results show the success of our approach for the simulation of flows with moderate Reynolds numbers.

More precisely, we consider the Navier–Stokes equations governing the flow of an incompressible viscous fluid in a bounded region $\Omega \subseteq \mathbb{R}^3$:

$$\begin{aligned} &\partial_t u - \frac{1}{\text{Re}}\,\Delta u + u\cdot\nabla u + \nabla p = f, && \text{in } \Omega,\\ &\nabla\cdot u = 0, && \text{in } \Omega, \end{aligned} \tag{0.1}$$

where $u:\mathbb{R}_+\times\Omega\to\mathbb{R}^3$ is the flow velocity, $p:\mathbb{R}_+\times\Omega\to\mathbb{R}$ is the pressure, $f:\mathbb{R}_+\times\Omega\to\mathbb{R}^3$ is a density of external forces and Re is the Reynolds number.

Boundary and initial conditions have to be added:

$$u = g, \quad \text{on } \partial\Omega,$$

with

$$\int_{\partial\Omega} g\cdot\nu \,\mathrm{d}o = 0,$$

ν outward unit vector normal to $\partial\Omega$,

$$u(0,\cdot) = u_0, \quad \text{in } \Omega.$$

After time discretization by means of an appropriate scheme we get the following weak formulation:

$$\begin{aligned} &\text{For } k=1,2,\ldots \text{ find } u^k \in g + \left(\mathring{H}^{1,2}(\Omega)\right)^n,\ p^k \in L_0^2(\Omega) \text{ such that}\\ &\langle L(\Delta t, u^k, p^k), \varphi\rangle + (\nabla\cdot u^k, q) = \langle f(\Delta t, u^{k-1}), \varphi\rangle, \end{aligned} \tag{0.2}$$

for all $\varphi \in (\mathring{H}^{1,2}(\Omega))^n$, $q \in L_0^2(\Omega)$, where

$$u^k = u(k\,\Delta t,\cdot) \in g + \left(\mathring{H}^{1,2}(\Omega)\right)^n, \qquad p^k = p(k\,\Delta t,\cdot) \in L_0^2(\Omega),$$

with some nonlinear functionals L and f depending on the discretization scheme. Let us look at this scheme as a black box for the moment. In Section 4 an example for a time discretization is presented. The adaptive method will not depend too much upon the scheme.

To discretize (0.2) in space we choose function spaces V_h and W_h for velocity and pressure consisting of piecewise polynomials on a grid $\mathscr{T}$. We now want to control the error of the computed solution u_h^k by adjusting the grid $\mathscr{T}_k$ on each time level k. Because dealing with the error resulting from the space discretization is the harder task, we ignore the error due to time discretization for simplicity.

The fully discrete equations then have the form:

For $k = 1, 2, \ldots$ find $u_{h_k}^k \in g_{h_k} + V_{h_k}$, $p_{h_k}^k \in W_{h_k}$ such that

$$\langle L(\Delta t, u_{h_k}^k, p_{h_k}^k), \varphi_{h_k}\rangle + (\nabla \cdot u_{h_k}^k, q_{h_k}) = \langle f(\Delta t, u_{h_{k-1}}^{k-1}), \varphi_{h_k}\rangle, \tag{0.3}$$

for all $\varphi_{h_k} \in V_{h_k}$, $q_{h_k} \in W_{h_k}$, where V_{h_k} and W_{h_k} are function spaces on the mesh $\mathscr{T}_{h_k}$. For simplicity we will drop the subscript h_k and write u^k, V_k, etc. instead.

In the sequel basically two tools are needed. First we need a criterion where to insert and where to eliminate degrees of freedom from a grid. This will be done by locally estimating the (space-)error of the computational solution. Secondly we have to develop a mechanism for local modification of a grid.

The paper is organized as follows. In Section 1 we briefly describe some local error estimators. Section 2 deals with the second tool. In Section 3 we explain how to use these tools for the purpose of an adaptive strategy. Some details of the actual implementation can be found in Section 4. In the last section we discuss numerical results showing the efficiency of our approach.

1. Error estimators

Many proposals for estimating the error of a computational solution can be found in literature. We mention three classes of estimators for laminar flow problems.

(i) *Heuristic methods.* Make use of physically motivated quantities like gradients, vorticity, etc. to select regions where refinement is needed [6].

(ii) *Error estimators based on interpolation estimates.* Starting from an interpolation estimate like

$$\| D^l(u - I_h u) \| \leqslant C \| h^{k-1} D^k u \|, \quad l = 0, \ldots, k-1,$$

with some interpolation operator I_h, the term $D^k u$ is approximated numerically. For example, let $k = 2$ and substitute $D^2 u$ by

$$D_h^2 u_h := \sup_{x \in \partial T} \frac{[\partial_\nu u_h(x)]}{h(x)},$$

where u_h is the numerical solution, $[\partial_\nu u_h(x)]$ denotes the jump of the normal derivative across the boundary of the simplex T and h is the local grid size. In [7,9] this idea is used to develop error estimators and show their reliability.

We choose the third method.

(iii) *Error estimators based on the solution of local problems respectively estimation of residuals* [1,2,4,17]. Consider first the stationary Stokes equations:

$$\begin{aligned} -\Delta u + \nabla p &= f, && \text{in } \Omega,\\ \nabla\cdot u &= 0, && \text{in } \Omega,\\ u &= g, && \text{on } \partial\Omega. \end{aligned} \tag{1.1}$$

The standard variational formulation is given by

Find $[u, p] \in [g, 0] + H$ such that

$$A([u, p], [v, q]) := (\nabla u, \nabla v) - (p, \nabla\cdot v) - (q, \nabla\cdot u) = (f, v), \tag{1.2}$$

for all $[v, q] \in H := (\overset{\circ}{H}{}^{1,2}(\Omega))^n \times L_0^2(\Omega)$, and for the discrete problem

Find $[u_h, p_h] \in [g_h, 0] + H_h$ such that

$$A([u_h, p_h], [v_h, q_h]) = (f, v_h), \tag{1.3}$$

for all $[v_h, q_h] \in H_h := V_h \times W_h$.

To estimate the error we solve local problems of the same type with the residuals as right-hand side. Let $T \in \mathscr{T}$, find $[u_T, p_T] \in [u_h, 0] + H_T$ such that

$$A([u_T, p_T], [v_T, q_T]) = R_T([v_T, q_T]), \tag{1.4}$$

for all $[v_T, q_T] \in H_T := X_T \times Y_T$.

X_T and Y_T are spaces of functions on T that are of higher order than V_h and W_h in a suitable sense. For instance, if we consider the so called *Taylor–Hood element*, that means

$$V_h := \left(\{v \in C^0(\Omega) \mid v|_T \in \mathscr{P}_2\} \cap \overset{\circ}{H}{}^{1,2}(\Omega)\right)^n,$$
$$W_h := \{w \in C^0(\Omega) \mid w|_T \in \mathscr{P}_1\} \cap L_0^2(\Omega),$$

set

$$X_T := \left(\{v \in \mathscr{P}_3 \mid v(Q) = 0,\ Q \text{ a vertex or a midpoint}\}\right)^n,$$
$$Y_T := \operatorname{span}\{\lambda_i\lambda_j,\ 0 \leqslant i < j \leqslant n\},$$

where λ_i are the barycentric coordinates of T.

R_T denotes the local residual:

$$R_T([v_T, q_T]) := (P_0 f + \Delta u_h - \nabla p_h, v_T)_T + (q_T, \nabla\cdot u_h)_T + [[\partial_\nu u_h], v_T]_{\partial(T\cap\Omega)},$$

where

$$P_0 f := \sum_T \left(\frac{1}{|T|}\int_T f\right)\chi_T.$$

Now define the local error estimator ϑ_T:

$$\vartheta_T := \|\nabla u_T\|_T + \|p_T\|_T, \quad \text{for } T \in \mathscr{T}, \tag{1.5}$$

and the global error estimator ϑ:

$$\vartheta := \left(\sum_{T\in\mathscr{T}} \vartheta_T^2\right)^{1/2} \tag{1.6}$$

As the calculation of ϑ for three-dimensional problems may add substantially to the computational cost, we are interested in less expensive estimators. This can be done by simply estimating the residuals with an appropriate scaling factor. Define

$$\eta_T := h^{2-n}\Big(C_1 \,|T|\, \| P_0 f - \nabla p_h \|_T^2 + C_2 \sum_{\Gamma\in\partial T\cap\Omega} |\Gamma|\, \| [\partial_\nu u_h] \|_\Gamma^2 + C_3 \| \nabla\cdot u_h \|_T^2\Big)^{1/2} \tag{1.7}$$

and

$$\eta := \Big(\sum_{T\in\mathscr{T}} \eta_T^2\Big)^{1/2}. \tag{1.8}$$

Verfürth [17] showed the following.

Proposition 1.1. *There are constants c, C such that*

(i) $\| \nabla(u-u_h) \|^2 + \| p - p_h \|^2 \leqslant C\eta^2 + \sum_{T\in\mathscr{T}} |T|\, \| f - P_0 f \|_T^2,$

(ii) $c\vartheta_T \leqslant \eta_T \leqslant C\vartheta_T,$ *for all* $T\in\mathscr{T}$.

Remark. The proposition implies that ϑ and η are asymptotically equivalent. Using ϑ we would expect more precise quantitative information about the error. If the main objective is to control an automatic adaptive process, η may be preferred because of its lower cost.

In order to apply these techniques to the instationary Navier–Stokes equations we modify the residual by adding the time derivative and the nonlinearity, thus getting the following definition for η (ϑ is changed in an analogous way):

$$\begin{aligned}\eta_T := h^{2-n}\Bigg(&\mathrm{Re}\, C_1 \,|T|\, \Big\| P_0\big(f - \hat{\partial}_t u_h - u_h\cdot\nabla u_h\big) + \frac{1}{\mathrm{Re}}\,\Delta u_h - \nabla p_h \Big\|_T^2 \\ &+ C_2 \sum_{\Gamma\in\partial T\cap\Omega} |T|\, \| [\partial_\nu u_h] \|_\Gamma^2 + C_3 \| \nabla\cdot u_h \|_T^2\Bigg)^{1/2}\end{aligned} \tag{1.9}$$

($\hat{\partial}_t u_h$ is the discrete time derivative of u_h).

Remark. (i) C_1, C_2, C_3 have to be fitted with the help of some explicit solutions. However, if the focus is more on qualitative information, the definition of the constants is less critical.

(ii) Actually we use a slightly simplified version of (1.9):

$$\begin{aligned}h^{2(n-2)}\eta_T^2 := {}&\mathrm{Re}\, C_1 \,|T|\, \| P_0(f - u_h\cdot\nabla u_h) - \nabla p_h \|_T^2 \\ &+ C_2 \sum_{\Gamma\in\partial T\cap\Omega} |\Gamma|\, \| [\partial_\nu u_h] \|_\Gamma^2.\end{aligned}$$

Our experience showed that the dropped terms are of low numerical significance. The divergence of u is used as a stopping criterion for the iterations of the (NVS)-solver and the volume integrals are scaled by $|T|$ which vanishes faster than $|\Gamma|$.

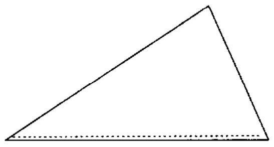
Fig. 2.1.

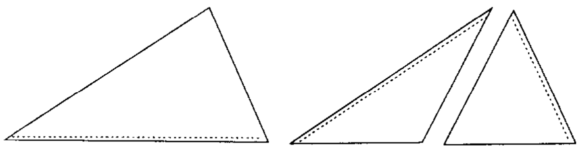
Fig. 2.2. Bisection of a single triangle.

2. Techniques for local grid modification

2.1. Local refinement

The local refinement of a tetrahedral mesh is studied in [5]. We follow the description given there.

There are many references about local grid refinement in two dimensions [3,6,10,14–17]. Most of the papers consider either dividing a triangle into two or into four new ones (where possible). Our strategy is based on the bisectioning of simplices. The 2-dimensional method is similar to that described in [14]. Unlike in the 2-d case there is hardly any algorithm for 3-d refinement. We introduce a refinement strategy which turns out to be a canonical generalization of the 2-d case. It can be applied to an arbitrary conforming initial triangulation.

For convenience we briefly describe the 2-d case first and then deal with the triangulation in $\mathbb{R}^3$.

2.1.1. The two-dimensional case

The "philosophy" behind our approach is to look at the situation as local as possible. Let us first consider a single triangle $T \in \mathbb{R}^2$. Mark an arbitrary edge, called the *refinement edge*, see Fig. 2.1.

If T has to be divided this will always be done by cutting through the midpoint of the refinement edge and the vertex opposite to the refinement edge. We get two new triangles. The position of the new refinement edges are as shown in Fig. 2.2. This leads to the following algorithm for (local) refinement of a given conforming triangulation $\mathcal{T}_k$ into $\mathcal{T}_{k+1}$.

2-d algorithm. Let each $T \in \mathcal{T}_k$ have one refinement edge and let Σ be the set of those triangles, which have to be divided.

(1) Bisect each $T \in \Sigma$ as described above. Let $\hat{\mathcal{T}}_k$ be the resulting (possibly nonconforming) triangulation.

(2) Let now Σ be the set of those triangles with a nonconforming node.

(3) If $\Sigma = \emptyset$, set $\mathcal{T}_{k+1} := \hat{\mathcal{T}}_k$ and stop. Otherwise go to (1).

Figure 2.3 shows an example $\mathcal{T}_0 \to \mathcal{T}_1$ with initial $\Sigma := \mathcal{T}_0$.

Proposition 2.1. *The algorithm stops in a finite number of steps, $\mathcal{T}_{k+1}$ is conforming and the sequence $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2, \ldots$ is stable.*

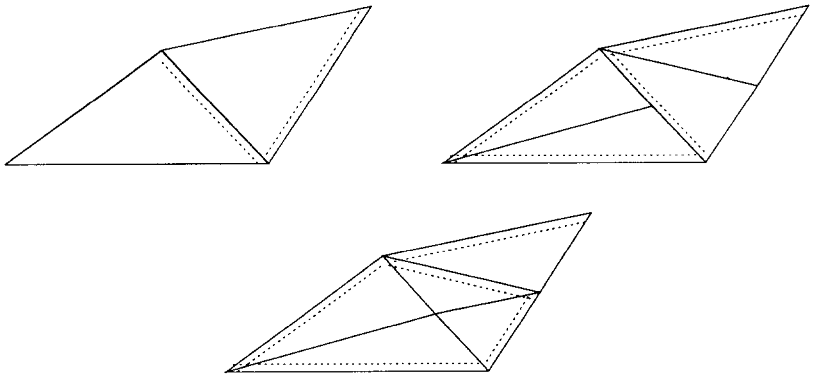

Fig. 2.3. $\mathcal{T}_0$, $\hat{\mathcal{T}}_0$ and $\mathcal{T}_1$.

Proof. The proof is very simple and can be found in [5]. □

Remark. Note that we do not require any compatibility condition for the initial position of the refinement edges of neighbouring triangles. A good choice would be the longest edge of each triangle.

2.1.2. The three-dimensional case

Let $\mathcal{T}$ be a conforming triangulation, consisting of tetrahedra $T \subseteq \mathbb{R}^3$. The main idea of the algorithm presented here is to divide T by bisection such that the faces of T are divided as in the 2-dimensional case.

Consider a tetrahedron $T \in \mathcal{T}$, which has been cut open along the three edges which meet at vertex P_4 and unfolded. Assume that there is one refinement edge in each of the four triangular faces of T (one example is shown in Fig. 2.4). We make the following assumptions.

(A1) For each tetrahedron there is at least one common refinement edge for two different faces of the tetrahedron, which meet at this edge ($P_{i_1}P_{i_3}$ in Fig. 2.4). We call such an edge a *global refinement edge* of the tetrahedron.

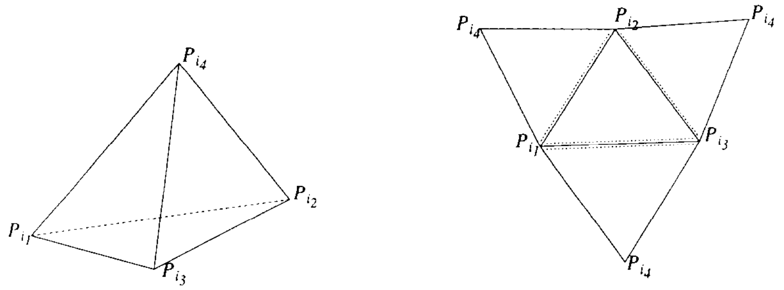

Fig. 2.4.

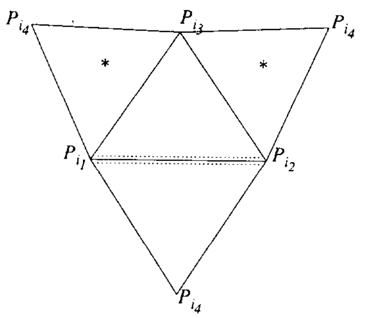

Fig. 2.5. " * " means an arbitrary distribution of the refinement edges.

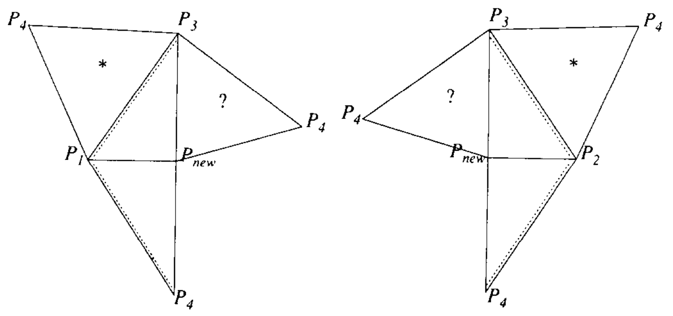

Fig. 2.6.

(A2) If T_1, T_2 are two tetrahedra with $T_1 \cap T_2 = S$ and S is a triangle, then the refinement edge of S with respect to T_1 and the refinement edge of S with respect to T_2 is the same.

Note that assumptions (A1) and (A2) can be fulfilled for an arbitrary conforming triangulation: by small (hypothetical) perturbations of the coordinates of the nodes one can achieve that there is exactly one longest edge for each triangle of the triangulation. Take this as the refinement edge of the triangle. Then (A1) and (A2) are fulfilled. Just as in the two-dimensional case we first look at a single tetrahedron, see Fig. 2.5.

Remark. The representation in Fig. 2.5 is unique if there is exactly one global refinement edge $\overline{P_{i_1}P_{i_2}}$ and if we require, e.g., $i_1 < i_2$ in addition. This representation is called *standard position*.

For simplicity we drop the subscript i and write P_1, $P_2, \ldots$ instead.

If there is a global refinement edge, we are allowed to bisect T by adding a new point P_{new}, the midpoint of $\overline{P_1P_2}$ and cut T along P_3P_{new} and $P_{\text{new}}P_4$. We get two new tetrahedra, see Fig. 2.6.

The refinement edges of the bisected triangles are chosen as in the 2-dimensional case. The only question is, how to choose the refinement edge for the new triangle $P_{\text{new}}P_4P_3$. One requirement is of course the condition that both new tetrahedra must have again a global refinement edge.

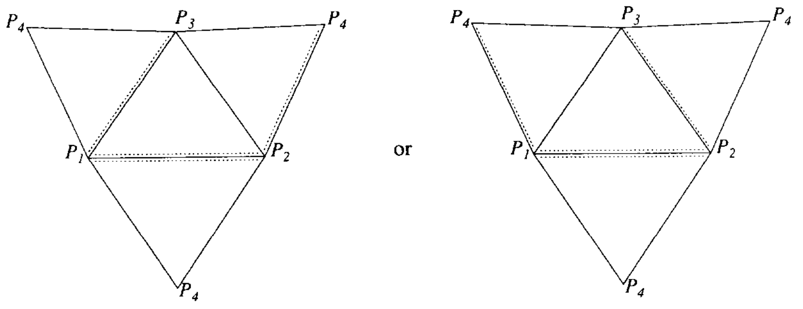

Fig. 2.7.

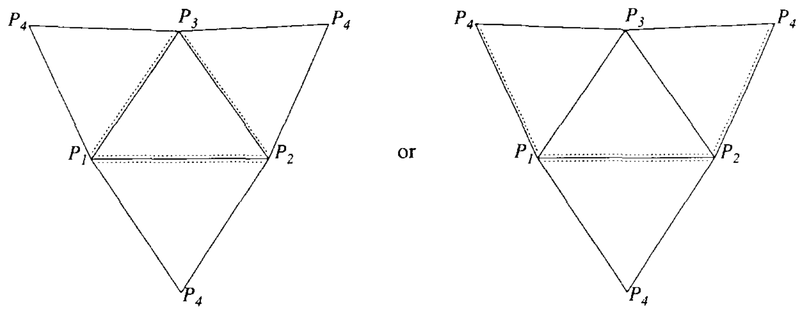

Fig. 2.8.

We present a mechanism which does not only fulfil this basic assertion but also generates a very regular structure (see, for example, Fig. 2.11). For this we look at a special situation for the moment.

Definition 2.2. A tetrahedron T is called *red*, if it has the distribution of refinement edges as shown in Fig. 2.7. It is called *black* in case of the situation shown in Fig. 2.8.

Note that for red and black tetrahedra assumption (A1) is fulfilled automatically.

In [5] is outlined how to choose the refinement edge for the new triangle, such that the following holds: if T is red, then its children are black. If a black tetrahedron T is divided and its father was red, then the children of T are again black. On the other hand, if T is black and its father was also black, then the children are red. For successive bisection we hence have the cycle

$$\text{red} \rightarrow \text{black} \rightarrow \text{black} \rightarrow \text{red}.$$

Now consider the case that a tetrahedron is neither red nor black. Again there is a position of the refinement edge of the new triangle such that the children are red or black (see [5] for details). So without loss of generality we only have red or black tetrahedra.

After having described how to handle the local situation, we now present the global algorithm, which is in fact identical with the two-dimensional one.

3-d algorithm. Let $\Sigma \subseteq \mathcal{T}_k$ be the set of tetrahedra to be divided.

(1) Bisect each $T \in \Sigma$ as described above. Let $\hat{\mathcal{T}}_k$ be the resulting (possibly nonconforming) triangulation.

(2) Let now Σ be the set of those triangles with a nonconforming node.

(3) If $\Sigma = \emptyset$, set $\mathcal{T}_{k+1} := \hat{\mathcal{T}}_k$ and stop. Otherwise go to (1).

We state the basic properties.

Proposition 2.3. *Let the conforming triangulation $\mathcal{T}_k$ fulfil assumptions* (A1) *and* (A2). *Then the above algorithm stops in a finite number of steps and $\mathcal{T}_{k+1}$ is conforming. The sequence $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2, \ldots$ is stable.*

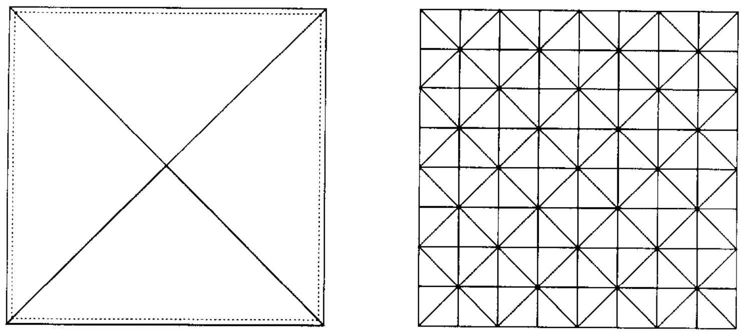

Fig. 2.9. $\mathcal{T}_0$, $\mathcal{T}_5$.

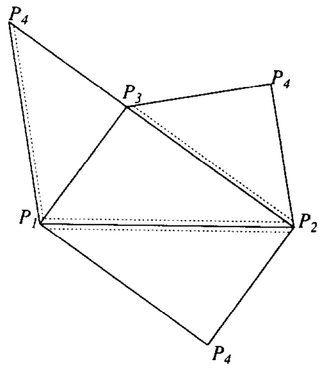

Fig. 2.10.

Proof. See [5]. □

2.1.3. "Standard" triangulations

One can easily check that the 2-d as well as the 3-d algorithm generates a regular structure inside a macrosimplex. More precisely, simplices obtained by refining one simplex l times are geometrically similar to those obtained after $l + n$ refinement steps.

Furthermore, there are triangulations which are globally consistent with this structure. Consider in two dimensions the triangulations shown in Fig. 2.9.

In 3-d we start with a subdivision of a cube consisting of six tetrahedra, each geometrically similar to the situation shown in Fig. 2.10, with $P_1 = (0, 0, 0)$, $P_2 = (1, 1, 1)$, $P_3 = (1, 0, 0)$ and $P_4 = (1, 0, 1)$. The macrotriangulation of the cube and a refined mesh is shown in Fig. 2.11.

Definition 2.4. Consider a macrotriangulation which can be transformed into one of the triangulations shown in Fig. 2.9 and 2.11; this means that there is a mapping

$$\mathcal{H} : \mathbb{R}^n \rightarrow \mathbb{R}^n$$

and $\mathcal{H}\mathcal{T}_0$ is a subtriangulation of one of the triangulations in Figs. 2.9 and 2.11 (including the position of the refinement edges). Every $\mathcal{T}_k$ that is obtained by refining $\mathcal{T}_0$ is called *standard triangulation*.

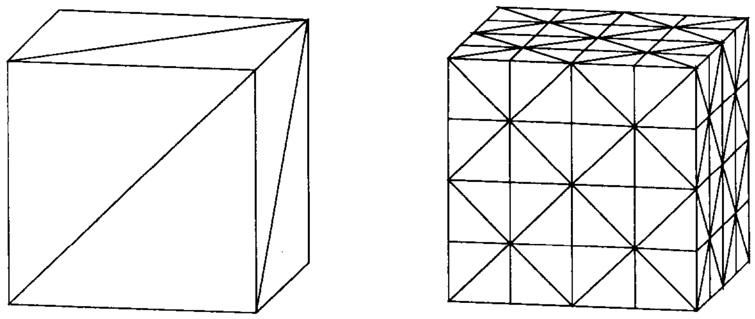

Fig. 2.11. $\mathcal{T}_0$, $\mathcal{T}_6$.

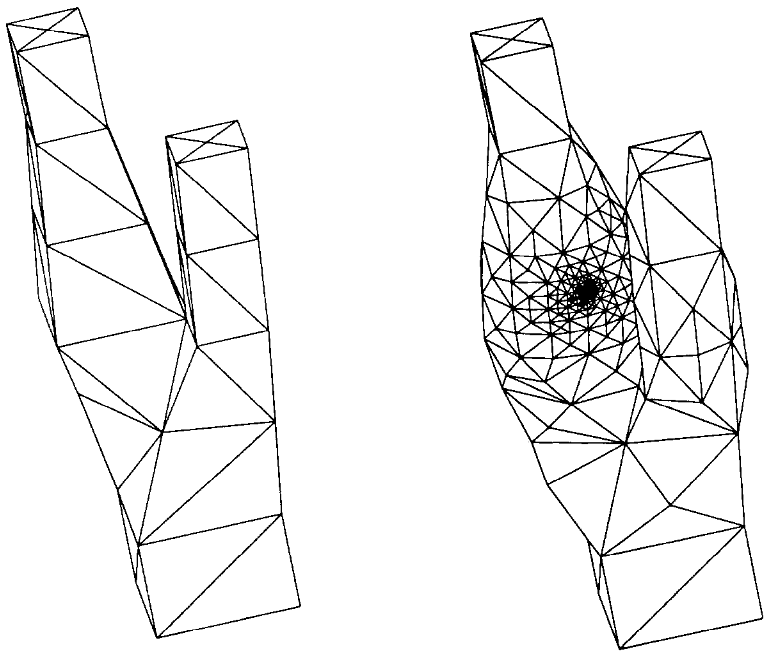
Fig. 2.12. Macrotriangulation and locally refined mesh.

An example of a locally refined triangulation obtained from a quite unregular macrotriangulation is shown in Fig. 2.12.

2.2. Local coarsening

As a fundamental concept we only consider coarsening such that refined simplices are coarsened in the same manner as they were refined, i.e., two "brothers" are melted into their father or in other words we just go back in the binary tree generated by the refinement process.

Let us begin with the remark that an adaptive algorithm should not permit the coarsening of any situation without certain restrictions. To see this, consider Fig. 2.13.

The solution of a problem might have required the refinement into the right corner of T as indicated. If some local criterion leads to a coarsening of T', all previously refined triangles have

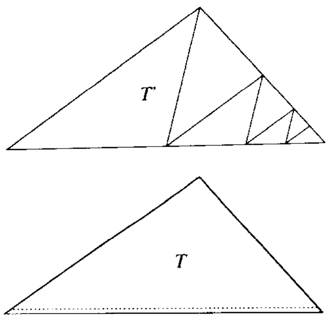

Fig. 2.13.

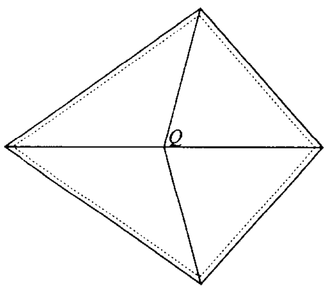

Fig. 2.14.

to be eliminated before. This "global" effect is of course not intended. To overcome this difficulty we look for configurations of simplices which permit to keep the process of coarsening local.

We make the following definitions.

Definition 2.5. (i) A simplex $T \in \mathscr{T}$ has *level l* if T was obtained after l refinement steps.

(ii) A simplex T is said to have *locally finest level* if the levels of all neighbours are less than or equal to the level of T.

(iii) Let $T \in \mathscr{T}$ and let T' be the father of T. The vertex P which was inserted while bisecting T' is called the *coarsening node* of T.

(iv) Let K be an edge of the triangulation $\mathscr{T}$ and K' the "father"-edge of K with midpoint Q. Set $M := \{T \in \mathscr{T} \mid T \cap K' \neq \emptyset\}$. If Q is the coarsening node for all $T \in M$, then M is called a *resolvable patch*.

Figure 2.14 shows a resolvable patch in two dimensions.

Remark. If M is a resolvable patch, all $T \in M$ can be coarsened without effecting any $T' \in \mathscr{T}$ outside M.

Resolvable patches are the configuration which we allow to be coarsened. This guarantees that the coarsening process stays local. But now the question arises whether there are "enough" resolvable patches in an arbitrary triangulation. The following can easily been shown.

Lemma 2.6. (i) *Let $\mathscr{T}$ be a refinement of a standard triangulation. If $T \in \mathscr{T}$ has locally finest level, then T belongs to a resolvable patch.*

(ii) *At least local structures in the interior of macrosimplices of an arbitrary triangulation can always be totally coarsened.*

Remark. Actually there are situations where a refined triangulation cannot be coarsened into its macrotriangulation, see Fig. 2.15. In this example refined triangles at the node where macrosimplices meet cannot be eliminated.

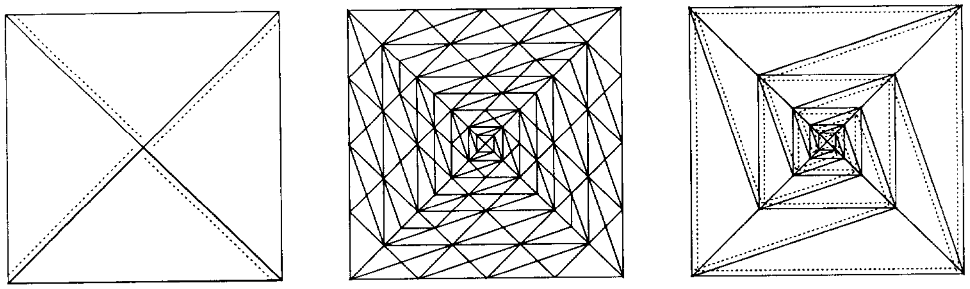
Fig. 2.15. $\mathscr{T}_0$, refined triangulation and final triangulation after coarsening.

3. Self-adaptive strategy

In this section we describe how the tools introduced above are used to adapt the mesh during the transient process. Because three-dimensional flow calculation requires a huge amount of storage — even for locally refined grids — only the triangulation for the current time step is kept in storage. Thus we also develop some special procedures, for instance to interpolate between grids of different time levels.

As already pointed out we want to apply the h-method on each time level. So the problem can be stated as follows. Given a certain tolerance $\epsilon > 0$, find the $\mathcal{T}_k$, k the kth time level, with fewest degrees of freedom, such that the estimation of the error η fulfils

$$\eta = \left(\sum_{T \in \mathcal{T}_k} \eta_T(u^k)^2 \right)^{1/2} \leqslant \epsilon.$$

We shall not attempt to solve this optimization problem exactly. It is sufficient to obtain grids which are in a certain sense nearly optimal. A reasonable criterion to establish a "good" mesh is the equidistribution of the error

$$\eta_T(u^k) \approx \frac{1}{\sqrt{N_k}}, \quad \text{with } N_k = \#\mathcal{T}_k.$$

This leads to the following criteria for refining or coarsening the mesh.

3.1. *Criteria for modifications of the grids*

Given k and $T \in \mathcal{T}_k$:

$$\text{if } \eta_T(u^k) \geqslant \frac{\theta_1 \epsilon}{\sqrt{N_k}}, \text{ refine } T\ m \text{ times,} \tag{3.1}$$

$$\text{if } \eta_T(u^k) \leqslant \frac{\theta_2 \epsilon}{\sqrt{N_k}}, \text{ coarsen } T\ r \text{ times,} \tag{3.2}$$

where $0 < \theta_2 < \theta_1 \leqslant 1$ are constants, $m, r \in \mathbb{N}$. We choose $m = r = 1$. Of course some compatibility conditions have to be added to insure that regions where nodes are eliminated do not intersect regions where nodes have been inserted. We actually permit a *resolvable patch* to be coarsened iff

(i) there is no T in the patch which fulfils (3.1),
(ii) there is at least one T which fulfils (3.2).

$\mathcal{T}_{k+1}$ can now be determined by an iterative process. Make a guess $\mathcal{T}_{k+1}^0$ for the new mesh (e.g., $\mathcal{T}_{k+1}^0 = \mathcal{T}_k$) and compute u_0^{k+1} on $\mathcal{T}_{k+1}^0$. Then $\mathcal{T}_{k+1}^1$ is obtained by applying (3.1) and (3.2) on $\mathcal{T}_{k+1}^0$ via evaluating $\eta(u_0^{k+1})$. Repeat this procedure until $\mathcal{T}_{k+1}^l$ is satisfactory and set $\mathcal{T}_{k+1} = \mathcal{T}_{k+1}^l$.

However, it is very expensive to compute each u_l^{k+1}. Provided that the time step Δt is small compared to the velocity of the local error and the corresponding local mesh size,

$$\Delta t \leqslant h(x) \mid u_{\text{error}}(x) \mid$$

($u_{\text{error}}(x)$ is the velocity of a "local perturbation"), we can choose a less expensive strategy. Given $\mathcal{T}_k$ and u^k we obtain $\mathcal{T}_{k+1}$ by simply applying (3.1) and (3.2) to $\mathcal{T}_k$ once.

Remark. The choice of θ_1 and θ_2 permits to influence the "character" of the triangulation. Small values of θ_2/θ_1 will yield more uniform grids whereas large values will create steeper gradients of the local grid size $h(x)$.

3.2. Interpolation between $\mathscr{T}_k$ and $\mathscr{T}_{k+1}$

Consider a time step k and the solution u^k on $\mathscr{T}_k$. The right-hand side $(f_i^k)_{i=1,\dots,N_k}$ of (0.3) for the new time step $k+1$ is given by

$$f_i^k = \langle f(u^k), \varphi_i^{k+1}\rangle,$$

where for instance $(\varphi_i^{k+1})_i$ is the nodal basis of a discrete subspace V_{k+1} of $(\mathring{H}^{1,2}(\Omega))^n$ on the grid $\mathscr{T}_{k+1}$. If $\mathscr{T}_{k+1} \geqslant \mathscr{T}_k$, there is a canonical representation of u^k on $\mathscr{T}_{k+1}$ by prolongation and $(f_i^k)_i$ can be calculated as usual.

Consider the case where $\mathscr{T}_{k+1} \leqslant \mathscr{T}_k$. This means that $f(u^k)$ is given on a finer grid than the test functions φ_i^{k+1}. We introduce the notation of the hierarchical basis ψ_i, $i = 1,\dots, N_k$. Let $\hat{\mathscr{T}}_l$, $l = 0,\dots, r$, be the conforming intermediate triangulations between $\mathscr{T}_k$ and $\mathscr{T}_{k+1}$ with $\hat{\mathscr{T}}_r = \mathscr{T}_k$ and $\hat{\mathscr{T}}_0 = \mathscr{T}_{k+1}$. The discrete function spaces on $\hat{\mathscr{T}}_l$ are denoted by $\hat{V}_l$. Let the enumeration of the nodes be such that

$$P_1,\dots, P_{M_l} \text{ are the nodes of } \hat{\mathscr{T}}_l.$$

Then

$$\psi_i := \hat{\varphi}_i^l \in \hat{V}_l, \quad \text{if } M_{l-1} < i \leqslant M_l,$$

and $\hat{\varphi}_j^l$, $j = 1,\dots, M_l$, is the nodal basis of $\hat{V}_l$. $u \in \mathscr{T}_k$ can be represented in terms of φ_i^k and ψ_i:

$$u = \sum_{j=1}^{N_k} u_j \varphi_j^k = \sum_{j=1}^{N_k} u_j^* \psi_j .$$

Note that $\varphi_i^{k+1} = \psi_i$. Let $S \in \mathbb{R}^{N_k \times N_k}$ be the matrix transforming $(u_i^*)_i$ into $(u_i)_i$:

$$\sum_j S_{ij} u_j^* = u_i .$$

Define $f_i := \langle f(u^k), \varphi_i^k\rangle$, $i = 1,\dots, N_k$, with test functions φ_i^k on the fine grid, which can be computed as usual and $f_i^* := \langle f(u^k), \psi_i\rangle$. Then we have

$$S^{\mathrm{T}} f = f^*,$$

and particularly,

$$(S^{\mathrm{T}} f)_i = \langle f(u^k), \psi_i\rangle = \langle f(u^k), \varphi_i^{k+1}\rangle, \tag{3.3}$$

for $i \leqslant N_{k+1}$.

Remark. There is a factorization for $S^{\mathrm{T}} = S_1^{\mathrm{T}} \cdots S_r^{\mathrm{T}}$ where S_l^{T} can easily be computed on each intermediate triangulation $\hat{\mathscr{T}}_l$ (see [18] for details).

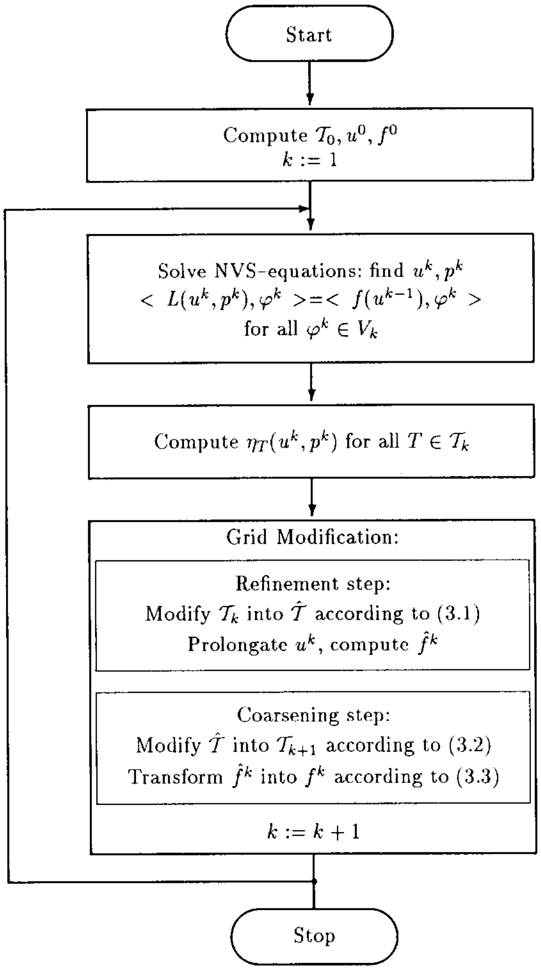
Start
Compute $\mathcal{T}_0, u^0, f^0$
$k := 1$
Solve NVS-equations: find u^k, p^k
$< L(u^k,p^k), \varphi^k > = < f(u^{k-1}), \varphi^k >$
for all $\varphi^k \in V_k$
Compute $\eta_T(u^k,p^k)$ for all $T \in \mathcal{T}_k$
Grid Modification:
Refinement step:
Modify $\mathcal{T}_k$ into $\hat{\mathcal{T}}$ according to (3.1)
Prolongate u^k, compute $\hat{f}^k$
Coarsening step:
Modify $\hat{\mathcal{T}}$ into $\mathcal{T}_{k+1}$ according to (3.2)
Transform $\hat{f}^k$ into f^k according to (3.3)
$k := k + 1$
Stop

Fig. 3.1.

3.3. Self-adaptive algorithm

Now we are able to present the algorithm for the time-dependent adaptive strategy, see Fig. 3.1.

4. Description of the implementation

In this section we describe the "black box" of the Navier–Stokes solver and mention some details of the actual implementation which are not straightforward.

4.1. Time discretization

We follow a proposal of [6] and use the so-called θ-scheme. Let $u^0 = u(0, \cdot)$; then

$$\frac{u^{k+\theta} - u^k}{\theta\,\Delta t} - \frac{\alpha}{\mathrm{Re}}\,\Delta u^{k+\theta} + \nabla p^{k+\theta} = f^{k+\theta} + \frac{\beta}{\mathrm{Re}}\,\Delta u^k - (u^k \cdot \nabla)u^k, \quad \text{in } \Omega,$$
$$\nabla \cdot u^{k+\theta} = 0, \quad \text{in } \Omega, \qquad u^{k+\theta} = g^{k+\theta}, \quad \text{on } \partial\Omega, \tag{4.1}$$

$$\frac{u^{k+1-\theta} - u^{k+\theta}}{(1-2\theta)\,\Delta t} - \frac{\beta}{\mathrm{Re}}\,\Delta u^{k+1-\theta} + (u^{k+1-\theta} \cdot \nabla)u^{k+1-\theta}$$
$$= f^{k+1-\theta} + \frac{\alpha}{\mathrm{Re}}\,\Delta u^{k+\theta} - \nabla p^{k+\theta}, \quad \text{in } \Omega,$$
$$u^{k+1-\theta} = g^{k+1-\theta}, \quad \text{on } \partial\Omega, \tag{4.2}$$

$$\frac{u^{k+1} - u^{k+1-\theta}}{\theta\,\Delta t} - \frac{\alpha}{\mathrm{Re}}\,\Delta u^{k+1} + \nabla p^{k+1}$$
$$= f^{k+1} + \frac{\beta}{\mathrm{Re}}\,\Delta u^{k+1-\theta} - (u^{k+1-\theta} \cdot \nabla)u^{k+1-\theta}, \quad \text{in } \Omega,$$
$$\nabla \cdot u^{k+1} = 0, \quad \text{in } \Omega, \qquad u^{k+1} = g^{k+1}, \quad \text{on } \partial\Omega, \tag{4.3}$$

for all $k \geqslant 1$, with $\theta = 1 - \frac{1}{2}\sqrt{2}$ and $\alpha = (1-2\theta)/(1-\theta)$, $\beta = \theta/(1-\theta)$.

By (4.1)–(4.3) the main problems in solving NVS numerically are uncoupled, namely the treatment of the divergence condition and the highly nonlinear structure. Formulas (4.1) and (4.3) are Stokes-like problems, whereas (4.2) is a nonlinear system without solenoidal constraint. In [6] efficient tools for solving these subproblems are introduced. The Stokes-like equations are transformed into operator equations for the pressure with a positive definite, self-adjoint operator. These transformed equations can easily be resolved by an abstract version of the conjugate gradient method via a cascade of *scalar* equations of the form $u - \tau\,\Delta u = f$.

Formula (4.2) is treated by a nonlinear conjugate gradient method with an appropriate preconditioning. Again the actual numerical work is done by solving a cascade of scalar equations of the above type (see [6] for details).

One of the most fundamental requirements on the scheme in connection with an adaptive approach is the unconditional stability. A stability condition of the form

$$\Delta t \leqslant \min_{x \in \Omega} h(x)^{\alpha},$$

with some $\alpha \geqslant 1$, would lead to inacceptable time step lengths.

Stability and convergence for a slightly modified version of (4.1)–(4.3) are shown in [8]. However, only conditional stability is proved there. Nevertheless, numerical experiments show that the scheme behaves well. We gained the experience that for given $h_0 > 0$ there is a Δt_0 such that (4.1)–(4.3) is stable for all $h(x) \leqslant h_0$ and $\Delta t \leqslant \Delta t_0$ at least for moderate Reynolds numbers.

Remark. With the time discretization (4.1)–(4.3) the right-hand side of (0.2) becomes (without external forces)

$$\langle f(\Delta t, u^{k-1}), \varphi \rangle = \frac{1}{\theta\,\Delta t}(u^{k-1}, \varphi) + \frac{\beta}{\mathrm{Re}}(\nabla u^{k-1}, \nabla\varphi) - (u^{k-1} \cdot \nabla u^{k-1}, \varphi).$$

4.2. Space discretization

We use the so-called *Taylor–Hood element* with piecewise quadratic, globally continuous velocities and piecewise linear, globally continuous pressures. This combination is stable with regard to the Babuška–Brezzi condition [11].

4.3. The implementation

Although nowadays there are powerful software tools for handling complicated data structures, we use FORTRAN as programming language. This guarantees the portability of the code and permits the profitable use of vector and parallel hardware.

Furthermore, multi-level data structures, which could be implemented even in FORTRAN, might make life much easier. The drawback of those techniques consists of higher storage requirements. For three-dimensional problems, storage becomes very expensive. We therefore use a data structure based on arrays (that has a high vectorization potential) and keep only information about the triangulation of the current time step. As already mentioned in Section 3 this requires some additional techniques, for instance the coarsening procedure and the interpolation formula.

In this section we sketch some details how the adaptive strategy introduced in Section 3.3 was actually implemented. The following code may be the body of a FORTRAN main program representing the algorithm:

```
 DO 1 K=1,N_TIME_LEVELS
      CALL REFINE(U,P,REF_FLAG)
      CALL CONNEC
      CALL RIGHT_HAND_SIDE(U,F)
      CALL DEREF(F,REF_FLAG)
      CALL COMPR
      CALL CONNEC
      CALL MATRIX
      CALL THETA_SCHEME
      CALL ERROR_ESTIMATOR(U,P,REF_FLAG)
1 CONTINUE
```

where NT is the number of tetrahedra, NV is the number of vertices, NK is the number of nodes, U(3,NK) is the velocity field, F(3,NK) is the right-hand side, P(NV) is the pressure and REF_FLAG(NT) is the flag, whether a tetrahedron has to be divided, coarsened or unchanged.

In the sequel we explain the various subroutines.

REFINE

REFINE executes the refinement step. The basic tool in this routine is the procedure which bisects a single tetrahedron. This is done in the following way.

- Let T be a tetrahedron with number K represented in standard position as shown in Fig. 4.1. (The representation in standard position simplifies many manipulations on the simplex.)
- Set NT := NT+1, bisect T and pass all information from T to its "children". The new tetrahedra get the numbers K (left one) and NT (right one).

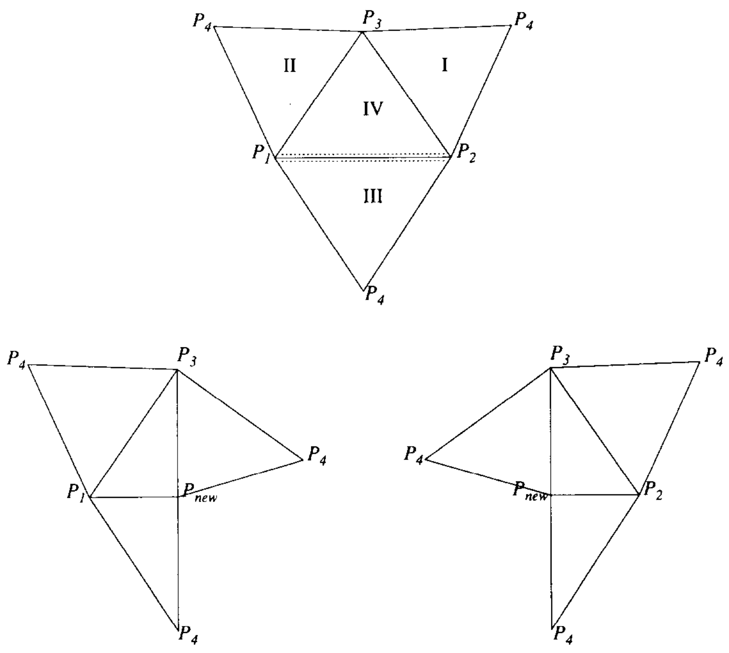

Fig. 4.1. A tetrahedron T in standard position and its "children".

- Due to the local design of the algorithm, exchange of information with adjacent tetrahedra is needed only in three cases (for notation see Fig. 4.1).
  - (i) The neighbour of T at face I has to be told that the number of its neighbour is now **NT**.
  - (ii) We need to check whether P_{new} already exists. This might be the case if a tetrahedron which lies at edge $\overline{P_1P_2}$ has been bisected before. To trace that, we store some information about edges.
  - (iii) There might be "conflicts" with the neighbours at face III and IV which arise from nonconforming nodes. The vectors keeping this information have to be updated.
- Finally the new tetrahedra **K** and **NT** are transformed into standard position.

Remark. Because we use quadratic elements for the velocity, not only vertices but also the midpoints of edges have to be taken into account during the grid modification. They carry information (of the old time step) which is needed, e.g., while evaluating the right-hand side. It turns out that we cannot enumerate the nodes for the velocity space in such a way that $P_1, \ldots, P_{\text{NV}}$ are the vertices of the triangulation. On the other hand, the nodes for the pressure space, say $P_{1'}, \ldots, P_{\text{NV}'}$, are just the vertices of the triangulation. Two permutation vectors are needed to provide the mapping between the two enumerations.

CONNEC, MATRIX

CONNEC generates the connectivity for the nodes of the pressure and velocity space. In **MATRIX** the stiffness matrices are assembled. The connectivity and matrix for the pressure space are stored in arrays as:

```
NEIGHBOUR_P(NKP,IBAND)
A_P(NKP,IBAND),
```

where **IBAND** is the maximal number of neighbours of a vertex.

Storing the information for the velocity space in the same way would require an inacceptable amount of memory. Because we use quadratic elements, the bandwidth is quite high. Furthermore, the hull of the matrix is irregular. Even for the standard triangulation the maximal number of nonzero elements is 124 whereas the average is 26. Therefore we want to store the elements of the matrix in a linear order. That means we have vectors of the form

```
NEIGHBOUR(NK*IBAND_AVERAGE)
A(NK*IBAND_AVERAGE).
```

In order to get vectorizable code, we store the elements of a row consecutively. For that purpose we use the vector

```
IPOINT(NK).
```

The elements $a_{i,j}$ are represented by

```
A(IPOINT(I)),...,A(IPOINT(I+1)-1).
```

To assign values to the vector **IPOINT** we a priori have to know the number of nonzero elements for each node. If we have this information for the pressure space (note that the nodes for the pressure space are just the vertices), we are able to calculate this number by means of the Euler formula

$$\#\text{vertices} - \#\text{edges} + \#\text{triangles} = 2 - \#\text{holes}, \tag{4.4}$$

for a triangulation of a closed 2-d manifold in $\mathbb{R}^3$. Let P_i be a node of our triangulation and $\mathsf{NT}_{\text{loc}} := \#\{T \in \mathcal{T} \mid P_i \in T\}$. Note that NT_{loc} can easily be calculated. Denote by q the number of neighbours of the node P_i. With the help of (4.4) it is obvious that:

if P_i is a vertex, then

$$q = 3\mathsf{NV}_{\text{loc}} + \mathsf{NT}_{\text{loc}} - 2, \quad \text{if } P_i \in \mathring{\Omega},$$
$$q = 3\mathsf{NV}_{\text{loc}} + \mathsf{NT}_{\text{loc}} - 1, \quad \text{if } P_i \in \partial\Omega;$$

if P_i is a midpoint of an edge, then

$$q = 4\mathsf{NV}_{\text{loc}} + 2, \quad \text{if } P_i \in \mathring{\Omega},$$
$$q = 4\mathsf{NT}_{\text{loc}} + 5, \quad \text{if } P_i \in \partial\Omega.$$

Here NV_{loc} denotes the number of neighbouring vertices in the pressure space.

RIGHT_HAND_SIDE
Assembles the right-hand side of the system (4.1).

DE_REF
Eliminates those refined tetrahedra which were marked in **REF_FLAG** and transforms the right-hand side **F** according to (3.3).

To coarsen a tetrahedron T we have to check whether T belongs to a resolvable patch. One of the problems that occur is to identify the coarsening node of the "father" after having melted its "children".

COMPR

After eliminating degrees of freedom there are "holes" left in arrays. **COMPR** compresses these arrays.

THETA_SCHEME

This routine represents the solver of NVS as described in Section 4.1. As mentioned above, the inner loop of the algorithm consists of several solvers for scalar equations of the type

$$u - \tau\,\Delta u = f.$$

These problems can efficiently be solved by a preconditioned conjugate gradient method.

ERROR_ESTIMATOR

Computes the estimation η_T, $T \in \mathcal{T}_k$, and fills the array **REF_FLAG** according to (1.9).

5. Numerical results

As a test example consider a local perturbation moving with approximately constant speed in a uniform flow. For that define

$$\phi(x) := Ux_1\left(1 + 0.5\frac{a^3}{r^3}\right), \qquad \psi(x) := -A\left(x_2^2 + x_3^2\right)\left(a^2 - r^2\right),$$

with $r := \sqrt{x_1^2 + x_2^2 + x_3^2}$ and a, U, $A > 0$. ϕ describes the potential of a flow around a ball with radius a and velocity Ue_1 at infinity. ψ is the so-called *Hill's spherical vortex* with strength A. Now define

$$v_0(x) := \begin{cases} \operatorname{rot} \psi(x), & |x| \leqslant a, \\ \nabla\phi(x), & |x| > a. \end{cases}$$

In the sequel we assume $U = \frac{1}{15} \cdot 0.2\ a^2 A$. This implies that v_0 is globally continuous and that it is a stationary solution of the Euler equations.

We solved NVS numerically with the following data:

$$\begin{aligned} &\Omega =]-1, 2[\times]0.5, 0.5[^2, \\ &u_0(x) = 0.5(v_0(x) + e_1), \quad \text{for } x \in \Omega, \\ &g(x) = e_1, \quad \text{for } x \in \partial\Omega, \\ &\mathrm{Re} = 1000, \qquad \Delta t = 0.04, \qquad a = 0.27, \qquad U = 1. \end{aligned}$$

Note that the initial and boundary data are almost compatible for $a \ll 1$. In Fig. 5.1 v_0 is sketched.

The following figures show the numerical results computed on a CONVEX C-210 obtained by using a uniform grid respectively the adaptive strategy.

As expected the vortex moves to the right with a velocity of about 0.5 and is damped by the influence of the viscosity (see Figs. 5.4, 5.6–5.11). The adaptive strategy is able to track the vortex through the computational domain. The error estimator recognizes the region occupied by the vortex as an area where a locally finer mesh is needed (see Figs. 5.5–5.8). Especially the

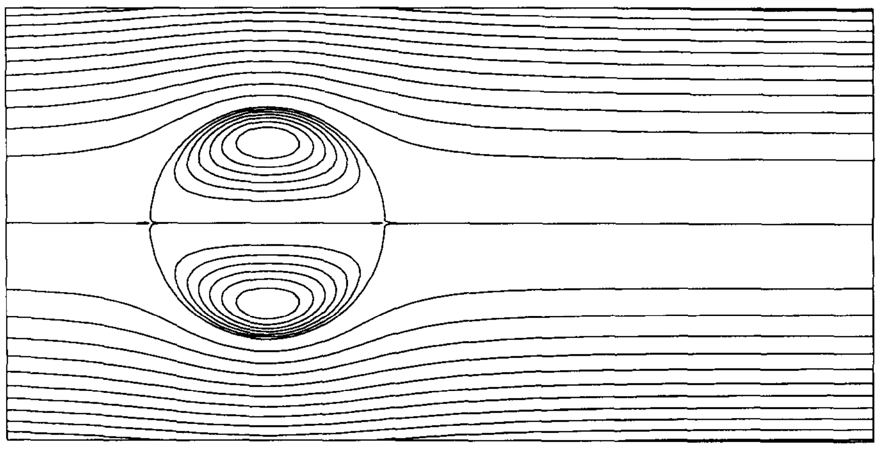

Fig. 5.1. Streamlines of v_0 on the plane $x_2 = 0$.

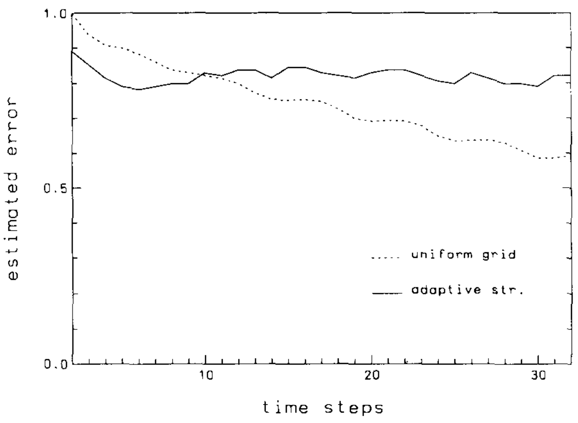

Fig. 5.2. History of the error.

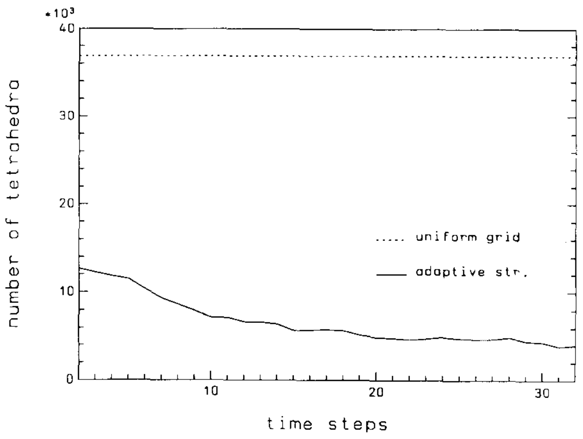

Fig. 5.3. History of the number of unknowns.

Table 5.1

Strategy	Total CPU-time	Overhead [a]
Uniform	81,780 sec	—
Adaptive	20,920 sec	1,820 sec

[a] "Overhead" includes all additionally computational work for the adaptive strategy such as calls of REFINE, MATRIX, CONNEC etc.

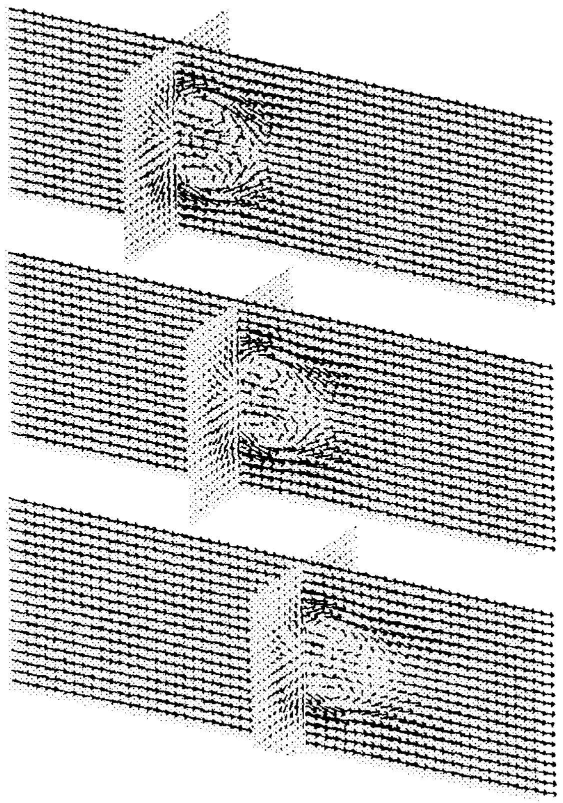
Fig. 5.4. Solution at time $t_1 = 0.0$, $t_2 = 0.64$, $t_3 = 1.28$; intersections with planes $x_2 = 0$ and $x_1 = -0.04$, $x_1 = 0.32$, $x_1 = 0.68$.

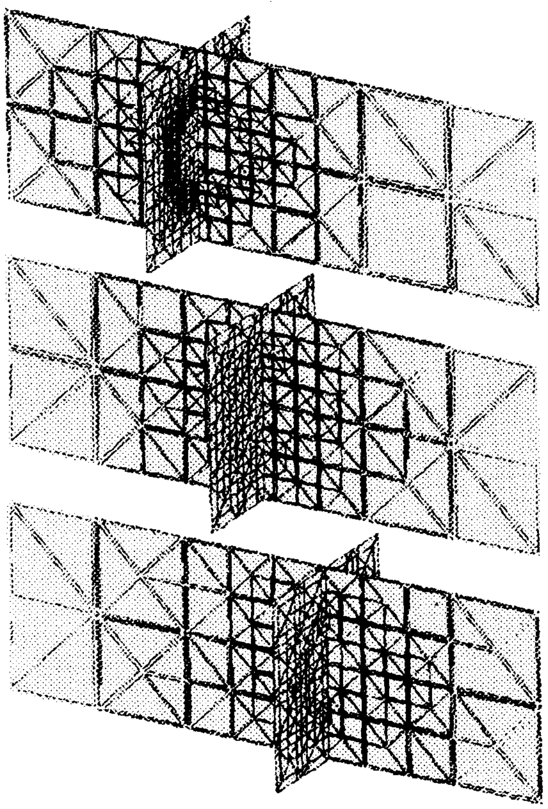
Fig. 5.5. Grids at time $t_1 = 0.0$, $t_2 = 0.64$, $t_3 = 1.28$; intersections with planes $x_2 = 0$ and $x_1 = 0.08$, $x_1 = 0.44$, $x_1 = 0.8$.

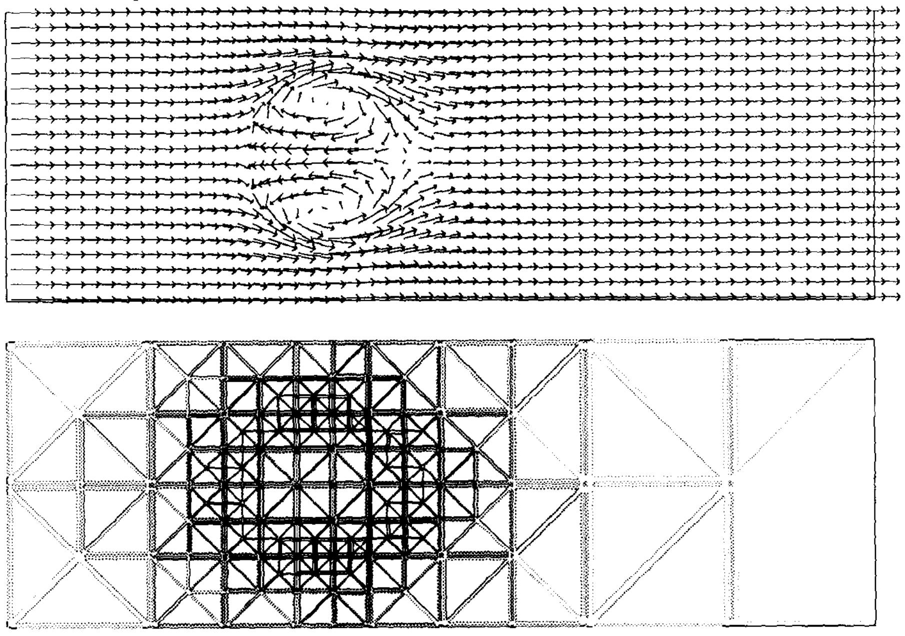
Fig. 5.6. Solution and grid at time $t_1 = 0$; intersection with plane $x_2 = 0$.

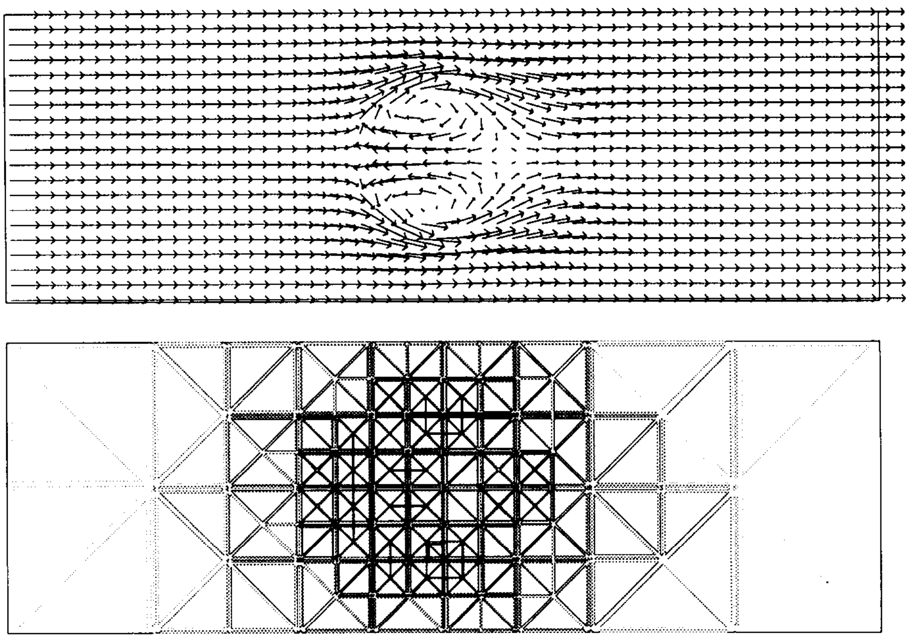

Fig. 5.7. Solution and grid at time $t_2 = 0.64$; intersection with plane $x_2 = 0$.

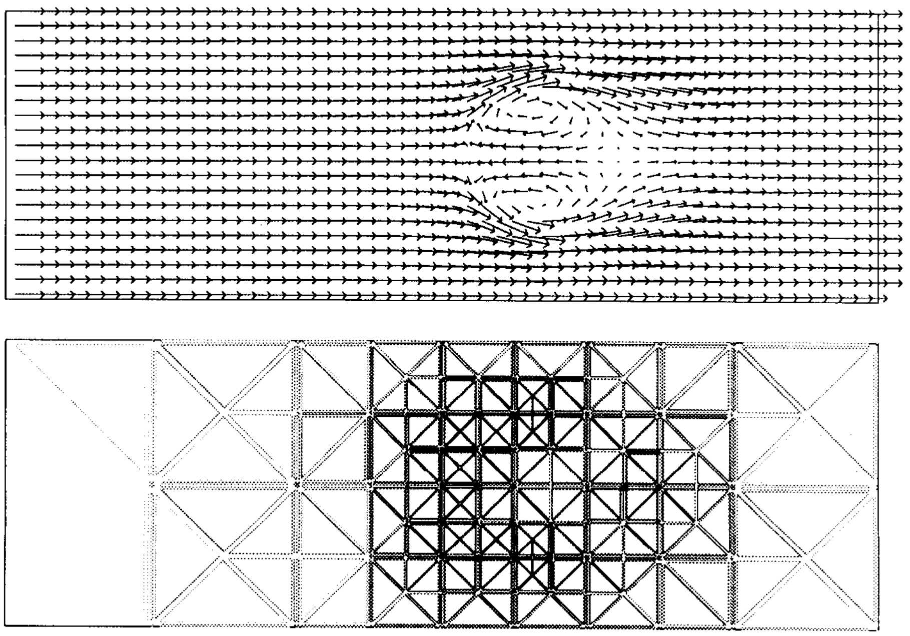

Fig. 5.8. Solution and grid at time $t_3 = 1.28$; intersection with plane $x_2 = 0$.

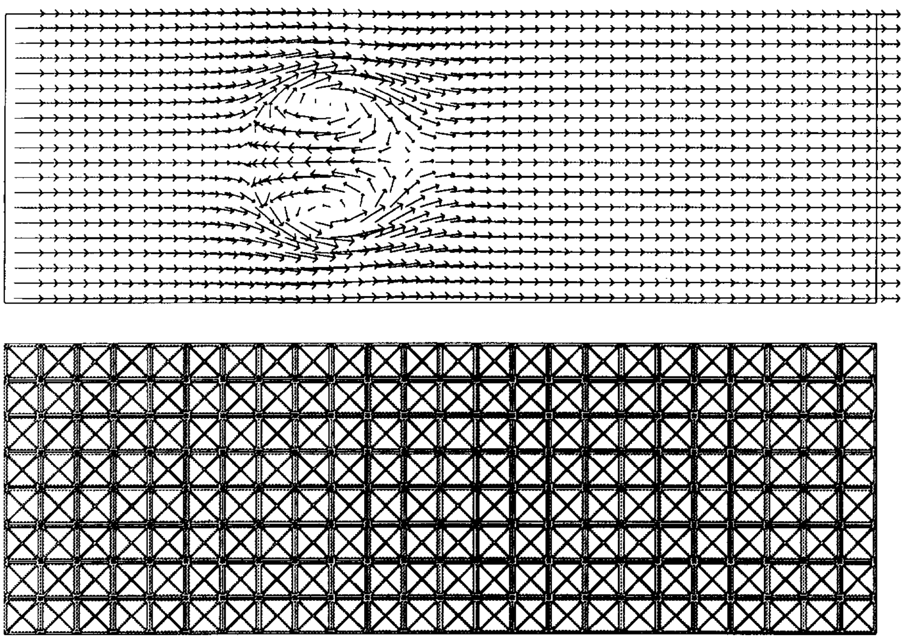
Fig. 5.9. Solution and uniform grid at time $t_1 = 0$; intersection with plane $x_2 = 0$.

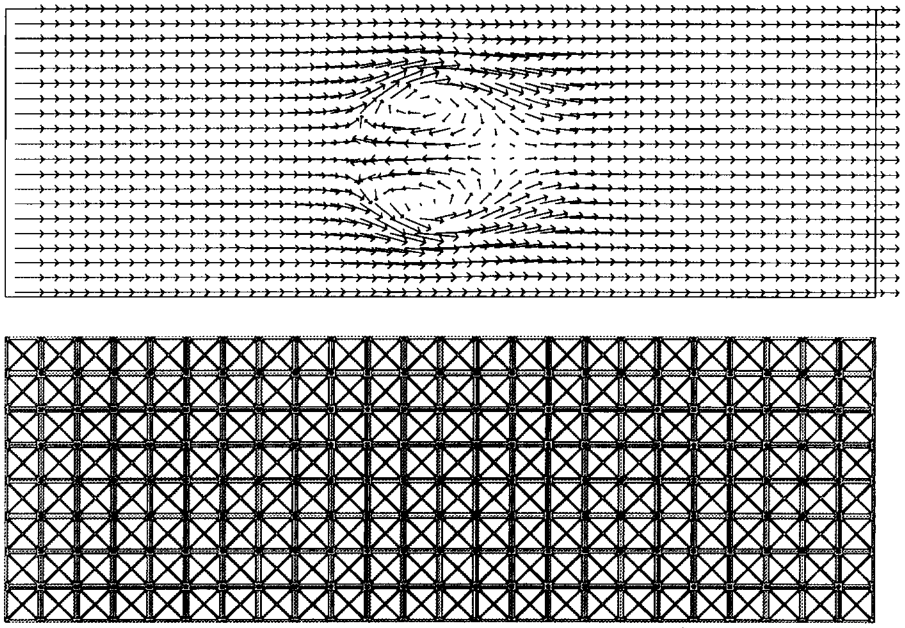
Fig. 5.10. Solution and uniform grid at time $t_2 = 0.64$; intersection with plane $x_2 = 0$.

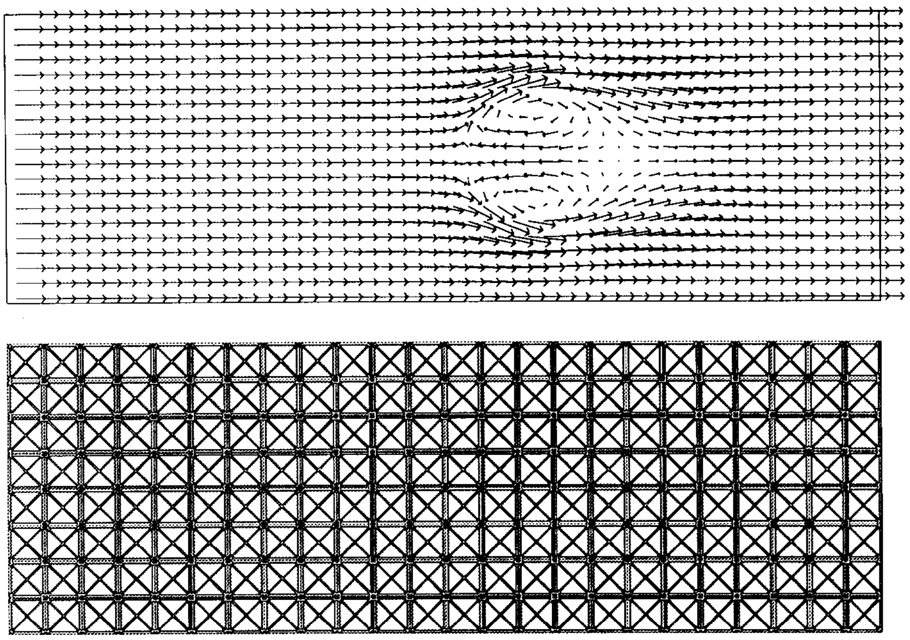

Fig. 5.11. Solution and uniform grid at time $t_3 = 1.28$; intersection with plane $x_2 = 0$.

boundary of the vortex — where there are jumps in derivatives of the solutions at time $t = 0$ — has to be refined.

When using a uniform grid, the error decreases due to the damping of the solution as time t increases. The adaptive strategy is able to reduce the amount of unknowns to keep a given tolerance in the error. Actually the error is nearly constant for each time step thus indicating the (quasi-)optimality of the grids (see Figs. 5.2 and 5.3).

Table 5.1 shows the benefit in CPU-time when using the adaptive strategy.

Figures 5.4–5.8 show the solutions and grids obtained by the adaptive strategy. The solution resulting from the computation with a uniform grid is presented in Figs. 5.9–5.11 for comparison. The flow is visualized by projecting arrows representing the velocity onto a clipping-plane. Different grey-values are used to indicate the mesh width. Dark means fine grid size whereas light colour indicates a locally coarse mesh.

Acknowledgement

I would like to express my thanks to the staff of the Graphics Laboratory of the SFB 256 at the University of Bonn, in particular M. Rumpf. His support in the application of the interactive graphics package GRAPE was essential for the evaluation and representation of the numerical results.

References

[1] I. Babuška and W.C. Rheinboldt, Error estimates for adaptive finite element computations, *SIAM J. Numer. Anal.* **15** (1978) 736–754.

[2] I. Babuška and W.C. Rheinboldt, A posteriori error estimates for the finite element method, *Internat. J. Numer. Methods Engrg.* **12** (1978) 1597–1615.

[3] I. Babuška, O.C. Zienkiewicz, J. Gago and E.R. de Oliveira, Eds., *Accuracy Estimates and Adaptive Refinements in Finite Element Computations* (Wiley, New York, 1986).

[4] R.E. Bank and A. Weiser, Some a posteriori error estimators for elliptic partial differential equations, *Math. Comp.* **44** (1985) 283–301.

[5] E. Bänsch, Local mesh refinement in 2 and 3 dimensions, Report 6, SFB 256, Univ. Bonn, 1989; *IMPACT Comput. Sci. Engrg.*, submitted.

[6] M.O. Bristeau, R. Glowinski and J. Periaux, Numerical methods for the Navier–Stokes equations. Application to the simulation of compressible and incompressible flows, *Comput. Phys. Rep.* **6** (1987) 73–188.

[7] K. Eriksson and C. Johnson, An adaptive finite element method for linear elliptic problems, *Math. Comp.* **50** (1988) 361–383.

[8] E. Fernandez-Cara and M.M. Beltran, The convergence of two numerical schemes for the Navier–Stokes equations, *Numer. Math.* **55** (1989) 33–60.

[9] C. Johnson, Adaptive finite element methods for diffusion and convection problems, *Comput. Methods Appl. Mech. Engrg.* **82** (1–3) (1990) 301–322.

[10] W.F. Mitchell, A comparison of adaptive refinement techniques for elliptic problems, *ACM Trans. Math. Software* **15** (4) (1989) 326–347.

[11] F. Otto, Die Babuška–Brezzi-Bedingung für das Taylor–Hood-element, Diplomarbeit, Inst. Angew. Math., Univ. Bonn, 1990.

[12] W.C. Rheinboldt, On a theory of mesh-refinement processes, *SIAM J. Numer. Anal.* **17** (1980) 766–778.

[13] W.C. Rheinboldt, Adaptive mesh refinement processes for finite element solutions, *Internat. J. Numer. Methods Engrg.* **17** (1981) 649–662.

[14] M.-C. Rivara, Algorithms for refining triangular grids suitable for adaptive and multigrid techniques, *Internat. J. Numer. Methods Engrg.* **20** (1984) 745–756.

[15] M.-C. Rivara, Mesh refinement processes based on the generalized bisection of simplices, *SIAM J. Numer. Anal.* **21** (1984) 604–613.

[16] M.-C. Rivara, A grid generator based on 4-triangles conforming mesh-refinement algorithms, *Internat. J. Numer. Methods Engrg.* **24** (1987) 1343–1354.

[17] R. Verfürth, A posteriori error estimators for the Stokes equations, *Numer. Math.* **55** (1989) 309–325.

[18] H. Yserentant, On the multi-level-splitting of finite element spaces, *Numer. Math.* **49** (1986) 379–412.