INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 22, 1041–1059 (1996)

A NON-LINEAR ADAPTIVE TRI-TREE MULTIGRID SOLVER FOR FINITE ELEMENT FORMULATIONS OF THE NAVIER–STOKES EQUATIONS

S. Ø. WILLE

Faculty of Engineering, Oslo College, Norway, Cort Adelersgate 30, N-0254 Oslo, Norway

SUMMARY

An iterative adaptive equation multigrid solver for solving the implicit Navier–Stokes equations simultaneously with tri-tree grid generation is developed. The tri-tree grid generator builds a hierarchical grid structure which is mapped to a finite element grid at each hierarchical level. For each hierarchical finite element multigrid the Navier–Stokes equations are solved approximately. The solution at each level is projected onto the next finer grid and used as a start vector for the iterative equation solver at the finer level. When the finest grid is reached, the equation solver is iterated until a tolerated solution is reached. The iterative multigrid equation solver is preconditioned by incomplete LU factorization with coupled node fill-in.

The non-linear Navier–Stokes equations are linearized by both the Newton method and grid adaption. The efficiency and behaviour of the present adaptive method are compared with those of the previously developed iterative equation solver which is preconditioned by incomplete LU factorization with coupled node fill-in.

KEY WORDS: grid generation; tri-tree; unstructured grid; multigrid; finite element; mixed formulation; analytic integration; adaptive solver; Navier–Stokes equations

INTRODUCTION

Intensive research on developing efficient algorithms for solving the Navier–Stokes equations for arbitrary geometries has taken place in several physical disciplines such as aerodynamics,[1] hydrodynamics[2] and haemodynamics.[3,4] For implicit solution algorithms,[4–7] direct equation solvers have shown limitations due to rather large computer storage and computer time requirements.[8] In view of this, iterative equation solvers have been paid extensive attention, with the ultimate goal to be able to solve the Navier–Stokes equations for large, time-dependent, three-dimensional problems with complex geometry. Although there have been substantial developments towards efficient solvers, there are still needs and possibilities for further improvements.

Recently, several iterative equation solvers for non-symmetric equation systems have been developed and tested.[7,9–12] These iterative equation solvers have gained quite a lot in both efficiency and robustness by the use of different preconditioning algorithms of the equation system.[13–15] In previous papers[5,7] a new incomplete LU factorization preconditioner with a coupled node fill-in algorithm was presented. The philosophy of this ILU preconditioner made it possible to obtain also a preconditioning matrix for the pressure coefficients in the equation matrix. Fill-ins with this algorithm were allowed where the nodes in the equation system were coupled and not only where the coefficients were initially different from zero. This ILU preconditioner revealed advantageous properties also when the equation system was reduced to form an inner–outer iterative algorithm.[7]

CCC 0271–2091/96/111041–19

Received February 1995
Revised August 1995

The most time-consuming operation in iterative equation solvers of the conjugate gradient type is matrix–vector multiplication. Since the finite element equations are solved for successively finer grids during the refinement procedure, the matrix generation of the equation system should be as fast as possible. Traditionally, numerical integration is applied to form the equation matrix. However, since simple elements such as triangles in two dimensions and tetrahedra in three dimensions are applied, the integration of the element matrix terms can be executed analytically. Analytical integration will then save a lot of computational work during the finite element calculations. The integration formulae consist of a constant part, independent of element size, multiplied by a term containing the relative location of the nodes within each element.

During the transition from coarse to finer grid the solution of the coarse grid is interpolated to the fine grid and used as a start vector at the fine grid. The refinement procedure on the grid consists of dividing each element into four new elements in two dimensions and eight new elements in three dimensions. Then some nodes will be common to both the coarse and the fine grid. For these nodes, solution values of the coarse grid are used directly. New nodes in the fine grid are generated at the midpoints between the nodes in the coarse grid. The start values for the iterations at these nodes in the fine grid are then found by linear interpolation. The main purpose of the present adaptive algorithm is to obtain better start vectors as the grids become more and more refined. When the finest grid is reached, the solution is iterated until the desired convergence criterion is satisfied.

In a previous paper a new tri-tree method[16] for generating unstructured grids,[17,18] the tri-tree algorithm for generating grids in two and three dimensions, was presented. The tri-tree algorithm method starts with a triangle or tetrahedron which is subdivided into four new triangles or eight new tetrahedra respectively. The tri-tree structure then has pointers like the quad-tree and oct-tree.[19,20] The main and essential difference is that the leaves in the tri-tree consist of triangles and tetrahedra. The triangulation procedure of the tri-tree element structure is then much simplified compared with that of the oct-tree structure and will only consist of connecting triangles or tetrahedra of different sizes. By introducing very mild restrictions on the tri-tree structure, which hardly affect the ability of local refinements, the triangulation procedure becomes very simple. The elements generated are optimal in the sense that they do not collapse during the refinements. The elements are equilateral triangles and tetrahedra, or at the interfaces of elements of different sizes the equilateral triangles will be divided into two and the equilateral tetrahedra will be divided into two or four.

During the triangulation procedure an efficient search algorithm is needed for finding co-ordinate points in space. In the present work a lexical tree search algorithm for the point co-ordinates has proved to be very efficient.

The initial triangle is successively subdivided into four new triangles and the tetrahedron into eight new tetrahedra. The successive subdivision is continued until the required level of refinement is reached. At each level of tri-tree refinement an associated finite element grid can be constructed and used for finite element calculations. The tri-tree data structure is therefore well suited for an adaptive algorithm.

EQUATIONS

The non-linear Navier-Stokes equations are given by

$$-\mu\nabla^2\mathbf{v} + \rho\mathbf{v}\cdot\nabla\mathbf{v} + \nabla p = 0 \quad \text{in} \quad \Omega, \tag{1}$$

$$-\nabla\cdot\mathbf{v} = 0 \quad \text{in} \quad \Omega, \tag{2}$$

where v is the velocity vector, p is the pressure and μ is the viscosity coefficient. The first equation is the equation of motion which contains a diffusion and a pressure gradient term. The second equation is the equation of continuity. A minus sign is introduced in the continuity equation in order to obtain the same sign for the pressure gradient as for the continuity equation in the finite element formulation. In the finite element formulation the velocities are approximated with quadratic basis functions and the pressure is approximated with linear basis functions on each element.[21] Denote the quadratic polynomials by N_i and the linear polynomials by L_i. Then by the Galerkin residual method and integration by parts the second-order finite element formulation of the Navier-Stokes equation system becomes

$$\begin{aligned} \mathbf{F}_\mathbf{v} &= \int_\Omega \mu \nabla N_i \cdot \nabla \mathbf{v}\, d\Omega + \int_\Omega \rho N_i \mathbf{v} \cdot \nabla \mathbf{v}\, d\Omega - \int_\Omega \nabla N_i p\, d\Omega - \int_{\delta\Omega} \mu N_i \frac{\partial \mathbf{v}}{\partial n}\, d\delta\Omega + \int_{\delta\Omega} N_i p\, d\delta\Omega = 0, \\ \mathbf{F}_\mathbf{p} &= -\int_\Omega L_i \nabla \cdot \mathbf{v}\, d\Omega = 0. \end{aligned} \tag{3}$$

The following equation system can then be solved by successive approximation for the second-order polynomial approximation:

$$\begin{bmatrix} \int_\Omega (\mu \nabla N_i \nabla N_j + \rho N_i \mathbf{v} \nabla N_j)\, d\Omega & -\int_\Omega \nabla N_i L_j\, d\Omega \\ -\int_\Omega L_i \nabla N_j\, d\Omega & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \int_{\delta\Omega} \left(-\mu N_i \frac{\partial \mathbf{v}}{\partial n} + N_i p \right) d\delta\Omega \\ 0 \end{bmatrix}. \tag{4}$$

There are several methods for linearizing this equation system. Usual linearization techniques involves the computation of gradients or approximate gradients, e.g. Newton or steepest descent methods. However, another simple way of linearization which has not previously been given attention is adaptive grid approximation. The adaptive grid linearization method will be discussed in a later section.

NEWTON LINEARIZATION

The Navier-Stokes equations contain one non-linear term, the convective acceleration, which requires a non-linear iterative solution procedure. The non-linear algorithm chosen is the Newton method, which is known to have a second-order convergence rate. The Navier–Stokes equations (3) then have to be differentiated with respect to the unknowns and the linear equation system which has to be solved at each Newton step is

$$\begin{bmatrix} \frac{\partial \mathbf{F}_\mathbf{v}^n}{\partial \mathbf{v}} & \frac{\partial \mathbf{F}_\mathbf{v}^n}{\partial p} \\ \frac{\partial \mathbf{F}_\mathbf{p}^n}{\partial \mathbf{v}} & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{v} \\ \Delta \mathbf{p} \end{bmatrix} = -\begin{bmatrix} \mathbf{F}_\mathbf{v}^n \\ \mathbf{F}_\mathbf{p}^n \end{bmatrix}, \tag{5}$$

where the matrix and the right-hand side are given by

$$\begin{bmatrix} \int_\Omega [\mu \nabla N_i \nabla N_j + \rho N_i(\nabla \mathbf{v} N_j + \mathbf{v} \nabla N_j)]\, d\Omega & -\int_\Omega \nabla N_i L_j\, d\Omega \\ -\int_\Omega L_i \nabla N_j\, d\Omega & 0 \end{bmatrix} \begin{bmatrix} \Delta \mathbf{v} \\ \Delta \mathbf{p} \end{bmatrix}$$

$$= -\begin{bmatrix} \int_\Omega (\mu \nabla N_i \cdot \nabla \mathbf{v} + \rho N_i \mathbf{v} \cdot \nabla \mathbf{v} - \nabla N_i p)\, d\Omega + \int_{\delta\Omega} \left(-\mu N_i \frac{\partial \mathbf{v}}{\partial n} + N_i p\right) d\delta\Omega \\ -\int_\Omega L_i \nabla \cdot \mathbf{v}\, d\Omega \end{bmatrix}. \tag{6}$$

If the initial solution $\mathbf{v}^0, p^0$ is chosen close enough to the final solution, convergence of the non-linear equation system is guaranteed. The solution is then updated at each Newton step with the correction found by solving (6):

$$\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta \mathbf{v}, \tag{7}$$

$$\mathbf{p}^{n+1} = \mathbf{p}^n + \Delta \mathbf{p}. \tag{8}$$

The number of Newton steps to obtain a converged solution is usually of the order of 5–10, depending on the strength of the non-linearities and the magnitude of the convergence criterion.

The Newton method for non-linear equation systems can also be applied favourably to the linear Stokes equations. For linear equation systems, only one Newton iteration is required. The advantage of a Newton formulation for linear equation systems appears when introducing the Dirichlet boundary conditions. The columns and rows in the equation matrix can then be zeroed with a one on the corresponding diagonal and a zero on the right-hand side. The Dirichlet value is included in the initial guess of the solution vector. The application of Dirichlet boundary conditions to the original equation matrix would be more complicated to maintain the advantageous symmetric property, as the Dirichlet condition multiplied by the corresponding column vector has to be subtracted from the right-hand side.

ADAPTIVE LINEARIZATION

The non-linear set of Navier–Stokes equations can be linearized in several ways. The Newton linearization can take place globally and the multigrid algorithm can be applied to solve the linearized Navier–Stokes equations for each Newton step. Another way of linearization is to linearize locally at each grid level and solve the non-linear Navier–Stokes equations at each grid level. However, an alternative or supplement to Newton linearization of the equation system is local grid adaptation to the solution, which will also contribute to the linearization of the equation system. From the analytic integration the following formula is obtained:

$$\frac{\int_\Omega \rho N_i \mathbf{v}_c \cdot \nabla N_{jc}\, d\Omega}{\int_\Omega \mu \nabla N_i \cdot \nabla N_j\, d\Omega} = \frac{al}{b}. \tag{9}$$

In this formula, a and b are constants independent of element size, while l is some characteristic length of the element. The formula shows that the magnitude of the matrix coefficient of the convection can be reduced arbitrarily compared with the diffusion coefficient in the implicit equation system by local

refinements. A natural criterion for adapting the grid to the solution of the Navier–Stokes equations would then be to fulfil the criterion

$$\int_\Omega \rho N_i \mathbf{v}_c \cdot \nabla \mathbf{v}_c \, d\Omega \ll \int_\Omega \mu \nabla N_i \cdot \nabla \mathbf{v} \, d\Omega. \tag{10}$$

The above relations are valid in both two and three dimensions and for first- and second-order polynomial approximations of the Navier–Stokes equations. By reducing the element size where the convection is large, the equation system becomes more and more linear and symmetric. If the local element size is then reduced sufficiently, this implicit adaptive linearization will for many Navier–Stokes applications appear to be sufficient and satisfactory.

ANALYTIC INTEGRATION

Let the linear basis functions be denoted by L_i and the quadratic basis functions by N_i. Then in three dimensions

$$L_i = a_i + b_i x + c_i y + d_i z.$$

The quadratic basis function can then be given as a function of the linear basis function. For the corner nodes i and midside nodes n respectively

$$N_i = L_i(2L_i - 1), \qquad N_n = 4L_jL_k,$$

where the nodes j and k are the corner nodes on each side of the midside node n. The corner nodes are numbered first, then the midside nodes.

Let n_d be the spatial dimension. The exact integrals can be computed by the formula

$$\int_\Omega L_i^\alpha L_j^\beta L_k^\gamma L_n^\delta L_m^\omega = \frac{\alpha!\beta!\gamma!\delta!\omega!}{(\alpha+\beta+\gamma+\delta+\omega+n_d)!} n_d!\Omega.$$

Let

$$\delta_{i,j,k,n,m} = \alpha!\beta!\gamma!\delta!\omega!. \tag{11}$$

The function $\delta_{i,j,k,n,m}$ is simple to implement: just count the number of equal indices and compute the corresponding faculties.

In the formulae below the δ-function is defined by

$$\delta_{ij} = \begin{cases} 2, & i = j, \\ 1, & i \neq j. \end{cases}$$

Linear matrix coefficients

Let the corner nodes have the local node numbers $1, \ldots, n_c$ and let the midside nodes be locally numbered as $n_c + 1, \ldots, n_e$. In the second-order basis function formulation the integrals of the derivatives in the equation matrix are given by

$$D = (n_d + 1)(n_d + 2),$$

$$iii = 1, \qquad jj = j,$$

$$\int_\Omega \frac{\partial N_{ii}}{\partial x}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega (4L_i - 1)(4L_j - 1)\frac{\partial L_i}{\partial x}\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega$$
$$= \{16\delta_{ij} - [8 - (n_\mathrm{d} + 1)](n_\mathrm{d} + 2)\}\frac{\partial L_i}{\partial x}\frac{\partial L_j}{\partial x}\frac{\Omega}{D}, \tag{12}$$
$$\int_\Omega L_{ii}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega L_i(4L_j - 1)\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega = [4\delta_{ij} - (n_\mathrm{d} + 2)]\frac{\partial L_j}{\partial x}\frac{\Omega}{D},$$

$$ii = i, \qquad jj = midpoint(n, m),$$

$$\int_\Omega \frac{\partial N_{ii}}{\partial x}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega (4L_i - 1)\frac{\partial L_i}{\partial x}\left(4L_m\frac{\partial L_n}{\partial x} + 4L_n\frac{\partial L_m}{\partial x}\right)\mathrm{d}\Omega$$
$$= 4\frac{\partial L_i}{\partial x}\left[4\delta_{im}\frac{\partial L_n}{\partial x} + 4\delta_{in}\frac{\partial L_m}{\partial x} - \left(\frac{\partial L_n}{\partial x} + \frac{\partial L_m}{\partial x}\right)(n_\mathrm{d} + 2)\right]\frac{\Omega}{D}, \tag{13}$$
$$\int_\Omega L_{ii}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega L_i\left(4L_m\frac{\partial L_n}{\partial x} + 4L_n\frac{\partial L_m}{\partial x}\right)\mathrm{d}\Omega = 4\left(\delta_{in}\frac{\partial L_m}{\partial x} + \delta_{im}\frac{\partial L_n}{\partial x}\right)\frac{\Omega}{D},$$

$$ii = midpoint(n, m), \qquad jj = midpoint(p, q),$$

$$\int_\Omega \frac{\partial N_{ii}}{\partial x}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega \left(4L_p\frac{\partial L_q}{\partial x} + 4L_q\frac{\partial L_p}{\partial x}\right)\left(4L_n\frac{\partial L_m}{\partial x} + 4L_m\frac{\partial L_n}{\partial x}\right)\mathrm{d}\Omega$$
$$= 16\left(\delta_{pn}\frac{\partial L_q}{\partial x}\frac{\partial L_m}{\partial x} + \delta_{qn}\frac{\partial L_p}{\partial x}\frac{\partial L_m}{\partial x} + \delta_{pm}\frac{\partial L_q}{\partial x}\frac{\partial L_n}{\partial x} + \delta_{qm}\frac{\partial L_p}{\partial x}\frac{\partial L_n}{\partial x}\right)\frac{\Omega}{D}. \tag{14}$$

Non-linear matrix coefficients

$$ii = i, \qquad kk = k, \qquad jj = j,$$

$$\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega (2L_i^2 - L_i)(2L_k^2 - L_k)(4L_j - 1)\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega$$
$$= [[16\delta_{iikkj} - \{8\delta_{iikj} + 8\delta_{ikkj} + 4\delta_{iikk} - [4\delta_{ikj} + 2\delta_{iik} + 2\delta_{ikk}$$
$$- \delta_{ik}(n_\mathrm{d} + 3)](n_\mathrm{d} + 4)\}(n_\mathrm{d} + 5)]]\frac{\partial L_j}{\partial x}\frac{\Omega}{D}, \tag{15}$$

$$ii = i, \qquad kk = k, \qquad jj = midpoint(n, m),$$

$$\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega = \int_\Omega (2L_i^2 - L_i)(2L_k^2 - L_k)4\left(L_m\frac{\partial L_n}{\partial x} + L_n\frac{\partial L_m}{\partial x}\right)\mathrm{d}\Omega$$
$$= \Bigg(\{16\delta_{iikkm} - [8\delta_{iikm} + 8\delta_{ikkm} - 4\delta_{ikm}(n_\mathrm{d} + 4)](n_\mathrm{d} + 5)\}\frac{\partial L_n}{\partial x}$$
$$\times\{16\delta_{iikkm} - [8\delta_{iikn} + 8\delta_{ikkn} - 4\delta_{ikn}(n_\mathrm{d} + 4)](n_\mathrm{d} + 5)\}\frac{\partial L_m}{\partial x}\Bigg)\frac{\Omega}{D}, \tag{16}$$

$$ii = i, \qquad kk = midpoint(n, m), \qquad jj = j,$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega (2L_i^2 - L_i)4L_nL_m(4L_j - 1)\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega \\ &= \{32\delta_{iinmj} - [16\delta_{inmj} + 8\delta_{iinm} - 4\delta_{inm}(n_\mathrm{d} + 4)](n_\mathrm{d} + 5)\}\frac{\partial L_j}{\partial x}\frac{\Omega}{D},\end{aligned} \tag{17}$$

$$ii = i, \qquad kk = midpoint(n, m), \qquad jj = midpoint(p, q),$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega (2L_i^2 - L_i)4L_nL_m4\left(L_p\frac{\partial L_q}{\partial x} + L_q\frac{\partial L_p}{\partial x}\right)\mathrm{d}\Omega \\ &= \left([32\delta_{iinmp} - 16\delta_{inmp}(n_\mathrm{d} + 5)]\frac{\partial L_q}{\partial x} + [32\delta_{iinmq} - 16\delta_{inmq}(n_\mathrm{d} + 5)]\frac{\partial L_p}{\partial x}\right)\frac{\Omega}{D},\end{aligned} \tag{18}$$

$$ii = midpoint(n, m), \qquad kk = k, \qquad jj = j,$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega 4L_nL_m(2L_k^2 - L_k)(4L_j - 1)\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega \\ &= \{32\delta_{kknmj} - [16\delta_{knmj} + 8\delta_{kknm} - 4\delta_{knm}(n_\mathrm{d} + 4)](n_\mathrm{d} + 5)\}\frac{\partial L_j}{\partial x}\frac{\Omega}{D},\end{aligned} \tag{19}$$

$$ii = midpoint(n, m), \qquad kk = k, \qquad jj = midpoint(p, q),$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega 4L_nL_m(2L_k^2 - L_k)4\left(L_p\frac{\partial L_q}{\partial x} + L_q\frac{\partial L_p}{\partial x}\right)\mathrm{d}\Omega \\ &= \left([32\delta_{kknmp} - 16\delta_{knmp}(n_\mathrm{d} + 5)]\frac{\partial L_q}{\partial x} + [32\delta_{kknmq} - 16\delta_{knmq}(n_\mathrm{d} + 5)]\frac{\partial L_p}{\partial x}\right)\frac{\Omega}{D},\end{aligned} \tag{20}$$

$$ii = midpoint(n, m), \qquad kk = midpoint(p, q), \qquad jj = j,$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega 4L_nL_m4L_pL_q(4L_j - 1)\frac{\partial L_j}{\partial x}\,\mathrm{d}\Omega \\ &= [64\delta_{nmpqj} - 16\delta_{nmpq}(n_\mathrm{d} + 5)]\frac{\partial L_j}{\partial x}\frac{\Omega}{D},\end{aligned} \tag{21}$$

$$ii = midpoint(n, m), \qquad kk = midpoint(p, q), \qquad jj = midpoint(r, s),$$

$$\begin{aligned}\int_\Omega N_{ii}N_{kk}\frac{\partial N_{jj}}{\partial x}\,\mathrm{d}\Omega &= \int_\Omega 4L_nL_m4L_pL_q4\left(L_r\frac{\partial L_s}{\partial x} + L_s\frac{\partial L_r}{\partial x}\right)\mathrm{d}\Omega \\ &= \left(64\delta_{nmpqr}\frac{\partial L_s}{\partial x} + 64\delta_{nmpqs}\right)\frac{\partial L_r}{\partial x}\frac{\Omega}{D}.\end{aligned} \tag{22}$$

Usually, numerical integration, e.g. Gauss integration, is applied to compute the coefficients in the finite element matrices. When simple elements such as triangles and tetrahedra are used, it is possible to

perform analytical integration. The coefficients of diffusion, convection, continuity and pressure gradient can be computed exactly.

TRI-TREE STRUCTURE

In the tri-tree search algorithm,[16] equilateral triangles and tetrahedra are used as basic domains. The equilateral triangles and tetrahedra are then subdivided into new equilateral triangles and tetrahedra. In two dimensions an equilateral triangle is divided into four triangles. A diagram of the two-dimensional tri-tree structure is shown in Figure 1. An initial equilateral triangle is divided into four new equilateral triangles. Each of these triangles can then be divided into another four equilateral triangles, and so on. The tree structure of these divisions is shown in the lower part of Figure 1. The record belonging to each triangle contains pointers to the triangles into which it is subdivided. This triangulation procedure therefore permits local refinements required by the geometric shape of the boundary as well as the properties of the solution.

In three dimensions an equilateral tetrahedron is divided into eight tetrahedra. The ordering of successive divisions is organized as a tree structure. The tree record structure needs nine integers in two dimensions and 14 integers in three dimensions in order to keep the necessary information at each level of subdivision.

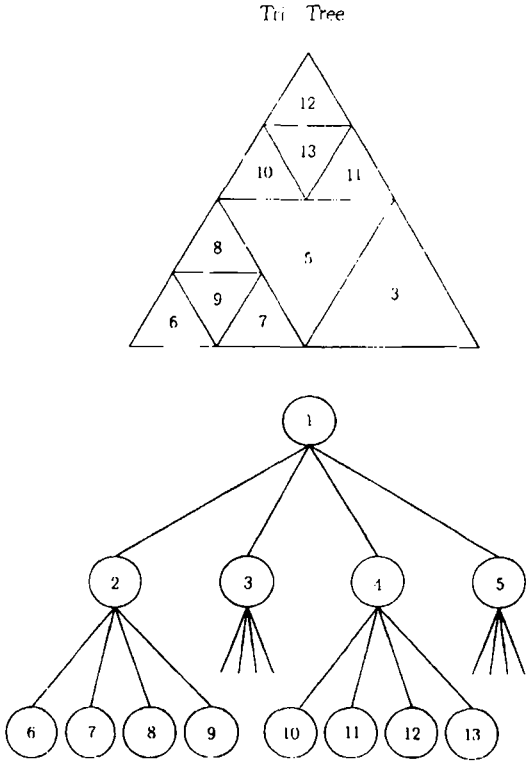
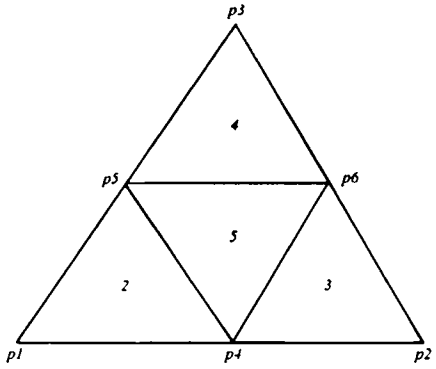


Figure 1. Hierarchical structure of the tri-tree. An initial equilateral triangle is divided into four new equilateral triangles. Each of these triangles can then be divided into another four equilateral triangles, and so on. The tree structure of these divisions is shown in the lower part of the figure. The record belonging to each triangle contains pointers to the triangles into which it is subdivided. This triangulation procedure therefore permits local refinements required by the geometric shape of the boundary as well as the properties of the solution

The records describing each two-dimensional triangular leaf are shown in Figure 2. A level number indicates the size of division and all triangles or tetrahedra of equal size will have the same level number. When a division is terminal, the level number is given a negative sign. In addition to the level number, a point index to each of the corners of the structure is stored. This is not strictly necessary, because the co-ordinates of each point can be calculated when they are needed. However, if the corner points are stored, the computing time is considerably reduced. The next positions in the structured record are pointers to the records of the divisions. When a triangle or tetrahedron is terminal, some of these pointers are used as pointers to the neighbour using triangles and tetrahedra instead. The last integer in the record points to the record of the parent triangle or tetrahedron. It is therefore possible to perform both up and down searches in the tri- tree.

When a triangle or tetrahedron is divided, the midpoint on each line between the corners is calculated. This point may already exist if the neighbour has a larger level number. If a point does not exist, it is added to the list of points. In order to be able to search for and add points fast, the list is organized as a binary tree. The binary tree, Figure 3, is sorted lexically on the point co-ordinates.



	Level	Corners			Pointers to divisions				Parent
1	1	p1	p2	p3	2	3	4	5	0
2	-2	p1	p4	p5	5	0	0		1
3	-2	p4	p2	p6	0	5	0		1
4	-2	p5	p6	p3	0	0	5		1
5	-2	p6	p4	p5	2	4	3		1

Figure 2. Numbering of triangular leaves in the tree structure together with global numbering of nodes. The record of each triangular structure contains information on the level of refinement at which the triangle is located. If the refinement level number is negative, the triangle is terminal in the tree structure. The following three numbers in the record point to the co-ordinates of the corners of the triangle. For a non-terminal triangular leaf the next four numbers point to the record of the four triangles into which it is divided. If the triangular leaf is terminal, three of these numbers are used as pointers to the records of neighbouring triangles. The last number in the triangle record points to the record of its parent

In order to find the neighbours of a tri-tree element, a search in the tri- tree is performed to find which tri-tree element contains a point slightly outside the edge or side of the present triangle or tetrahedron. The point to use in the tri-tree search is given by

$$P = P_g + (P_g - P_c)/d + \varepsilon(P_g - P_c). \tag{23}$$

In this expression, P_g is the centre of gravity and P^c is a corner in the tri-tree element. The spatial dimension is d ($d=2$ or 3) and ε is a small constant which depends on the accuracy of the actual computer. If ε is zero, P is the point where the line from the corner P_c through the point of gravity hits the opposite edge or side. For small ε the point P will be on the line from corner through the point of gravity slightly outside the tri-tree element. The constant ε should be chosen so that the computer representation of

$$P_g + (P_g - P_c)/d \neq P_g + (P_g - P_c)/d + \varepsilon(P_g - P_c) \tag{24}$$

in only two or three of the least significant digits. The point P defined in this way is a point slightly outside the element edge or side opposite to the corner P_c. A search in the tri-tree for a tri-tree element which encloses a point can either start at the root of the tree or at the location of the last search. If the points which are searched for are introduced in a random fashion, it will be most efficient to start at the root of the tree. When a search for the point P defined above is performed, the *a priori* knowledge is that the point is enclosed in an adjacent tri-tree element. The probability is therefore high that the adjacent tri-tree element belongs to the same subtree. If the tri-tree element belongs to the same subtree, it is faster to start the search at the present location, or even better at one level above the present location, than from the root of the tree. On the average, experiments indicate that it is most efficient to start the search at one level above the present. At each level the four triangles in two dimensions and the eight tetrahedra in three dimensions are explored to find which one contains the point.

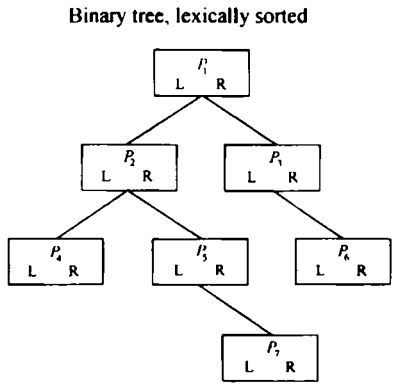
In the balancing procedure a tree element is refined if more than one neighbour is at a smaller level. The balancing procedure is an interative procedure. After the balancing procedure the tri-tree is valid for triangulation. In two dimensions there is at most one node at the midpoint of one of the edges of the triangles. This tri-tree triangle is divided into two finite element triangles. In three dimensions the situation is more complex. Each equilateral tri-tree tetrahedron can either have one node on one of the edges or three nodes at the edges of one of the sides. If there is one node at one edge, the tetrahedron is divided into two. If there are three nodes at the edges of one side, the tetrahedron is divided into four finite element tetrahedra. The triangulation procedure is only applied to tri-tree elements which are inside the computational domain. When the tri-tree is triangulated, the finite elements are kept in a finite element structure and the tri-tree structure is stored to be used later when the multigrid is further adapted to the solution.

ADAPTIVE SOLVER

Let G^k denote the set of grids $\{G^k: k = 1, \ldots, N\}$, where the grids G^k are in increasingly finer order. Let $\mathbf{x}^k \in \mathbf{X}^k$ be the set of functions which we require to solve the set of differential equations on the grid G^k. Let the transfer operator from coarse to fine grid be $\mathbf{P}^k$: $\mathbf{x}^{k-1} \rightarrow \mathbf{x}^k$, where $\mathbf{P}^k$ is the prolongation from coarse to fine. Let the set of differential equations to be solved on G^k be given by

$$\mathbf{F}^k(\mathbf{x}^k) = \mathbf{b}^k. \tag{25}$$

Let *Smooth*($\mathbf{x}$, $\tilde{\mathbf{x}}$) be a smoothing or approximate solution algorithm defined on every grid G^k, $\tilde{\mathbf{x}}$ the start vector and $\mathbf{x}$ the smoothed vector. The adaptive multigrid algorithm is then defined by



Given two points, P and Q

where

$P = [x, y, z]$ and $Q = [u, v, w]$

then $P \leq Q$ if

if x $\leq$ u
if x = u y $\leq$ v
if x = u y = v z $\leq$ w

Figure 3. During the refinement process the nodes with co-ordinates are stored in a binary tree. The key to each node is the co-ordinates, which determine whether one node is smaller or larger than another. The nodes are then lexically sorted and a fast search algorithm will decide whether a point during the refinement procedure is already present in the tree structure

Choose $\tilde{\mathbf{x}}^k$
for $(k = 1; k (= N - 1; k + +)$
{
$\mathbf{x}^k = \mathbf{x}^k + \mathbf{P}^k(\mathbf{x}^{k-1} - \tilde{\mathbf{x}}^{k-1})$;
Smooth$(\mathbf{x}^k, \tilde{\mathbf{x}}^k)$;
}
Solve $\mathbf{F}^N(\mathbf{x}^N) = \mathbf{b}^N$ iteratively.

The initial triangle or tetrahedron is successively refined until the desired refinement level is reached. At each tri-tree level of refinement a finite element grid is constructed and the set of differential equations is solved approximately for this grid. The approximate solution on one finite element multigrid level is then interpolated and projected onto the finer grid and used as a start vector for this grid.

The prolongation $\mathbf{P}^k$ is the mapping from coarse to fine grid. The values of the common nodes are taken from the coarse grid and the values of the new nodes at the midpoints of each side are interpolated linearly. The linear interpolation procedure is simply to take the average between two corner nodes. The prolongation algorithm is applied in both two and three dimensions. There exist more complicated local smoothing algorithms which take into account several neighbouring nodes. However, as local smoothing is followed by global smoothing, a simple first-order local smoothing algorithm is sufficient.

The critical part of the adaptive multigrid algorithm is the global smoothing method. The special problem which arises with the Navier-Stokes equations is the zero diagonal block[5,7] associated with the continuity equation, which implies non-positive definiteness of the equation matrix. Thus smoothing algorithms such as Gauss–Seidel and traditional ILU factorization cannot be applied directly as

smoothing procedure. However, if some rather arbitrary postconditioning[22] matrix is used, this limitation can be overcome. The difficulty with non-positive definiteness can also be avoided with inner– outer iterations. As the equation matrix is non-symmetric, the usual conjugate gradient type of smoothing cannot be applied either. The introduction of inner–outer iterations and a postconditioning matrix certainly represents an increase in superfluous work. In the present work the CGSTAB conjugate gradient method[11,12] with coupled node fill-in, which is often considered as an iterative equation solver, is used as smoother.

The adaptive multigrid algorithm starts with the coarsest grid, computing a smoothed or exact solution for this grid. This solution and the corresponding residual are then prolonged to the finer grid. At the finest multigrid level the solution is determined fully converged. When the equation system is solved for the finest grid, the adaptive multigrid cycle is complete.

The smoothing algorithm within each adaptive multigrid iteration can be just a few iterations with the CGSTAB smoother or a fully converged solution found by the CGSTAB equation solver. For linear equations the equation solver can be stopped either after a fixed small number of iterations or by a convergence criterion defined by

$$\frac{\|\mathbf{r}^k\|}{\|\mathbf{b}^k\|} < \varepsilon_l, \tag{26}$$

where $\mathbf{r}^k$ is the residual and $\mathbf{b}^k$ is the right-hand side at multigrid level k. However, for non- linear equations this convergence criterion is not quite suitable, as the right-hand side approaches zero in the Newton iteration procedure. A better convergence criterion for the linear iterations for solving non-linear equations is

$$\frac{\|\delta\mathbf{x}^k\|}{\|\mathbf{x}^k\|} < \varepsilon_l, \tag{27}$$

where $\mathbf{x}^k$ is the solution of the non-linear equation system and $\delta\mathbf{x}^k$ is the update in the linear equation solver. For the linear Stokes equations the two convergence criteria are equivalent and of the same order of magnitude.

NUMERICAL EXPERIMENTS

The node numbering delivered by the tri-tree grid generator is shown in Figure 2. The nodes in the tri-tree generator are numbered as new nodes are introduced during refinement of the grid. The corners are numbered first, and when the final refinement level is reached, the midside nodes are introduced in element order. However, this way of ordering the nodes is not optimal when the equation system is preconditioned by incomplete factorization with coupled node fill-in. Four ways of numbering the nodes were tested in a previous paper.[23] In the sorting algorithm used in the present experiments, Figure 4, the nodes are sorted with respect to their distance from the centre of the grid. The node which is furthest away from the geometrical centre is given the smallest number. The node in the middle of the grid gets the highest number.

The test problem for the Navier–Stokes equations is the cavity problem and the boundary conditions are shown in Figure 5. In Figure 6 a hierarchy of the multigrid is shown. By the projection algorithm the equation system is first solved for the coarsest grid. The solution is projected onto the finer grid and used as start vector in the Newton iterations. The projection of the solution onto the finer grid consists of using the solution on the coarse grid for common nodes and computing the linear interpolation of the solution for nodes in the fine grid which are not present in the coarse grid.

Figure 7 shows the solution of the Stokes equations in terms of velocity vectors and pressure isobars. The velocity vectors show the flow circulation and the pressure isobars show a pressure minimum in the

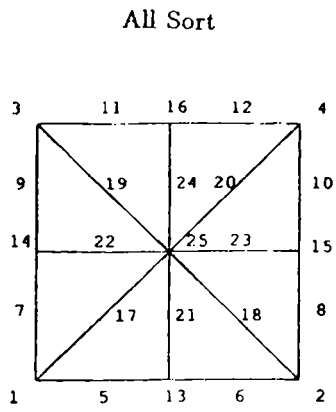


Figure 4. Node ordering after sorting the nodes with respect to their distance from the geometrical centre of the grid. The boundary nodes are numbered first and the centre node has the highest number

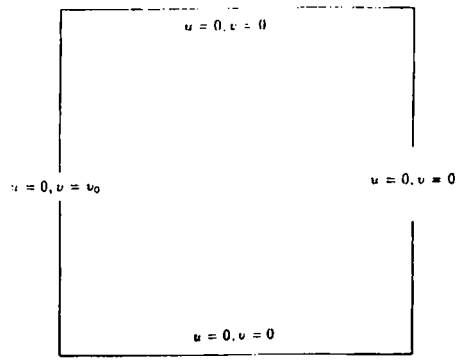


Figure 5. Test problem of two-dimensional cavity flow with boundary conditions. The velocities are zero at the walls, except for the left wall where a tangential velocity is specified

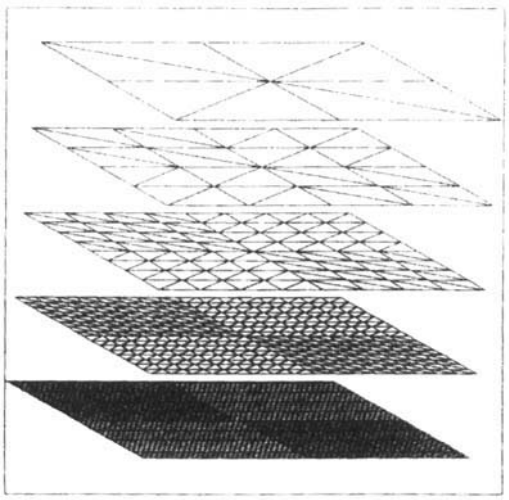


Figure 6. Hierarchy of grids used in computations. The initial grid is shown at the top and has eight finite elements with a total of nine corner nodes. At the next level of refinement each of these elements is divided into four new elements, giving a total of 32 elements and 25 corner nodes. The start vector for each finer grid is the solution from the coarser grid for common nodes. The start values for new nodes are found by linear interpolation

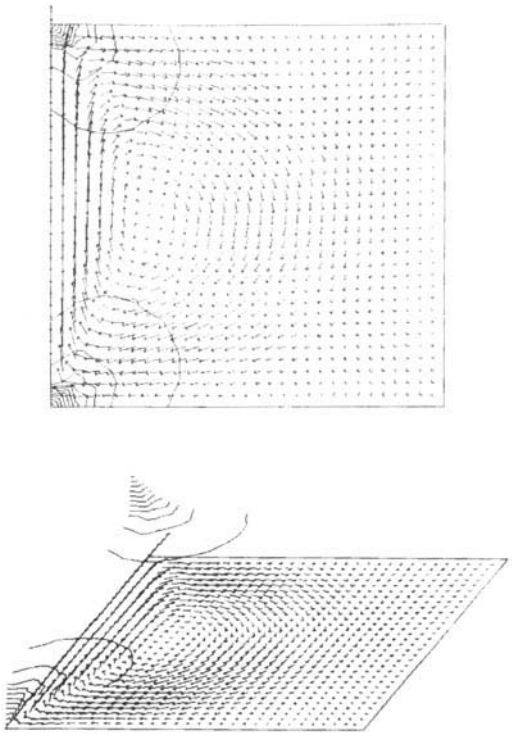


Figure 7. Two-dimensional solution of Stokes equations for cavity flow. The solution is presented in terms of pressure isobars and velocity vectors. The upper part of the figure shows the solution seen from above and the lower part shows the solution rotated in space. The height of each isobar corresponds to the pressure value at the isobar. The level of refinements is 4 with a 16 × 16 rectangular grid. The number of degrees of freedom is 2267

upstream part of the cavity and a pressure maximum at the downstream corner of the cavity. Both the pressure and flow show antisymmetric patterns with respect to a midline normal to the main direction of flow.

Figure 8 shows the solution of the Navier-Stokes equations for a Reynolds number of 500. The solution of the Navier-Stokes equations differs essentially from the solution of the Stokes equations. The antisymmetric patterns of the Stokes solution are completely absent. The pronounced pressure minimum at the upstream corner has disappeared, while the pressure maximum at the downstream corner is still present. The velocity circulation has become more circular in shape and the centre of the vortex has moved upstream and deeper into the cavity.

An overview of the cavity multigrid parameters is given in Table I. For each multigrid level of refinement the grid size and numbers of velocity nodes, pressure nodes and degrees of freedom are shown. In Table II the number of iterations and amount of work are shown for solving the Stokes equations for different refinement levels of the multigrid of the cavity problem. With the original method the zero vector is used as start vector for the Newton iterations. For the projection method the solution of the coarser grid is projected onto the finer grid and used as start vector for the non-linear iterations of the finer level. When the zero start vector is applied, it is possible to obtain a solution of the Stokes equations up to level 6. For finer grids the equation solver fails to converge. For levels 5 and 6 it is also necessary to restart the linear iterative solver to avoid stagnant solutions. The properties of the linear iterative equation solver with regard to stagnant and non-converging solutions are in accordance

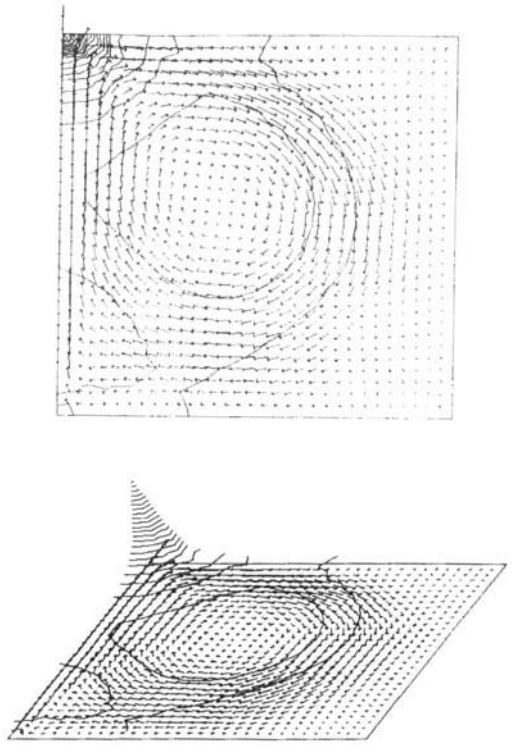


Figure 8. Two-dimensional solution of Navier–Stokes equations for cavity flow. The solution is presented in terms of pressure isobars and velocity vectors. The upper part of the figure shows the solution seen from above and the lower part shows the solution rotated in space. The height of each isobar corresponds to the pressure value at the isobar. The level of refinements is 4 with a 16 × 16 rectangular grid. The number of degrees of freedom is 2267

with the increasing condition number with grid refinements and increasing number of degrees of freedom of the linear equation system.

Table I. Parameters for the different grid levels. The first column shows the level of refinement. The second column shows the number of grid divisions in two dimensions for the cavity problem. The third and fourth columns show the numbers of velocity and pressure nodes respectively for the second-order finite element formulation. The last column show the number of degrees of freedom for each grid level

Level	Grid	Velocity nodes	Pressure nodes	Degrees of freedom
1	2 × 2	25	9	59
2	4 × 4	82	25	187
3	8 × 8	290	82	659
4	16 × 16	1090	290	2267
5	32 × 32	4225	1090	9539
6	64 × 64	16641	4225	37507
7	128 × 128	66049	16641	148739
8	256 × 256	263169	66049	592387
9	512 × 512	1,050625	263169	2364419

Table II. Number of iterations for solving the Stokes equations with the original method and the projection method. The initial and iterative amounts of work are shown in terms of number of multiplications $\times 10^{-3}$. The initial work is the work performed during the incomplete factorization with coupled node fill-in and the iterative work is the number of multiplications $\times 10^{-3}$ during one iteration. All nodes are sorted with respect to the geometrical centre of the grid for both the original and the projection method. The zero vector is used as start vector for the linear iterations in the original method and the solution from the coarser grid is used as start vector for the projection method. The linear convergence criterion is $\varepsilon_c = 10^{-4}$. The subscripts indicate the number of iterations between restarts of the iterative equation solver

	Iterations		Work	
Level	Original	Projection	Initial	Iterative
1	2	2	15	10
2	4	4	58	37
3	14	8	225	139
4	60	18	893	540
5	337_{30}	11	3553	2130
6	1861_{80}	9	14,180	8459
7	—	6	58,400	33,480
8	—	3	134,200	132,961
9	—			

The projection method shows a maximum number of iterations for multigrid level 4. For finer grids the number of iterations required to obtain a converged solution is decreased. At multigrid level 8 as few as three iterations are necessary for convergence. This property of the projection method is clearly explained by the fact that the projected solution from the coarser grid is very close to and almost within the convergence criterion of the solution of the finer grid. In contrast, the original method is incapable of reaching a converged solution at all, owing to the condition number of the Stokes equation system, for finer grids.

In Table III the number of iterations necessary for solving the Navier-Stokes equations for a Reynolds number of 500 is shown. In the experiments the solution of the Stokes equations is used as start vector for the Newton iterations. The numbers of linear iterations in these experiments are considerably less than those appearing in previous work.[5] The reduction in the number of iterations in this work is due to the sorting of the nodes. In the previous work the nodes were numbered row by row, first the corner nodes, then the midside nodes.

Table III. Number of Newton iterations and number of linear iterations within each Newton iteration for the original method, where the Stokes solution is used as start vector for the Newton iteration. The linear and non- linear convergence criteria are $\varepsilon = 10^{-4}$. The Reynolds number is 500. The nodes are all sorted with respect to distance from the geometrical centre of the grid

	Iterations										
Level	Newton	Linear									Sum
3	9	27	29	27	21	7	4	5	3	1	124
4	9	66	23	57	44	23	7	4	8	3	235
5	9	72	27	22	39	32	15	6	10	4	227
6	9	83	63	56	59	68	33	47	6	10	425

The effect of introducing non-linearities gradually into the equation system is demonstrated in Table IV. The start vector of the Newton iterations is the linear solution of the Stokes equations. The non-linearity of the equation system is introduced by increasing the density in steps of $\Delta\rho = 333$ from $\rho = 333$ to 1000. The final Reynolds number of 500 is then reached. The increasing density method requires fewer linear iterations, both in each Newton iteration and totally, to obtain convergence compared with the original Newton method, except for multigrid level 5. The increase in the number of iterations at level 5 may be caused by a small perturbation in the start vector or in the intermediate computation and experiments indicate that this is not a general property of the method.

The convergence properties of the projection method are shown in Table V. The numbers of linear iterations needed at all multigrid levels to achieve convergence are considerably less than for both the original Newton method and the increasing density method. The table also shows that the total number of convergent iterations is fairly constant and independent of refinement level. The total amount of work in obtaining the solution must include the work in obtaining the solution for all coarser grids. For example, based on the figures given in the table, the total amount of work in obtaining the solution with the original Newton method is approximately six times the amount of work in obtaining the solution with the projection method. The dominating amount of work for the projection method is the work required at the finest grid level. In order to compare the two methods, it is therefore sufficient to compare the numbers of iterations required by them at the same level of refinement.

Table IV. Number of Newton iterations and number of linear iterations within each Newton iteration for the original method, where the solution of the Stokes equations is used as start vector for the Newton iteration. The linear and non-linear convergence criteria are $\varepsilon = 10^{-4}$. The Reynolds number is 500. The density is increased for the first three Newton iterations. In the first Newton iteration the density is $\rho = 333$, in the second $\rho = 666$ and for the following iterations the density is $\rho = 1000$

	Iterations										
Level	Newton	Linear									Sum
3	9	19	17	25	23	6	4	5	3	3	105
4	9	22	12	13	26	7	3	3	3	1	90
5	9	49	26	69	12	49	21	12	4	11	253
6	9	104	24	46	27	17	44	23	49	10	344

Table V. Number of iterations and number of linear iterations within each Newton iteration for the projection method, where the projection from the coarser grid is used as start vector for the Newton iteration. The linear and non-linear convergence criteria are $\varepsilon = 10^{-4}$. The Reynolds number is 500

	Iterations								
Level	Newton	Linear							Sum
4	4	43	29	12	3	0	0	0	87
5	7	15	16	15	6	3	5	3	70
6	7	11	21	5	6	18	8	5	74
7	7	21	12	7	3	7	3	5	58
8	7	6	21	19	4	5	5	13	73

DISCUSSION

The goal of this work has been to develop a solution algorithm for the Navier–Stokes equations which is robust, fast and sparse. The robustness is attached to the implicit solution techniques for the differential equation system. The speed of the algorithm is tied to the computer time needed. The sparsity is linked to the storage requirements of the algorithm. The adaptive multigrid method described in this paper seems to some extent to have these properties.

In the present paper an adaptive multigrid method for solving the Navier–Stokes equations is developed. The adaptive multigrid algorithm may be considered as consisting of five essential parts: multigrid generation, adaptive refinement, matrix integration, intergrid transition and adaptive equation solver.

The multigrid generation is based on the tri-tree algorithm, which permits the construction of a finite element grid at each tree level. The tri-tree algorithm allows for adapting the grid both to irregular geometry and to the solution of the system of differential equations. The matrix generation is executed by analytic integration and is therefore fast enough for the coefficients in the equation matrix to be easily generated whenever needed in the solution algorithm. The transition from coarse to fine grids is direct for the common nodes and linear interpolation is used for the new nodes.

The most important property of the adaptive multigrid algorithm is that when the grid is sufficiently refined, the start vector is almost within the solution tolerance and only a few iterations are needed. For more complex boundary conditions and geometries when local spatial refinement is needed, this property can be used to obtain an accurate solution where large gradients in the solution occur.

The Navier–Stokes equations can be considered as a composite of the Stokes equations and the Euler equation. The solution of the Stokes equation is, for relatively smooth boundaries and smooth initial conditions, in some sense a smooth solution. The coarseness of a regular grid in the entire computational domain should then be determined by the accuracy required in this smooth solution. The solution of the Euler equation is of a finite different nature. The solution varies from some rapid decay or rise to a shock appearing inside a limited number of finite elements, but is reasonably smooth or constant elsewhere. When a large gradient in the solution is due to an initial condition, the discontinuity is transmitted through the domain. When a large gradient in the solution value is caused by an irregular boundary shape, the solution is stationary within the computational domain. In both these situations the grid has to be refined locally around gradients in order to achieve an appropriate accuracy of the solution at the sites where these gradients occur.

The results of the present work show that owing to the increase in condition number of the Stokes equations with refinements of the multigrid, it becomes more and more difficult to achieve a convergent solution as the multigrid is refined. In contrast, it is advantageous to have a fine grid in order to linearize and symmetrize the Navier–Stokes equations owing to the non-linear convection term. These investigations indicate that the conjugate gradient algorithm together with the original Newton formulation of the Navier–Stokes equations is not capable of solving the Navier–Stokes problem for very fine grids or very large non-linearities. The projection algorithm developed in the present paper overcomes these shortcomings of the original Newton method and reveals properties which resolve the competitive requirements arising from the condition number and the non-linearity of the equation system.

Further research will concentrate on adaptive tri-tree multigrid structures for irregular grids and local multigrid adaptation to discontinuities in both the solution and the boundary geometry.

ACKNOWLEDGEMENT

The author is grateful to Olav Dahl for discussions of numerical methods.

REFERENCES

1. T. J. R. Hughes, L. P. Franca and G. M. Hulbert, 'A new finite element formulation for computational fluid dynamics. VIII. The Galerkin/least-squares method for advective–diffusive equations', *Comput. Methods Appl. Mech. Eng.*, **73**, 173–189 (1989).
2. T. Utnes, 'Finite element modeling of quasi-three dimensional nearly horizontal flow', *Int. j. numer. methods fluids*, **12**, 559–576 (1991).
3. S. Ø. Wille, 'Numerical simulations of steady flow inside a three dimensional aortic bifurcation model', *J. Biomed. Eng.*, **6**, 49–55 (1984).
4. E. Barragy and G. F. Carey, 'A partitioning scheme and iterative solution for sparse bordered systems', *Comput. Methods Appl. Mech. Eng.*, **70**, 321–327 (1988).
5. O. Dahl and S. Ø. Wille, 'An ILU preconditioner with coupled node fill-in for iterative solution of the mixed finite element formulation of the 2-D and 3-D Navier–Stokes equations', *Int. j. numer. methods fluids*, **15**, 525–544 (1992).
6. S. Ø. Wille, 'Pulsatile pressure and flow in arterial aneurysm simulated in a mathematical model', *J. Biomed. Eng.*, **3**, 153–158 (1981).
7. S. Ø. Wille, 'A preconditioned alternating inner–outer iterative solution method for the mixed finite element formulation of the Navier–Stokes equations', *Int. j. numer. methods fluids*, **18**, 1135–1151 (1994).
8. A. George and J. W. Liu, *Computer Solutions of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
9. P. K. W. Vinsome, 'Orthomin, an iterative method for solving sparse sets of simultaneous linear equations', *Proc. Fourth Symp. on Reservoir Simulation*, Society of Petroleum Engineers of AIME, New York, 1976, pp. 147–159.
10. D. M. Young and K. C. Yea, 'Generalized conjugate-gradient acceleration of non-symmetrizable iterative methods', *Lin. Alg. Appl.*, **34**, 159–194 (1980).
11. P. Sonneveld, 'CGS, a fast Lanczos-type solver for non-symmetric linear systems', *SIAM J. Sci. Stat. Comput.*, **10**, 36–52 (1987).
12. H. A. van der Vorst, 'Bi-CGSTAB: a fast and smoothly converging variant of Bi-CG for the solution of non-symmetric linear systems', *SIAM J. Sci. Stat. Comput.*, in press.
13. G. F. Carey, K. C. Wang and W. D. Joubert, 'Performance of iterative methods for Newtonian and generalized Newtonian flows', *Int. j. numer. methods fluids*, **9**, 127–150 (1989).
14. C. Vincent and R. Boyer, 'A preconditioned conjugate gradient Uzawa- type method for the solution of the Stokes problem by mixed Q1–P0 stabilized finite elements', *Int. j. numer. methods fluids*, **14**, 289–298 (1992).
15. O. G. Johnson, C. A. Michelli and G. Paul, 'Polynomial preconditioners for conjugate gradient calculations', *SIAM J. Numer. Anal.*, **20**, 362–376 (1983).
16. S. Ø. Wille, 'A structured tri-tree search method for generation of optimal unstructured finite element grids in two and three dimensions', *Int. j. numer. methods fluids*, **14**, 861–881 (1992).
17. J. Peraire, M. Vadati, K. Morgan and O. Zienkiewicz, 'Adaptive remeshing for compressible flow computations', *J. Comput. Phys.*, **71**, 449–466 (1987).
18. R. Lohner, K. Morgan and O. C. Zienkiewicz, 'An adaptive finite element procedure for compressible high speed flows', *Comput. Methods Appl. Mech. Eng.*, **51**, 441–464 (1985).
19. W. J. Schroeder and M. S. Shepard, 'A combined octree/Delauney method for fully automatic 3-D mesh generation', *Int. j. numer. methods eng.*, **29**, 37–55 (1990).
20. H. K. Ruud and S. Ø. Wille, 'An advancing front algorithm for three dimensional mesh generation', *Proc. NUMETA 90, Numerical Methods in Engineering: Theory and Applications*, January 1990.
21. C. Taylor and P. Hood, 'A numerical solution of the Navier–Stokes equations using the finite element technique', *Comput. Fluids*, **1**, 73–100 (1973).
22. W. Hackbush, *Multi-Grid Methods and Applications*, Springer, Berlin, 1985.
23. S. Ø. Wille, 'An adaptive unstructured tri-tree iterative solver for mixed finite element formulation of the Stokes equations', *Int. j. numer. methods fluids*, **22**, 899–913 (1996).