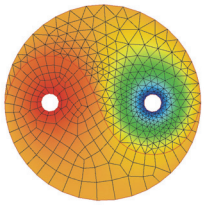
Publishing House
AKAPIT

COMPUTER METHODS IN MATERIALS SCIENCE

Informatyka w Technologii Materiałów

Vol. 11, 2011, No. 1



TOWARDS USING ADAPTIVE HYBRID MESHES IN FEM SIMULATIONS OF FLOW IN ARTIFICIAL HEART CHAMBERS

KRZYSZTOF BANAŚ*, PAWEŁ CYBUŁKA, PIOTR MACIOŁ, KAZIMIERZ MICHALIK, PRZEMYSŁAW PŁASZEWSKI

AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Kraków, Poland
**Corresponding author: banas@metal.agh.edu.pl*

Abstract

The paper presents an implementation of adaptive hybrid meshes in a FEM code designed for 3D simulations of flow in artificial heart chambers. Special emphasis is put on taking into account moving parts of artificial heart chambers, like valves. Toward this goal a weak formulation for problems with moving boundaries and algorithms for mesh modifications, including remeshing, are developed.

Key words: incompressible flow, finite element method, mesh movement, remeshing, artificial heart chamber

1. FLOW SIMULATIONS WITH MOVING BOUNDARIES

One of important factors in the design of artificial heart chambers is to take into consideration the moving parts of the chamber – valves and possibly membranes. Numerical simulations aiding the design process should properly resolve fluid-structure interaction, which for flow part of simulations means resolving flows with moving boundaries. To this end proper formulations have to be utilized and special technical means in simulation codes applied.

1.1. Finite element formulation for incompressible flows with moving boundaries

The Navier-Stokes equations of incompressible fluid flow consist of two equations for unknown velocities $\boldsymbol{u}(\boldsymbol{x}, t)$ and pressure $p(\boldsymbol{x}, t)$:

$$\begin{aligned} \boldsymbol{u} &= \hat{\boldsymbol{u}}_0 \quad \text{on } \Gamma_D \\ (\nu\boldsymbol{\nabla u})\boldsymbol{n} - p\boldsymbol{n} &= \boldsymbol{g} \quad \text{on } \Gamma_N \end{aligned}$$

accompanied by the boundary conditions:

The derivation of the formulation for simulations with moving boundaries consist of several steps that include: applying the standard finite element procedures of multiplying by test functions $\boldsymbol{w}(\boldsymbol{x}, t)$ and $q(\boldsymbol{x}, t)$, integrating over the space-time computational domain $(0,\Delta t)\times\Omega$, transformation from the variable space domain to the invariable mesh domain and consistent application of the change of variables from the space coordinates $\boldsymbol{x}$ to the mesh coordinates $\boldsymbol{\chi}$. The final formulation is the following:

Find such functions $\boldsymbol{u}(\boldsymbol{\chi}, t)$ and $p(\boldsymbol{\chi}, t)$ that for every pair of test functions $\boldsymbol{w}(\boldsymbol{\chi}, t)$ and

$$\begin{aligned} \frac{\partial \boldsymbol{u}}{\partial t} + (\boldsymbol{u}\cdot\boldsymbol{\nabla})\boldsymbol{u} - \nu\boldsymbol{\nabla}^2\boldsymbol{u} + \boldsymbol{\nabla}p &= \boldsymbol{f} \quad \text{in } \Omega \\ \boldsymbol{\nabla}\cdot\boldsymbol{u} &= 0 \quad \text{in } \Omega \end{aligned}$$

$q(X, t)$ the following holds:

ISSN 1641-8581

$$\int_0^{\Delta t}\int_\Omega \left(\frac{\partial \boldsymbol{u}}{\partial t}+\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\chi}}\left(\boldsymbol{I}+\frac{\partial \hat{\boldsymbol{v}}(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}}\cdot t\right)^{-1}(\boldsymbol{u}-\hat{\boldsymbol{v}}(\boldsymbol{\chi}))\right)\boldsymbol{w}|\det \boldsymbol{J}|d\Omega dt$$
$$+\int_0^{\Delta t}\int_\Omega \left(\nu\frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\chi}}\cdot\left(\boldsymbol{I}+\frac{\partial \hat{\boldsymbol{v}}(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}}\cdot t\right)^{-1}-p\boldsymbol{I}\right)\frac{\partial \boldsymbol{w}}{\partial \boldsymbol{\chi}}\cdot\left(\boldsymbol{I}+\frac{\partial \hat{\boldsymbol{v}}(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}}\cdot t\right)^{-1}|\det \boldsymbol{J}|d\Omega dt$$
$$+\int_0^{\Delta t}\int_\Omega \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{\chi}}\left(\boldsymbol{I}+\frac{\partial \hat{\boldsymbol{v}}(\boldsymbol{\chi})}{\partial \boldsymbol{\chi}}\cdot t\right)^{-1}q|\det \boldsymbol{J}|d\Omega dt+\int_\Omega(\boldsymbol{u}(\boldsymbol{\chi},0)-\hat{\boldsymbol{u}}(\boldsymbol{\chi},0))\boldsymbol{w}(\boldsymbol{\chi},0)d\Omega$$
$$=\int_0^{\Delta t}\int_\Omega \boldsymbol{f}\boldsymbol{w}|\det \boldsymbol{J}|d\Omega dt-\int_0^{\Delta t}\int_{\Gamma_N}\boldsymbol{g}\boldsymbol{w}|\det \boldsymbol{J}_S|dSdt$$

where ˆv is the constant in time mesh velocity, ˆu initial velocity for time step and $\boldsymbol{J}$ denotes the Jacobian of the transformation from coordinates $\boldsymbol{\chi}$, local to the mesh, to space coordinates $\mathbf{x}$ (with $\boldsymbol{J}_S$ being its counterpart for 2D integration over boundary faces). The derivatives with respect to vector parameters denote the respective Jacobian matrices.

In actual computations linear and multi-linear (depending upon the element type) approximations are used for each component of vector functions $\boldsymbol{u}(\boldsymbol{\chi}, t)$ and $\boldsymbol{w}(\boldsymbol{\chi}, t)$ as well as scalar functions $p(\boldsymbol{\chi}, t)$ and $q(\boldsymbol{\chi}, t)$. Moreover, the formulation is augmented with stabilizing terms in a standard SUPG-GLS fashion (Franca and Frey (1992)).

1.2. Mesh module for 3D adaptive hybrid meshes.

To manage 3D adaptive hybrid (tetrahedral-prismatic) meshes within a finite element code designated to perform incompressible fluid flow simulations according to the presented above formulation, a special mesh module (as described by Banaś (2004)) was designed and implemented. In its current version it has support for 1-irregular hybrid meshes containing prisms and tetrahedrons and a new organization of data structures, briefly described below.

The design objectives for the storage scheme for possibly 1-irregular meshes (as described by Demkowicz (1989) and Paszyński (2007) i.e. having at most a single hanging node for each edge) focus on three main aspects. First, the mesh module was designed for parallel execution, second its implementation tries to maximize cache hit-ratio and, finally, despite the constraints of the first two aspects, it tries to maintain flexibility.

We make several assumptions concerning the organization of the data structure. Every element in the mesh is assembled from several mesh entities. Elements can be refined (broken) and then de-refined (clustered) if needed. Information about refinements is stored in the data structure in an explicit way – divided elements are still kept in memory. Mesh is stored in a continuous manner in memory, despite it's tree-like topology shown in figure 1, where red elements are real elements in the mesh, and black are virtual parent elements.

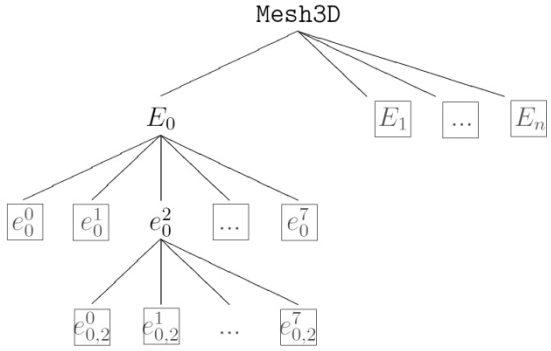


Fig 1. *Tree-like structure of elements.*

The storage scheme is a flatten tree. For elements from figure 1.1 the order is: $e^0{}_0$ $e^0{}_1$ $e^0{}_{0,2}$ $e^1{}_{0,2}$ ... $e^7{}_{0,2}$ ... $e^7{}_0$ E_1 ... E_n. Bounded elements are real elements in the mesh, and others are virtual parent elements. This allows for very fast derefinement process and gives information about element connectivity. When there is a need to iterate over a mesh, the natural order of elements is used – all divided elements are ignored as presented in figure 2.

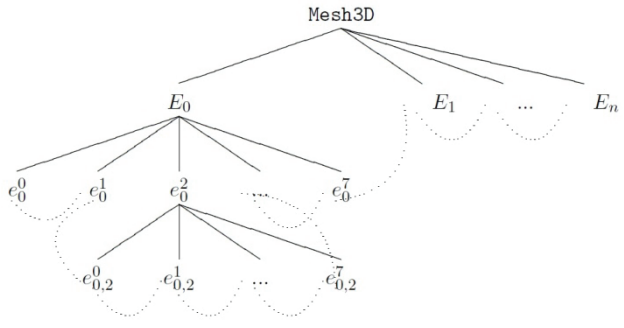


Fig. 2. *Natural order of elements. Dotted line marks transitions from one element to another.*

Natural order of elements is implemented as reloaded operators: next element – 'operator++' and previous element – 'operator--'. Next element is defined as the next leaf on the right hand side from the current leaf in tree-like hierarchy of elements. First element is the farmost left leaf and last element is the farmost right leaf in hierarchy.

Each type of mesh entity (vertex, edge, face and interior) is managed by its own memory manager, each one allowing for parallel memory operations on different entities in multi-threaded execution environment. Improvement of program execution speed in terms of memory was achieved by increasing memory request hit-ratio (making memory calls more local in terms of memory space). The basis of memory manager implementation is a simple continuous chunk of memory, filled with a sequence of objects of a given type. Using templates we replace cross references through memory (generated by standard technique of using pointers and allocating needed memory at run-time) by adjusting structures and classes before compilation. The implementation allows us to keep information in one place in memory, not only to keep in one place pointers to information. The main drawback of the solution is very slow and memory consuming process of adding or removing objects from the sequence, which requires reallocating all, or almost all objects in memory. We can avoid this by previously allocating much greater chunks of memory than actually needed, but this may lead to wasting memory resources. We notice that adding and removing objects is done only during adaptations of the mesh. Therefore, in our storage scheme, during first phase of refinement/derefinement process the objects are only marked to divide or remove, no one is created or deleted. Only when finalizing mesh adaptations, new memory requirements are computed, based on previously marked entities, and then single one memory reallocation is performed, if needed, which takes into account all objects' modifications. When this operation succeeds actual adaptation is performed.

Code was also designed with parallelism in mind. Two levels of parallelism were envisaged. First, global parallelism that works at the level of the whole finite element computational domain. At this level we focus on domain decomposition and load balancing problems. For small number of subdomains this level can be sufficient. However, with the increasing number of subdomains communication requirements grow and convergence properties of iterative solvers deteriorates. Hence the second level of parallelism is considered, with multicore processors and possibly hardware accelerators in mind.

At this level we concentrate on a single computational node described as a shared memory multithreaded execution environment. At the node level there is also possibility of handling more than one mesh simultaneously, e.g. when solving problems of fluid-structure interaction where there could be one mesh for fluid simulation and the other for structure. We can also use different approximation fields on the same geometrical mesh thanks to the distinction between the geometry and the solution degrees of freedom (managed by a separate approximation module).

To increase flexibility two main factors were taken into account. The first was the ability to interface with several different mesh sources and/or linear equations solvers. Mesh module was equipped with different input and output interfaces. The second factor is extensibility. It was achieved using template programming and meta programming, which combine program speed with advanced object oriented techniques. This allows one to easily extend the program with new types of mesh elements and other mesh entities.

1.3. Boundary layer with prismatic elements

As it was mentioned above, the program is able to handle hybrid, tetrahedral and prismatic, meshes. Prismatic elements are used close to solid boundaries, where boundary layers form. Prismatic elements allow for one-dimensional breaking, only in the direction normal to the boundary, to provide better resolution for high velocity gradients in boundary layers.

The layered, prismatic part of the mesh is combined with the tetrahedral part that fills the rest of the computational domain. The code handles hybrid meshes for standard calculations, as well as for problems with moving boundaries.

1.4. Moving parts of the mesh.

The algorithm for modifying and rebuilding the mesh during simulations with moving boundaries is presented using an example of a moving valve. It has the following stages.

1. First step: loading part of the mesh which have to be rebuilt into mesh generator.
   1.1. Designating part of the whole mesh which have to be rebuild.

1.2. Importing the mesh part into generator (vertices, boundary information, sub-domain information: fluid-structure boundary and moving structure information are imported).

2. Second step: building local mesh and defining sub-domains.
   2.1. Mesh building is done by using imported vertices and new vertices added to improve mesh quality. Delaunay method based algorithm is used.
   2.2. Based on the boundary vertices, sub-domains for valve, chamber, and external space (not used in simulation, but needed by re-meshing algorithm) are defined. (Example of domain partition is shown in figure 3).

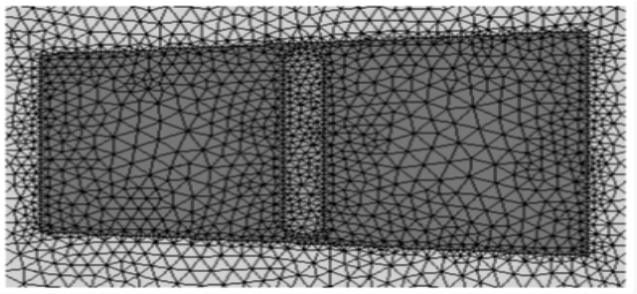


Fig. 3. *Domain partitioning during mesh rebuilding. Darkest sub-domain defines local chamber (fluid) sub-domain. Light gray in the middle refers to valve (structure), and lightest gray outside is for external space.*

3.3. Further mesh improvement – new vertices are added to achieve this goal. Element quality is determined by some chosen criteria e.g. minimal angle in triangle face etc.

4. Fourth step: adjusting solution values and putting back modified subdomain into computational domain.
   4.1. Interpolation of new degrees of freedom values in global mesh.
   4.2. Replacement of valve movement subdomain of global mesh by newly created local subdomain. Necessary renumeration of vertices and elements is done during this step.

It is important to note that during whole simulation the first and the second step will be executed only once. After the first iteration valve movement will be modelled only by the third and the fourth step. The saved r-adaptation sub-domain will be used. Thus after first iteration we obtain stable mesh interface between local and global mesh. Also computational module will receive elements meeting given quality criteria.

1.5. Test meshes

Figures 5 and 6 present several examples of meshes handled by the code. Meshes from figure. 1.5 are constructed for typical validation tests – backward facing step and von Karman vortex street. The mesh in figure. 1.6 is a mesh for the actual geometry of the artificial heart chamber.

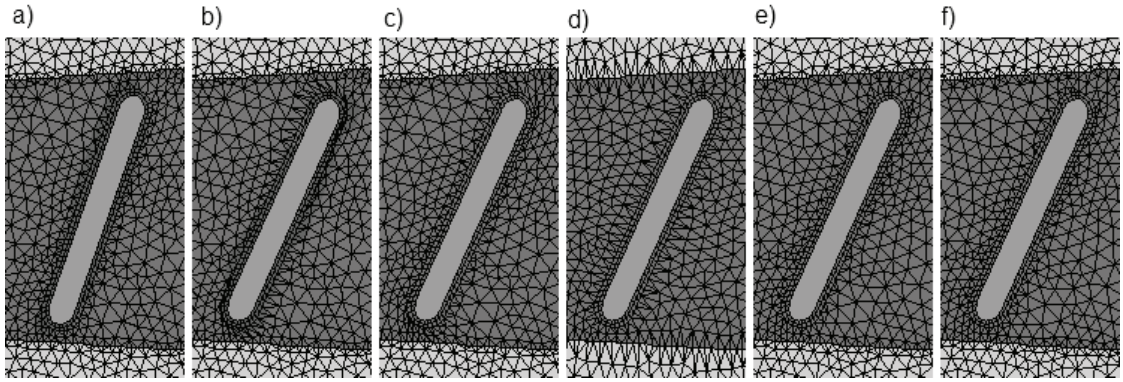


Fig. 4. *Six steps of re-meshing near moving valve with boundary layer. a.) initial state, b.) moved valve, c.) 1st mesh smoothing (Laplace), d.) equal volume based r-adaptation, e.) 2nd smoothing, f.) improving mesh quality according to given criteria.*

3. Third step: actual mesh rebuilding to move valve (see figure 4).
   3.1. Moving whole valve sub-domain into new position, given by the external valve movement algorithm
   3.2. Basic mesh improvement done by moving deformed mesh by r-adaptation algorithm. Vertices alignment is stored in an appropriate array.

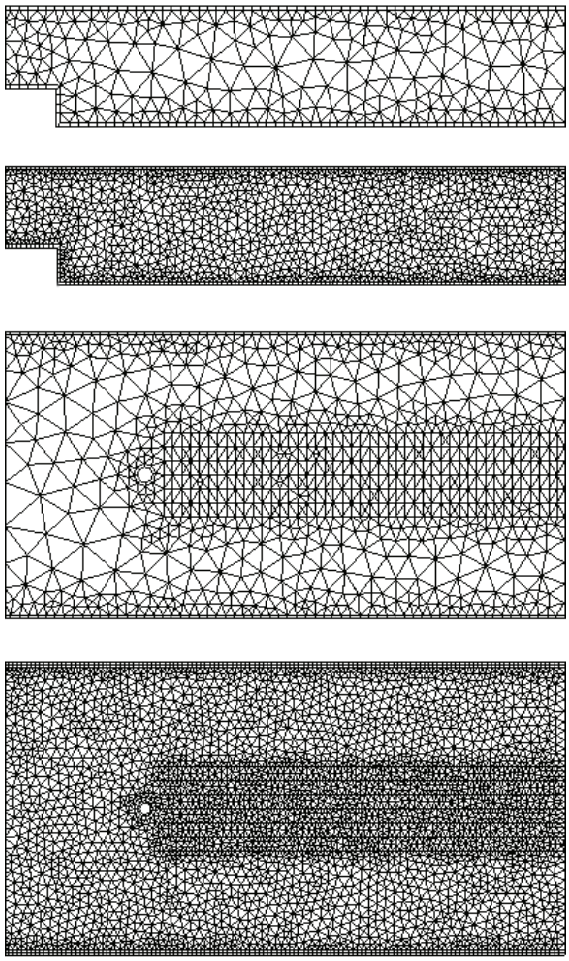


Fig. 5. *Examples of test meshes.*

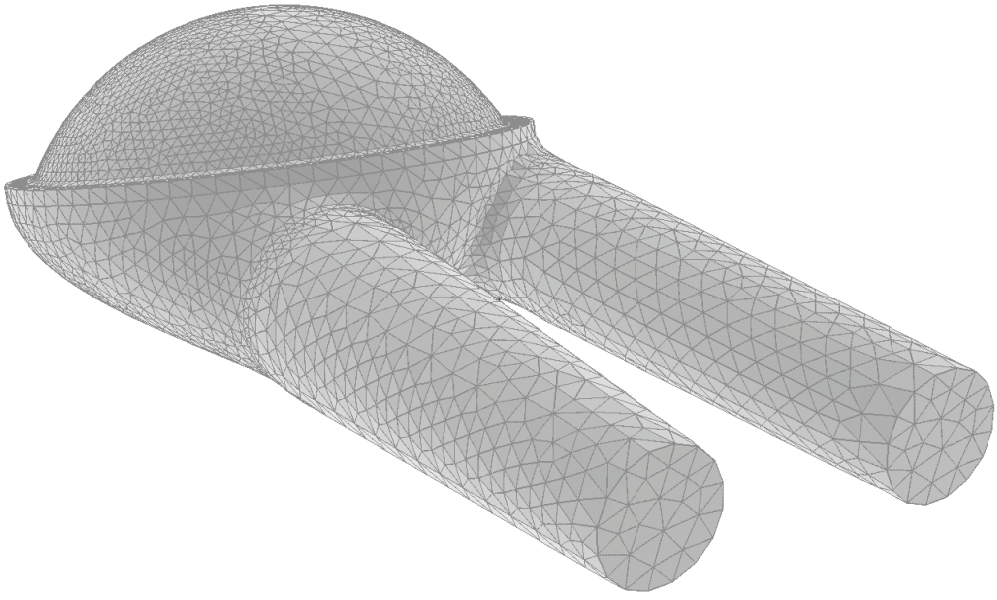


Fig. 6. *Example of "artificial heart chamber" model meshing.*

2. CONCLUSIONS

The paper described important steps undertaken to use adaptive hybrid meshes in a FEM code designed for 3D simulations of flow in artificial heart chambers. These include a new finite element formulation for incompressible flows in the domains with moving boundaries, an algorithm for handling meshes near moving parts of the domain. Example meshes handled by the code were presented, including meshes with moving valves and a mesh for the actual geometry of the artificial heart chamber.

BIBLIOGRAPHY

Banaś, K., 2004, A model for parallel adaptive finite element software, in: Kornhuber, R., Hoppe, R., P´eriaux, J., Pironneau, O., Widlund, O., Xu, J., (eds), Domain Decomposition Methods in Science and Engineering, *Lecture Notes in Computational Science and Engineering*, Springer, 40, 159–166.

Demkowicz, L., Oden, J., Rachowicz, W., Hardy, O., 1989, Towards a universal h-p adaptive finite element strategy, Part.1 Constrained approximation and data structure, *Computer Methods in Applied Mechanics and Engineering*, 77, 79–112.

Paszynski, M., Demkowicz, L., 2007, Parallel, fully automatic hp-adaptive 3d finite element package, *Engineering with Computers*, 23, 241.

Franca, L., Frey, S., 1992, Stabilized finite element methods. II. The incompressible Navier-Stokes equations, *Computer Methods in Applied Mechanics and Engineering*, 99, 209-213.

ACKNOWLEDGEMENTS

Financial assistance of the MNiSzW, project PSS no. 08/WK/P02/0001/SPB-PSS/2008, is acknowledged.

WYKORZYSTANIE ADAPTACYJNYCH SIATEK HYBRYDOWYCH W PROGRAMIE SYMULACJI PRZEPŁYWÓW W KOMORZE SZTUCZNEGO SERCA METODĄ ELEMENTÓW SKOŃCZONYCH

Streszczenie

Artykuł przedstawia zasady wykorzystania adaptacyjnych siatek hybrydowych w programie do trójwymiarowych obliczeń metodą elementów skończonych, zaprojektowanym do symulacji przepływu w sztucznych komorach serca. Szczególny nacisk został położony na uwzględnienie ruchomych części siatki występujących w sztucznym sercu, takich jak np.: zastawki. W celu poprawnego modelowania rozważanych zagadnień zostało opracowane specjalne sformułowanie słabe dla problemów z przemieszczającym się brzegiem, jak również algorytmy do poprawnego zarządzania przemieszczającą się siatką, w tym m.in. do remeshingu.

Received: December 3, 2010
Received in a revised form: December 29, 2010
Accepted: December 31, 2010