INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 22, 899–913 (1996)

AN ADAPTIVE UNSTRUCTURED TRI-TREE ITERATIVE SOLVER FOR MIXED FINITE ELEMENT FORMULATION OF THE STOKES EQUATIONS

S. Ø. WILLE

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

SUMMARY

An iterative adaptive equation solver for solving the implicit 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 grid the Stokes equations are solved. The approximate 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.

In order to reduce the overall work, the element matrices are integrated analytically beforehand. 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.

The efficiency of the incomplete coupled node fill-in preconditioner is shown to be largely dependent on the global node numbering. The preconditioner is therefore tested for the natural node ordering of the tri-tree grid generator and for different ways of sorting the nodes.

KEY WORDS: grid generation; tri-tree; unstructured grid; finite elements; mixed formulation; analytic integration; adaptive solver; 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] For implicit solution algorithms,[4–6] direct equation solvers have shown limitations due to rather large computer storage and computer time requirements.[7] 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.[8–12] 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[5,8] 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

CCC 0271–2091/96/090899–15

Received October 1994
Revised August 1995

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.[8]

In the present work the global order of node numbering was found to play an important role in the convergence rate. Several node-ordering sequences have therefore been tested. An advantageous node-ordering scheme seems to be to number the nodes in such a way that during the incompleted elimination there will be no more contribution to the element matrix from already eliminated nodes. The most efficient node ordering for incomplete coupled node fill-in preconditioning seems to be to number the nodes from the periphery of the finite element grid towards the centre of the grid in an increasing number sequence.

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 approximately 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 points 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.

The approximate solution procedure for the Navier–Stokes equations is more complicated than for other positive definite systems owing to the zero diagonal block in the equation matrix. In previous papers[5,8] the coupled node fill-in LU factorization was designed and applied as preconditioning for the Stokes and Navier–Stokes equations. In the same papers the Bi-CGSTAB conjugate gradient algorithm proved to work well also for equation systems which were not positive definite. In the Bi-CGSTAB smoothing algorithm the element matrices are only needed in matrix–vector multiplication. The matrix–vector product can be done node by node for each element and the equation matrix need not be assembled and stored. The matrix coefficients are generated whenever needed.

EQUATIONS

The Stokes equations are linear and are given by

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

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

where $\mathbf{v}$ 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, two different orders of basis functions are applied for approximating velocities and pressure. With the first-order basis functions the velocities are approximated by linear polynomials and the pressure is considered constant on each element. With the second-order basis functions the velocites are approximated with quadratic basis functions and the pressure is approximated with linear basis functions on each element.[21] The Babuska–Brezzi condition is satisfied for both these finite element formulations. Denote the quadratic polynomials by N_i, the linear polynomials by L_i and the constant polynomial on each element by K_e. Then by the Galerkin residual method and integration by parts the first-order finite element formulation of the Stokes equation system becomes

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

The second-order finite element formulation of the Stokes equation system becomes

$$\begin{aligned}\mathbf{F}_\mathbf{v} &= \int_\Omega \mu\nabla N_i\cdot\nabla\mathbf{v}\,\mathrm{d}\Omega - \int_\Omega \nabla N_i p\,\mathrm{d}\Omega - \int_{\delta\Omega}\mu N_i\frac{\partial\mathbf{v}}{\partial n}\,\mathrm{d}\delta\Omega + \int_{\delta\Omega} N_i p\,\mathrm{d}\delta\Omega = 0,\\ \mathbf{F}_\mathbf{p} &= -\int_\Omega L_i\nabla\cdot\mathbf{v}\,\mathrm{d}\Omega = 0.\end{aligned} \tag{4}$$

The following equation system can then be solved for the first-order formulation:

$$\begin{bmatrix}\displaystyle\int_\Omega \mu\nabla L_i\nabla L_j\,\mathrm{d}\Omega & \displaystyle-\int_\Omega \nabla L_i K_e\,\mathrm{d}\Omega\\ \displaystyle-\int_\Omega K_e\nabla L_j\,\mathrm{d}\Omega & 0\end{bmatrix}\begin{bmatrix}\mathbf{v}\\ \mathbf{p}\end{bmatrix} = -\begin{bmatrix}\displaystyle\int_{\delta\Omega}\left(-\mu L_i\frac{\partial\mathbf{v}}{\partial n} + L_i p\right)\mathrm{d}\delta\Omega\\ 0\end{bmatrix}. \tag{5}$$

For the second-order formulation the equation system becomes

$$\begin{bmatrix} \int_\Omega \mu \nabla N_i \nabla N_j \, d\Omega & -\int_\Omega \nabla N_i L_j \, d\Omega \\ -\int_\Omega L_i \nabla N_j \, d\Omega & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{p} \end{bmatrix} = -\begin{bmatrix} \int_{\delta\Omega} \left(-\mu N_i \frac{\partial \mathbf{v}}{\partial n} + N_i p \right) d\delta\Omega \\ 0 \end{bmatrix}. \tag{6}$$

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

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

where the nodes j and k are the corner nodes on each side of the midside node n. The local numbering of the nodes is shown in Figure 2 (see next section). 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

$$\int_A L_i^\alpha L_j^\beta L_k^\gamma L_l^\delta = \frac{\alpha!\beta!\gamma!}{(\alpha+\beta+\gamma+\delta+n_d)!} n_d! \Omega.$$

The integrals appearing in the matrices then become

$$\int_\Omega L_i \, d\Omega = \frac{\Omega}{n_d + 1},$$

$$\int_\Omega L_i^2 \, d\Omega = \frac{2\Omega}{(n_d+1)(n_d+2)},$$

$$\int_\Omega L_i L_j \, d\Omega = \frac{\Omega}{(n_d+1)(n_d+2)}, \quad i \neq j.$$

In the formulae below the δ-function is defined by

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

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 first-order basis function formulation the integrals of the derivatives in the equation matrix are given by

$$i \leqslant n_c, \qquad j \leqslant n_c,$$

$$\int_\Omega \frac{\partial L_i}{\partial x} \frac{\partial L_j}{\partial x} \, d\Omega = \frac{\partial L_i}{\partial x} \frac{\partial L_j}{\partial x} \Omega,$$

$$\int_\Omega K_e \frac{\partial L_j}{\partial x} \, d\Omega = K_e \frac{\partial L_j}{\partial x} \Omega. \tag{7}$$

In the second-order basis function formulation the integrals of the derivatives in the equation matrix are given by

$$i \leqslant n_c, \qquad j \leqslant n_c,$$

$$\int_\Omega \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}\,d\Omega = [16(\delta_{ij}+1) - 8(n_d+2) + (n_d+1)(n_d+2)]\frac{\partial L_i}{\partial x}\frac{\partial L_j}{\partial x}\frac{\Omega}{(n_d+1)(n_d+2)},$$
$$\int_\Omega L_i \frac{\partial N_j}{\partial x}\,d\Omega = [4(\delta_{ij}+1) - (n_d+2)]\frac{\partial L_j}{\partial x}\frac{\Omega}{(n_d+1)(n_d+2)}, \tag{8}$$

$$n \leqslant n_c, \qquad n_c < m \leqslant n_e,$$

$$\int_\Omega \frac{\partial N_n}{\partial x}\frac{\partial N_m}{\partial x}\,d\Omega = 4\frac{\partial L_n}{\partial x}\left[4(\delta_{nj}+1)\frac{\partial L_i}{\partial x} + 4(\delta_{ni}+1)\frac{\partial L_j}{\partial x} - \left(\frac{\partial L_i}{\partial x}+\frac{\partial L_j}{\partial x}\right)(n_d+2)\right]\frac{\Omega}{(n_d+1)(n_d+2)},$$
$$\int_\Omega L_n \frac{\partial N_m}{\partial x}\,d\Omega = 4\left((\delta_{ni}+1)\frac{\partial L_j}{\partial x} + (\delta_{nj}+1)\frac{\partial L_i}{\partial x}\right)\frac{\Omega}{(n_d+1)(n_d+2)}, \tag{9}$$

$$n_c < p \leqslant n_e, \qquad n_c < q \leqslant n_e,$$

$$\int_\Omega \frac{\partial N_p}{\partial x}\frac{\partial N_q}{\partial x}\,d\Omega = 16\Big((\delta_{in}+1)\frac{\partial L_j}{\partial x}\frac{\partial L_m}{\partial x} + (\delta_{jn}+1)\frac{\partial L_i}{\partial x}\frac{\partial L_m}{\partial x}$$
$$+(\delta_{im}+1)\frac{\partial L_j}{\partial x}\frac{\partial L_n}{\partial x} + (\delta_{jm}+1)\frac{\partial L_i}{\partial x}\frac{\partial L_n}{\partial x}\Big)\frac{\Omega}{(n_d+1)(n_d+2)}. \tag{10}$$

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, continuity and pressure gradient can be computed exactly. For second-order polynomial approximation, all these terms are integrals of second-order polynomials.

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.

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.

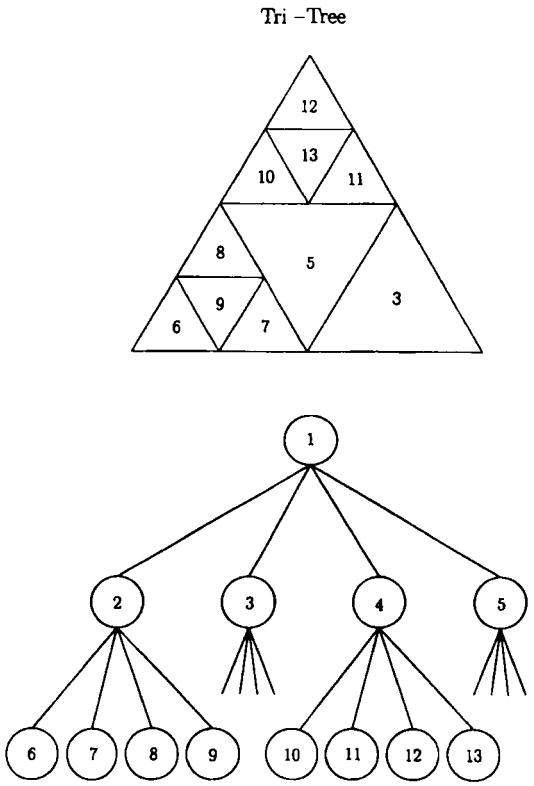


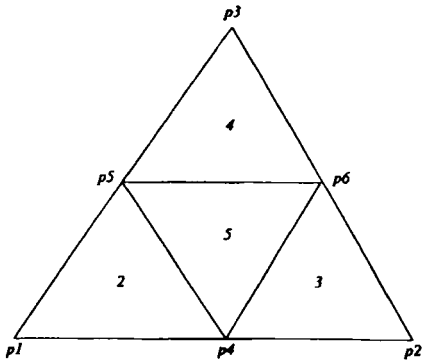
Figure 1. Hierarchical structure of the tri-tree. An initial equilateral triangle is divided into four new equilateral triangles. Each of these triangles can then be divided into another four equilateral triangles, and so on. The tree structure of these divisions is shown in the lower part of the figure. 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

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 neighbouring 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.

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

$$P = P_g + (P_g - P_c)/d + \varepsilon(P_g - P_c). \tag{11}$$



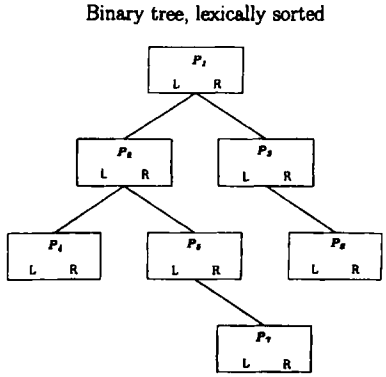
	Level	Corners			Pointers to divisions				Parent
1	1	p1	p2	p3	2	3	4	5	0
2	-2	p1	p4	p5	5	0	0		1
3	-2	p4	p2	p6	0	5	0		1
4	-2	p5	p6	p3	0	0	5		1
5	-2	p6	p4	p5	2	4	3		1

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 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

$$P_g + (P_g - P_c)/d \neq P_g + (P_g - P_c)/d + \varepsilon(P_g - P_c) \tag{12}$$

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,



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 generated during the refinement procedure is already present in the tree structure

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 grid 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

$$\mathbf{F}^k(\mathbf{x}^k) = \mathbf{b}^k. \tag{13}$$

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 algorithm is then defined by

```
Choose x̃^k
for {k = 1; k (= N − 1; k + +}
{
    x^k = x^k + P^k(x^{k−1} − x̃^{k−1});
    Smooth(x^k, x̃^k);
}
Solve F^N(x^N) = 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 grid 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 algorithm is the global smoothing method. The special problem which arises with the Navier–Stokes equations is the zero diagonal block[5,8] 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 grid level the solution is determined fully converged. When the equation system is solved for the finest grid, the adaptive cycle is complete.

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

$$\frac{\|\mathbf{r}^k\|}{\|\mathbf{x}^k\|} < \varepsilon_g, \tag{14}$$

where $\mathbf{r}^k$ is the residual and $\mathbf{x}^k$ is the solution vector at grid level k. The complete adaptive iteration is stopped by the same convergence criterion with $\varepsilon = 10^{-4}$.

NODE-ORDERING SCHEME

The node numbering for the tri-tree grid generator is shown in Figure 4a. 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. The other way of node ordering which has been investigated is based on sorting nodes. In the sorting algorithm the nodes are sorted with respect to their distance from the centre of the grid. The node which is furthest away from the centre is given the smallest number. In the first ordering scheme, where all nodes are sorted, Figure 4b, the nodes are sorted regardless of whether they are corner or midside nodes. In the second ordering scheme, Figure 4c, the corner and midside nodes are sorted separately and the corner nodes are numbered before the midside nodes. In Figure 4d the corner and midside nodes are again sorted separately but the midside nodes are numbered before the corner nodes.

NUMERICAL EXPERIMENTS

The test problem is channel flow with the boundary conditions shown in Figure 5. The velocities are set to zero at the walls and a parabolic velocity profile is imposed at the inlet. The Reynolds number for the

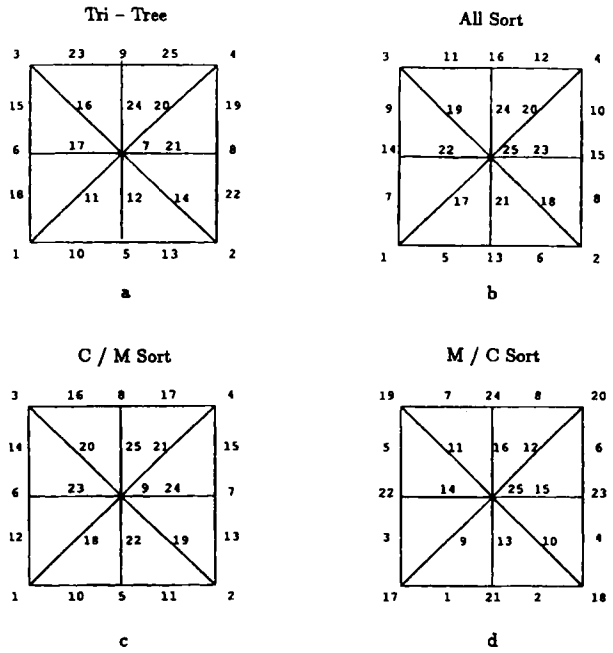


Figure 4. Different methods for global numbering of nodes: a, the numbering of nodes achieved by tri-tree grid generator; b, all the nodes are sorted with respect to distance from centre of grid; c, corner and midside nodes are sorted separately and corner nodes numbered first; d, corner and midside nodes are sorted separately and midside nodes numbered first

channel flow is 1000. The successive grids used at each refinement level are shown in Figure 6. Experiments are performed for both first- and second-order basis functions approximating the solution. With first-order basis functions the velocities are approximated by linear polynomials and the pressure is approximated by a constant on each element. With second-order basis functions the velocities are approximated by quadratic polynomials and the pressure is approximated by linear polynomials on each element. The pressure and velocities are shown in Figure 7 for the second-order polynomial approximation at grid level 3 and Reynolds number 1000. At each level of refinement the finite element equatin system is solved iteratively by Bi-CGSTAB preconditioned by incomplete factorization with coupled node fill-in. The original algorithm is obtained by using the zero vector as start vector for the iterative solution procedure. The projection algorithm uses the projected solution from the coarser grid as start vector. The projection algorithm will therefore have a start vector which is much closer to the solution vector than that of the original algorithm. The convergence criterion at all grid level is set to $\varepsilon = 10^{-4}$.

The numerical experiments are performed for first-order and second-order basis functions. The effect of sorting the nodes for the first-order basis functions is shown in Table I. The results in this table indicate that the number of iterations necessary to reach convergence is considerably reduced when the nodes are sorted compared with the case when the nodes appear in tri-tree order. For the most refined grids, levels 4 and 5, the iterations became stagnant and the iterative equation solved had to be restarted.

Table II shows the effect of sorting the nodes for the second-order basis functions. Three ways of sorting the nodes are investigated. The first method consists of sorting all the nodes regardless of whether they are corner or midside nodes. Compared with tri-tree node ordering, all sorting methods needed fewer iterations for convergence. The number of iterations to convergence for the three sorting methods did not differ significantly. For the most refined grid, level 5, it was also necessary to restart the iterations for the second-order basis functions.

The results of using the projection of the solution from the coarser grid as start vector for the first-order basis functions are shown in Table III. All the nodes are sorted in these experiments. The velocity solution from the coarser grid is interpolated linearly. However, as the pressure is constant on each element, is was difficult to compute a good projection for the pressure which both increased the convergence rate and converged accurately enough to the right solution. In fact, the best starting value for the pressure was the zero vector. The behaviour of the equation solver is very much improved by the projection algorithm, especially for the finest grids. For the most refined grids it was not possible to obtain a converged solution at all without using the projected solution of the velocities as start vector.

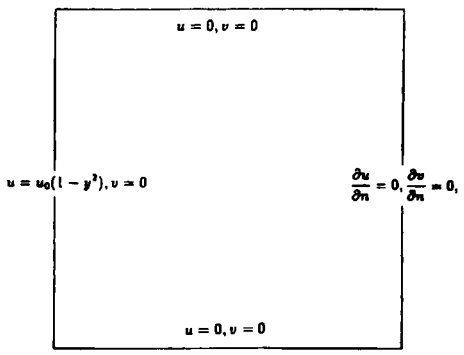


Figure 5. Test problem of channel flow with boundary conditions. The velocities are zero at the walls. At the inlet a parabolic velocity profile is introduced

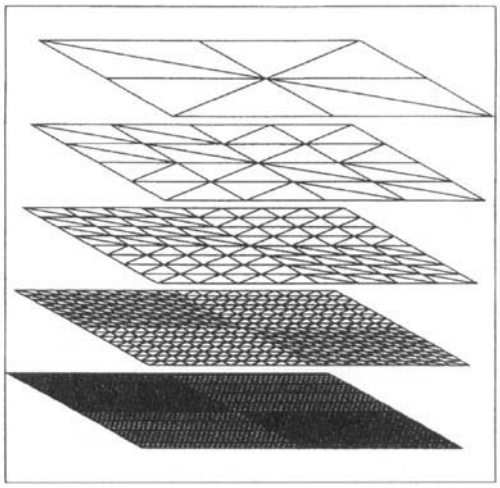


Figure 6. Hierarchy of grids used in computations. The initial grid is shown at the top and has eight finite elements with a total of nine corner nodes. At the next level of refinement each of these elements is divided into four new elements, giving a total of 32 elements and 25 corner nodes. The start 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

The simulation results for the projection algorithm are shown in Table IV. For the original solution algorithm where the zero vector is used as start vector, all nodes are sorted. For the projection algorithm the three different sorting methods are applied. Again the three sorting methods reveal no significant differences with respect to convergence rate. The projection algorithm is considerably faster for medium grid size and converges with as little as one linear iteration for the finest grids. The reason for convergence within one iteration is of course due to the fact that the start vector is within the tolerance of the final solution. However, in contrast, the original method does not converge at all. The total amount of work in obtaining the solution by the projection method must for completeness include the work in

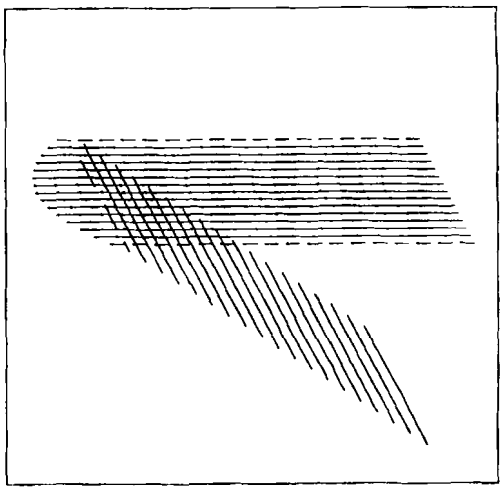


Figure 7. Pressure and velocities of channel flow for Reynolds number 1000 at grid level 3. The pressure is displayed as isobars and the velocities are shown as velocity vectors

Table I. Number of iterations for different grids for first-order basis functions. The first column shows the level of refinement of the grid. The second column shows the number of finite elements in each co-ordinate direction. The third column shows the number of degrees of freedom for each refinement level. The last two columns show the number of iterations to convergence for the tri-tree node ordering and for all nodes sorted, respectively

Level	Grid	Deg. free.	Tri-tree	All sort
2	4 × 4	82	13	10
3	8 × 8	290	37	21
4	16 × 16	1090	113	59_{30}
5	32 × 32	4226	334_{70}	243_{70}

Table II. Number of iterations for different grids for second-order basis functions. The first column shows the level of refinement of the grid. The second column shows the number of finite elements in each co-ordinate direction. The third column shows the number of degrees of freedom for each refinement level. The last columns show the number of iterations to convergence for the tri-tree node ordering, for all nodes, for the corner and midside nodes sorted separately with the corner nodes numbered first and for the corner and midside nodes sorted separately with the midside nodes numbered first, respectively

Level	Grid	Deg. free.	Tri-tree	All sort	C/M sort	M/C sort
1	2 × 2	57	6	3	5	6
2	4 × 4	187	6	5	5	6
3	8 × 8	659	19	12	14	11
4	16 × 16	2467	89	40	36	37
5	32 × 32	9539	417_{50}	206_{50}	229_{50}	205_{50}

Table III. Number of iterations and computational work for different grids with and without using the projection of the solution from coarser grid as start vector in the iterations. In both cases the iterative equation solver uses incomplete factorization with coupled node fill-in. In these experiments a set of first-order basis functions is used, with linear approximation for the velocities and constant approximation for the pressure on each element. The first column shows the level of grid refinement. The second column shows the number of degrees of freedom. The number of original iterations is found when the zero vector is used as start vector. The number of projected iterations is obtained by using the projection of the solution from the coarser grid as start vector. The last two columns show work in terms of number of multiplications × 10^{-3}. The initial work is performed during the incomplete coupled node fill-in factorization. The iterative work is performed during one linear iteration. The subscript numbers in the third and fourth columns indicate the number of iterations between restarts of the iterative equation solver

				Work	
Level	Deg. free.	Original all sort	Projection all sort	initial	iterative
2	82	10	10	5	8
3	290	21	20	18	30
4	1090	59_{30}	31	72	117
5	4226	243_{70}	130_{50}	286	460
6	16,642	—	418_{50}	1142	1819

Table IV. Same parameter values as in Table III for second-order basis functions used in the element formulation. The velocities are approximated by quadratic basis functions and the pressure is approximated by linear basis functions on each element. The third column shows the number of iterations needed when all nodes are sorted using zero start vector. In the next three columns the projected solution from the coarser grid is used as start vector. Three different sorting methods for the projection algorithm are applied, namely all nodes sorted, the corner and midside nodes sorted separately with the corner nodes numbered first and the corner and midside nodes sorted separately with the midside nodes numbered first. Note that the number of iterations to obtain the solution increases until level 4 when using the projection solution as start vector. Then the number of iteration starts to decrease. This phenomenon is explained in the text. The difference sorting methods do not result in a significant difference in number of iterations to convergence. The two last columns show work in terms of number of multiplications $\times 10^{-3}$. The initial work is performing the incomplete coupled node fill-in decomposition. The iterative work is the number of multiplications $\times 10^{-3}$ performed during one linear iteration. The subscript number in the third column indicates the number of iterations between restarts of the iterative equation solver

			Projection			Work	
Level	Deg. free.	Original all sort	All sort	C/M sort	M/C sort	Initial	Iterative
1	59	3	3	3	3	15	10
2	187	5	4	4	4	58	37
3	650	12	9	8	7	225	139
4	2467	40	15	15	17	893	540
5	9539	206_{50}	9	4	5	3553	2130
6	37,507	—	1	1	1	14,180	8459
7	148,739	—	1	1	1	58,400	33,480

obtaining the solution for all coarser grids. Even then, the amount of work to be performed is favourable, as the sum of work executed for the coarser grids is of the same order of magnitude as for the finest grid. All experiments in this investigation have been performed on a standard workstation and the computing time is of the order of minutes.

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 method described in this paper seems to some extent to have these properties.

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

The grid 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 and linear interpolation is used for the midside nodes. The iterative solver, Bi- CGSTAB, allows for zero diagonal blocks in the equation matrix.

The most important property of the adaptive algorithm is that when the grid is sufficiently refined, the start vector is 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.

Further work with adaptive tri-tree grid structures for irregular grids will consider non-linear adaptive iterative solvers, linear and non-linear multigrid methods, local grid adaptation to discontinuities in both the solution and the boundary geometry and hyperbolic upwinding schemes.

ACKNOWLEDGEMENTS

The author is grateful to Britt von Krogh for corrections of the manuscript and 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. A. George and J. W. Liu, *Computer Solutions of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
8. 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).
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.