MECHANICS AND CONTROL
Vol. 32 No. 4 2013 http://dx.doi.org/10.7494/mech.2013.32.4.143

ANNA PERDUTA*

ENHANCING MESH ADAPTATION CAPABILITIES OF GetFEM++ "FEM ENGINE" WITH MAdLib LIBRARY

ABSTRACT

This paper describes enhancing mesh adaptation capabilities of GetFEM++ library. GetFEM++ is a versetile software package in C++ for implementing FEM based solvers for partial differential equations. Although GetFEM++ offers a wide range of mathematic tools for model descripion and discretization, it does only provide basic mesh refinement facilities. To enhance GetFEM++ capabilities, a mesh adaptation package MAdLib is used. The paper discusses basic requirements for mesh adaptation packages and presents MAdLib basic usage. The paper discusses implementation issues related to integration of both libraries. Examples illustrate capabilities of mesh adapation package and FEM framework extended by mesh adaptation facilities.

Keywords: *software, numerical analysis, finite element method, mesh adaptation packages*

MOŻLIWOŚCI ROZSZERZANIA BIBLIOTEKI GetFEM++ O PAKIETY ADAPTACYJNE

Artykuł przedstawia możliwości rozszerzania biblioteki GetFEM++ o obliczenia adaptacyjne. GetFEM++ jest biblioteką zaimplentowaną w C++ umożliwiającą rozwiązywanie cząstkowych równań różniczkowych w oparciu o metodę elementów skończonych. GetFEM++ oferuje szeroki zakres narzędzi matematycznych, które mogą być wykorzystane dla opisu modelu, jednak oferuje jedynie podstawowe możliwości w zakresie adaptacji siatki. Biblioteka MAdLib umożliwiająca przeprowadzanie procesu adaptacji siatki została wykorzystana w celu wzbogacenia możliwości biblioteki GetFEM++. W artykule zostały przedstawione podstawowe wymagania stawiane pakietom adaptacyjnym oraz zaprezentowane wybrane możliwości biblioteki MAdLib. Omówione zostały problemy związane z integracją obu narzędzi. Przykłady prezentują zastosowania pakietu MES rozszerzonego o obliczenia adaptacyjne.

Słowa kluczowe: *oprogramowanie, metoda elementów skończonych, pakiety adaptacyjne*

1. INTRODUCTION

Modern Finite Element Method (FEM) frameworks are required to be flexible enough to be applied to wide range of mechanical problems like plasticity, thermal conduction, fracture mechanics, coupled problems. As the computational capabilities of computing units get higher, the higher is the demand for more accurate results obtained from discretization methods used for solving partial differential equation systems. Therefore it is important that FEM frameworks provide routines for minimizing the errors arising due to the applied discretization method. This problem may be addressed by providing mesh adaptation procedures in FEM framework.

The paper presents the possibilities of extending GetFEM++ library with MAdLib – a mesh adaptation library. GetFEM++ is used as a primary FEM engine in FEMDK software – a FEM framework for analysis of coupled problems in structural mechanics (Putanowicz 2011).

The purpose of this paper is to put attention to implementation issues related to FEM framework programming. One of the aims is to share experience gained while implementing mesh adaptation facilities in FEM framework. The paper describes issues related to programming and C++ language use.

1.1. What is GetFEM++

GetFEM++ (GetFEM++ 2010) is an Open Source object oriented library, written in C++ with interfaces for Python and Matlab languages. It is a generic FEM library providing components for building FEM based simulations. The library supports different types of structured and unstructured meshes, offers a variety of interpolation methods and integration schemes. GetFEM++ provides routines for evaluation of fields of any rank, computation of fields' derivatives and basic norms. The library also delivers basic refinement procedures and basic error estimator. GetFEM++ may be used with different linear algebra solvers and is able to run in parallel environments.

1.2. What is MAdLib

MAdLib (MAdLib 2010) is an Open Source library implemented in C++. The library is designed as generic component for mesh adaptation tasks. Mesh adaptation delivered in the library is based on local mesh modifications. The library provides support for handling geometric model and mesh refinement for moving objects. MAdLib may be used as a standalone library or may be coupled with miscellaneous packages e.g. FEM library. Authors of MAdLib are also the authors of Gmsh mesh generator (Gmsh 2010), thus MAdLib is compatible with Gmsh data formats. MAdLib may be run in parallel mode, in this case Metis library is required for the mesh partitioning.

1.3. Mesh adaptation overview

The application of discretization methods for solving partial differential equations introduces approximation errors. It is desired to minimize these errors, so the obtained solution is more accurate. The mesh adaptation provides information about

* Institute for Computational Civil Engineering, Cracow University of Technology, Warszawska 24, 31-155 Cracow, Poland; A.Perduta@L5.pk.edu.pl

error and gives guidance for adjusting mesh to the specified problem, including choice of the mesh generation algorithm, element distribution in the domain, interpolation order. In finite element analysis mesh adaptation is one of the most time consuming tasks in the whole process of problem analysis. Therefore it is important to deliver efficient tools for mesh refinement purposes.

The task of mesh adaptation may be divided to two separate sub-problems: error estimation and mesh modification. Error estimation gives information about error quantity and its distribution in the domain. Starting from this one can determine where the mesh should be refined. Error estimation may be divided into two types of procedures: *a priori* and *a posteriori*. The *a priori* error estimation gives a general information on the convergence of the solution, it is based on assumptions made before the analysis process. The *a posteriori* error estimation gives information about error measure, it is computed on the basis of the approximate solution (Zienkiewicz *et al.* 2005, Ainsworth and Oden 2000).

Mesh modification provides procedures for modifying mesh to obtain the one that satisfies the specified mesh size requirements. The are two main classes of mesh modification: topological and geometrical. The main objective of topological modification is to adapt mesh element to specified requirements like mesh element size. In geometrical modification the goal is to generate a mesh that conforms geometry of represented model. The crucial issue is to ensure that elements from the resultant mesh will have quality good enough for analysis purposes. Mesh modification may be handled in two ways: one is to replace current mesh with the new one adapted to a specified size field; the other is to perform local mesh modification on mesh cavities to adjust mesh to a specified size field. Mesh cavities are regarded as regions obtained by removing some elements, that should (the regions) be filled with new, finer or coarser elements.

It is desirable that mesh adaptation package used for FEM analysis would provide separation between error estimation and mesh modification. Such separation of concerns gives more flexibility in the choice of different error estimators and provides a possibility of coupling different type of adaptation packages.

The paper focuses on the mesh modification problem, as it is the main feature delivered by MAdLib library and the main subject of the paper. It does not discuss the choice of appropriate error estimation technique, however due to the MAdLib capabilities of working with various types of error estimators, it is possible to investigate variety of estimators that are commonly used in FEM.

2. GetFEM++ NATIVE MESH ADAPTATION FACILITIES

As it was mentioned in the previous section, GetFEM++ library provides basic refinement procedures. Mesh refinement is based on a method described in Bank *et al.* (1983). This method of refinement may be applied to two dimensional meshes. In GetFEM++ refinement of triangular meshes is implemented.

For the selection of mesh elements that should be refined one may use basic error estimator delivered in GetFEM++. The error estimator is based on the jump of the normal derivative at element boundary. Because this type of element error indicator is suitable only for elements of the first order, care must be taken to approximate appropriate field with linear elements. If it is required one may implement more sophisticated error estimator based on basic estimator and norms delivered in GetFEM++.

It is worth to mention that GetFEM++ does not offer any support of geometric model, but it does support basic generation of structured meshes. If one wants to use more sophisticated model in analysis, the appropriate mesh must be delivered and plugged into GetFEM++ data structures using import functions. It is important to notice that in this case the input mesh will be the best approximation of the underlying geometrical model, so the generation of the mesh should be taken carefully.

3. MESH ADAPTATION PACKAGE AS AN INDEPENDENT COMPONENT

In this section general requirements for mesh adaptation package are presented.

3.1. Atomic mesh modification operations

One of the fundamental facilities of mesh adaptation library are procedures, that allow for elementary mesh modification. One can distinguish basic mesh modification operations: edge split, edge collapse, edge swap. The basic operations may be combined together to provide more specialized mesh transformation operators. One may divide these operators into two groups: the first is used to deliver prescribed mesh size field and the second to ensure mesh qua-lity. Ensuring the mesh quality is an important problem during the mesh refinement, the goal is to produce meshes suitable for analysis. There are lot of investigations on mesh quality assurance (Zavattieri *et al.* 1996) and there exist libraries especially designed for mesh optimization problems, for example Mesquite library (Mesquite 2009). Figures 1–4 illustrate the basic mesh modifications for 2D case.

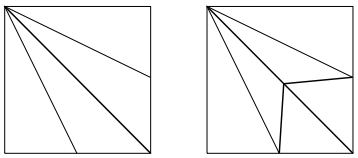


Fig. 1. Edge split – the thickened edge in the left figure is splitted into two parts

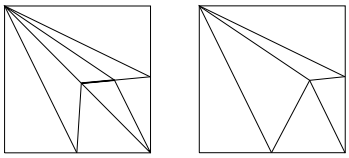


Fig. 2. Edge collapse – thickened edge in the left figure is removed from the mesh

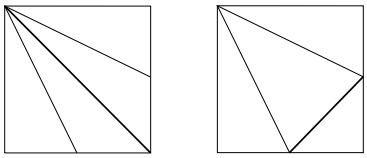


Fig. 3. Edge swap – remeshing of elements surrounding edge is done to improve the quality of the worst element

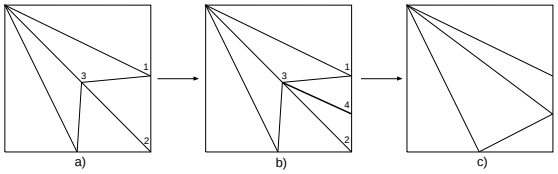


Fig. 4. Face collapse – the face with vertices (1), (2), (3) from the left figure is removed. This is a compound operator – first the edge split operator is used (b), than the newly created edge is collapsed (c)

3.2. Mesh density description

Mesh adaptation is based on a description of mesh density. This may be defined by specifying mesh size field, where the size field is understood as an edge length. For the problem one may construct a function $\delta(\mathbf{x},t)$ that defines an optimal size of edge at any given point $\mathbf{x}$ and any given time t. Based on the size field function, the non-dimensional length L_e of an edge e may be computed as follows:

$$L_e = \int_e \delta^{-1}(\mathbf{x},t)dl$$

This measure may be understood as the number of divisions that must be performed to obtain prescribed edge length. When the edge length L_e is equal to one, the edge has an optimal length. It is highly unlikely to obtain a mesh with all L_e quantities equal to one, thus some range of acceptable lengths should be defined. One may define acceptable L_e as $L_e \in [L_{low}, L_{up}]$. If the edge length exceeds defined range, the element is regarded as incorrect one. The definition of acceptable range of edge lengths is crucial to the mesh adaptation procedures, there are some assumptions on the edge range (Borouchaki *et al.* 1998), the authors of MAdLib library also have investigated this problem and presented their results in (Compère *et al.* 2008).

Various mathematical tools should be developed to deliver mesh size field description capabilities in mesh adaptation package. One of the issues is to provide support for defining isotropic and anisotropic sizes. In MAdLib both types of size fields are supported.

3.3. Handling of geometric model

Usually in finite element method initial analysis of problem is carried on coarse mesh and in the next steps the mesh is refined to better match the underlying geometric model. Therefore, it is important that adaptation package will provide support of geometric model. Geometric models are described as Boundary Representation (B-Rep) models (Hoffmann 1989), thus the support specifically for this type of representation is required in adaptation package.

3.4. Handling of fields on meshes

When dealing with finite element analysis, one interpolate different types of fields on the mesh. It is desired that mesh adaptation package will deliver structures that allow to store the additional data prescribed to the mesh entities. It is essential to provide proper data update during the mesh refinement procedure, ensuring that no data will be lost. One may use simple interpolation techniques to recompute data field to be defined on the new mesh, or may use specific type of mesh families like hierarchical meshes.

3.5. Support for moving objects

A moving object may be understood as an object moving in a fluid, a group of objects moving relative to each other or as an moving discontinuity (e.g. crack tip). As there are currently lot of problems devoted to CFD, support of adaptive mesh motion is required. The adaptation package that provides support for moving objects, should contain algorithms allowing for mesh refinement and node repositioning.

4. OVERVIEW OF MAdLib CAPABILITIES

As it was described in section 3, every mesh adaptation package must handle the prescription of mesh size. MAdLib provides special classes for size field description. In figure 5 a class hierarchy delivered in MAdLib is presented.

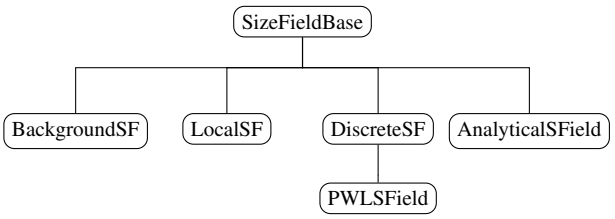


Fig. 5. MAdLib's size field class hierarchy

The choice of class type used for mesh adaptation is up to the user of the library. In Discrete Size Field, the size is prescribed at the vertices and linearly interpolated on edges. Using Analytical Size Field it is possible to describe a mesh size field given by a mathematical expression in the string form and parsed by mathematical parser (MAdLib uses Mathex (SSCILIB) for this purpose) or as a C++ function defined by user. If the user has a size field described in input mesh file, it is possible to use Background Mesh to read the file and perform adaptation. Local Size field allows for mesh refinement in area defined around geometric objects.

From the DiscreteSF a PWLSField class is derived. The PWLSField (piecewise linear size field) class delivers basic functionality for discrete size field description to perform an adaptation procedure. One needs to define a size (isotropic or anisotropic) for every mesh edge. The other possibility is to

use scaling functions available in the class. There is also the possibility to prepare a function that will be used to build the size field – a size needs to be computed and prescribed for every vertex of the mesh. That kind of facility is especially useful for building the size field based on error estimation. Below a simple example of MAdLib usage is presented.

Listing 1: Basic sample of MAdLib's usage

```
1 #include "MAdLib/MAdLib.h"
2
3 int main(int argc, char **argv) {
4
5   MAd::pGModel gmodel = NULL;
6   MAd::GM_create(&gmodel, "GeoModel");
7   MAd::GM_read(gmodel, "solid.geo");
8
9   MAd::pMesh mesh = MAd::M_new(gmodel);
10  MAd::M_load(mesh, "solid.msh");
11
12  MAd::PWLSField * sizeField = new MAd::PWLSField(mesh)
        ;
13  sizeField->setCurrentSize();
14  sizeField->scale(0.5);
15
16  MAd::MeshAdapter * adapter =
17       new MAd::MeshAdapter(mesh, sizeField);
18  adapter->run();
19
20  MAd::M_writeMsh(mesh,"myMesh.msh",2);
21
22  return 0;
23 }
```

First geometric model and mesh are read from files. Lines 12–18 contain the main part of adaptation procedure. At first an object representing the size field must be defined on the mesh. Here the piecewise linear size field is used. One needs to define size at each vertex of the mesh, in the example current size of mesh elements is take as the initial size field, which is then the scaled by a scaling factor equal to 0.5. It means that while refining each mesh edge is likely to be splitted into two subedges.

After the user prepares a size field object, he may create a `MeshAdapter` object, which is the main object responsible for performing the mesh adaptation procedure. In the adapter object one must input a mesh object and may define any number of size fields of various type. Here only one piecewise linear size field is prescribed. Finally the mesh adaptation procedure may be run. During the procedure execution an output containing information on performed mesh modifications is displayed. When the adaptation procedure is done one may write the resultant mesh to a MSH file.

In the presented listing 1 in order to refine the mesh a scaling factor was applied. It is also possible to perform mesh refinement based on the definition of edge length and other quantities defining the mesh quality. To define the acceptable edge length one may use functions that are delivered in `MeshAdapter` class. Several functions from the class are presented in code 2.

Listing 2: A snippet from AdaptInterface.h – available functions for setting parameter of mesh adaptation

```
    void setEdgeLenSqBounds(double lower,
                                    double upper);

    void setCollapseOnBoundary(bool accept=true,
                               double tolerance=1.e-6);

    void setSwapOnBoundary(bool accept=true,
                           double tolerance=1.e-6);

    void setNoSwapQuality(double noSwapQuality);
    void setSwapMinImproveRatio(double ratio);
    void setSliverQuality(double sliverQuality);
```

In `MeshAdapter` class various statistic functions are delivered, so the user may observe the adaptation process and the resultant quality of the mesh. One may print detailed information about sliver elements handled during the adaptation procedure or write basic informations about mesh quality, e.g. mean ratio, to a Gmsh POS file. The following snippet of code presents the usage of output functions defined in `MeshAdapter` class.

Listing 3: Saving additional output from adapter object

```
1 std::ofstream file;
2 file.open("statistics.txt");
3 adapter->printStatistics(file);
4 adapter->printSliverRegionStatistics(file);
5 file.close();
6
7 adapter->writePos("postpro.pos", MAd::OD_SIZEFIELD_MEAN
      );
8 adapter->writeMsh("output.msh");
```

It is possible to perform mesh adaptation on different levels of generality. At the highest level one may choose the run method to perform the adaptation, like in examples given above. There is also the possibility of performing a general operations that have influence on the whole mesh structure, for example splitting the longest edges of mesh. At the lower level one may execute a group of basic operators in a loop e.g. edge swap loop. The lowest level contains primary operators on elementary entities, that is: one can perform edge collapse on a single edge of a mesh. That kind of separation gives the user more control of the adaptation process.

The basic format on which MAdLib operates is Gmsh MSH file containing a mesh description. MAdLib provides procedures for handling geometric model, but an additional library is required to provide this facility – MAdLib delivers interface to Gmsh library for geometry support. By providing support of the geometry, it is possible to adjust the mesh to the geometric curvatures. MAdLib delivers procedures that may be used to define density of the elements near the curved boundaries.

If one does not have a geometric model, initial mesh may be used instead. An important issue is that the initial mesh will also give the best approximation of the geometry and no geometric curvature adjustment may be performed.

5. IMPLEMENTING GetFEM++ INTERFACE TO MAdLib

As it was shown in the previous section, MAdLib may be used as a standalone library. In this section coupling of MAdLib with GetFEM++ library will be discussed.

The main problem while coupling both libraries is the difference in the data structure. It is crucial to ensure that no topological and geometric information will be lost while exchanging data between these libraries. Also assignment of

mesh elements to specified geometric entities should be preserved.

In GetFEM++ the mesh object may contain mesh regions to which different data may be prescribed. Data connection with the mesh is described in a special object designed for modeling the FEM problem. On contrary, MAdLib requires that all additional data will be prescribed to the mesh entities. Accordingly MAdLib delivers special data structures for storing additional data. The following procedure was implemented to enable mesh transfer between GetFEM++ and MAdLib library:

Listing 4: Mesh import procedure outline

```
* read the number of vertices in GetFEM++ mesh

* iterate through all vertices in GetFEM++ mesh
   and add equivalent entities to MAdLib mesh

* iterate through regions defined in GetFEM++ mesh
   – add to the MAdLib mesh element assigned
     to the GetFEM++ region and attach information
     about geometric entity to which element is assigned

* iterate through the remaining elements of GetFEM++ mesh
   – add elements to MAdLib mesh
```

Similar procedure is implemented for importing mesh from MAdLib to GetFEM++ structures.

The important step is to provide a function for mesh size field evaluation. The goal of integration of both packages is to combine GetFEM++ error estimator with MAdLib's size field prescription. A piecewise linear size field was built for the analysis purposes. Because MAdLib provides iterator objects that allow to iterate through all mesh entities, it is possible to prescribe the size to every mesh vertex.

Having implemented import and mesh size field functions, one may use both libraries to perform mesh adaptation procedure. In this case it is not necessary to design a function that imports solution from GetFEM++ to MAdLib because direct access to the solution is not needed for MAdLib to perform mesh adaptation. The size field is built on error estimation which is derived from the solution, but is computed in GetFEM++ and assigned to corresponding elements from MAdLib's mesh.

Nevertheless transfer of the additional data should be discussed for the completeness of this section. The main problem with attaching data to the mesh does not lie in the implementation of transfer procedure from given solver to MAdLib's structure – the main issue is proper data update. It is expected that when the refinement procedure is done, data attached to the mesh will be kept up to date. In order to ensure correct data update MAdLib delivers a callback function mechanism. A callback function is a function that is called after every performed operation, in the case of MAdLib that is after every mesh modification. Such callback function may be used to provide different schemes of interpolation of the data attached to the mesh. It is the user responsibility to provide adequate callback function to his code. More information on MAdLib's callback function mechanism are presented in (Compère *et al.* 2010). Figure 6 presents basic workflow of mesh adaptation process for solving problems with GetFEM++ library extended with MAdLib library.

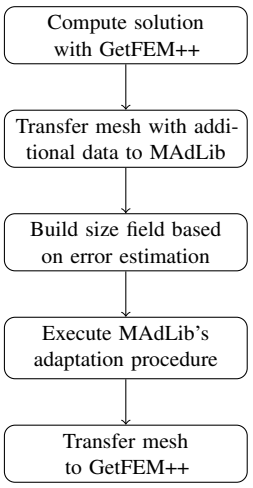


Fig. 6. Adaptation procedure used for solving FEM problems with GetFEM++ and MAdLib libraries

6. ENHANCING SUPPORT FOR MOVING OBJECTS

Authors of MAdLib library have provided a support for mesh refinement in moving domain. This is a difficult task to program as it requires support of r-refinement, the type of refinement where the nodes of the mesh are repositioned but no new entities are added. This type of refinement may provide poor quality of the resultant mesh.

In MAdLib node repositioning is based on elasticity analogy. For more information reader may view (Compère *et al.* 2010). When the object is moved the mesh around an object is also refined. For handling mesh motion MAdLib delivers a class called `MobileObject`. Using this class one may specify geometric entities that are part of geometric object subjected to movement. To describe a motion one may define a displacement vector or velocity.

MAdLib offers a special size field that may be used with moving objects to refine a mesh in a specified radius. The size field is `LocalSF` to which one define a group of geometric entities that constitute a moving object. One may define isotropic or anisotropic mesh size and define a radius describing the area of mesh refinement.

As the problem is time dependent MAdLib offers time support in `MeshAdapter` class. In listing 5 a basic example of moving geometric objects is presented.

Listing 5: Sample of usage of MAdLib's MobileObject class

```
1 #include "MAdLib/MAdLib.h"
2
3 #include <vector>
4
5 int main(int argc, char* argv[]){
6
7   MAdLibInitialize(&argc, &argv);
8
9   MAd::pGModel gmodel = NULL;
10  MAd::GM_create(&gmodel, "FirstModel");
```

```
11   GM_read(gmodel, "surf.geo");
12
13   MAd::pMesh mesh = MAd::M_new(gmodel);
14   M_load(mesh, "surf.msh");
15
16   MAd::mobileObject *mob = new MAd::mobileObject(mesh);
17
18   mob->addGEntity(1,5);
19   mob->addGEntity(1,6);
20   mob->addGEntity(1,7);
21   mob->addGEntity(1,8);
22
23   std::vector<std::string> kinematics;
24   kinematics.push_back("-1");
25   kinematics.push_back("-2");
26   kinematics.push_back("0.0");
27   mob->setDxKinematics(kinematics);
28
29   MAd::LocalSizeField *localSF =
30     new MAd::LocalSizeField(mesh);
31   localSF->setIsoSize(0.05, "0.01");
32   localSF->addGeometricEntity(1,5);
33   localSF->addGeometricEntity(1,6);
34   localSF->addGeometricEntity(1,7);
35   localSF->addGeometricEntity(1,8);
36
37   localSF->updateTree();
38
39   mob->addLocalSField(localSF);
40
41   MAd::MeshAdapter *adapter =
42     new MAd::MeshAdapter(mesh);
43
44   adapter->storeInitialCoordinates();
45
46   MAd::mobileObjectSet *mobSet =
47     new MAd::mobileObjectSet();
48   mobSet->insert(mob);
49   adapter->registerObjects(mobSet);
50
51   MAd::PWLSField *pwlSF = new MAd::PWLSField(mesh);
52   pwlSF->setCurrentSize();
53   pwlSF->scale(0.5);
54
55   adapter->addSizeField(pwlSF);
56
57   double t = 0.0;
58   double dt = 1.0;
59   adapter->setTime(t);
60   adapter->moveObjectsAndReposition(t, dt);
61   adapter->run();
62
63   MAd::M_writeMsh(mesh, "output.msh",2);
64   delete adapter;
65   MAd::M_delete(mesh);
66   MAd::GM_delete(gmodel);
67
68   MAdLibFinalize();
69
70   return 0;
71 }
```

First one needs to define mobile object with geometric entities (lines 16–21). One should define how the object will be moved, in this case a displacement vector is prescribed (lines 23–27). The user may use `x,y,z` variables which represent the initial coordinates, and `t` which represents the time. To every mobile object a local size field must be prescribed. If user defines a local size field (lines 29–37) he must specify at which geometric entities the size field is defined (line 39). After the mobile object definition, user must create adapter object, to which mobile object must be plugged. First a `MobileObjectSet` is defined and then one mobile object is added to the set (lines 46–48). As one may see the user may define number of different mobile objects. Then the mobile set is added to the adapter (line 49). To move an object `moveObjectsAndReposition` function from `MeshAdapterClass` is called. This function moves the boundaries and performs node repositioning. It is also possible to use `partlyMoveObjects` function that only moves the boundaries without node repositioning. Because mesh motion introduces reshaping of mesh elements, one may use smoothing functions to smooth the mesh. In MAdLib, basic Laplace smoothing technique is implemented.

It is also worth to mention that MAdLib offers access to direct data structures of mesh entities, so it is possible to directly change the position of specific mesh entities. This may be another way of handling a mesh motion and this feature was used to generate presented examples described in section 7.4. However these are lower API functions, and one should use two aforementioned functions defined in `MeshAdapter` class. The higher level functions are more general, they give control of mesh motion process and provide mesh quality assurance, while the lower level functions are designed to perform specific task i.e. change node's position.

7. EXAMPLES

7.1. Geometric model

As the first example MAdLib's capabilities of handling the geometric model are presented. The boundary of geometric model is represented by a straight line and B-Spline curve. In figure 7 geometric model and input mesh are presented. As one may see initial mesh does not match well the underlying geometric model. In figure 8 there are presented meshes after the refinement procedure, a simple scaling factor was introduced to obtain the resultant meshes. One may see that the output meshes better match underlying geometric models. Introducing a scaling factor and prescribing the mesh density on the curvature one may obtain a refined mesh as presented in figure 8.

It is worth to mention that the discussed example is adequate when one is given an initial coarsed mesh and the adaptation procedure is used to refine the mesh. In finite element analysis one first uses appropriate mesh generator to produce a mesh that better matches the geometric model. The goal of this example is to present that one may use MAdLib's capabilities of handling the geometric model in FEM analysis.

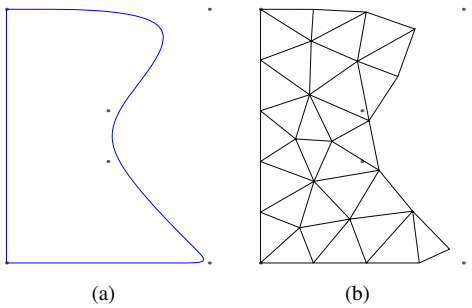
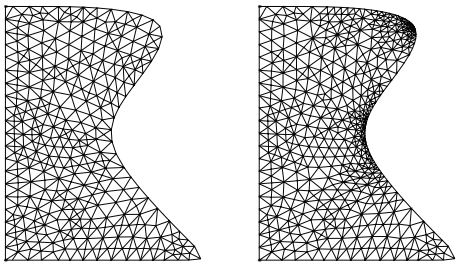


Fig. 7. Geometric model (a) and input mesh (b)



(a) (b)

Fig. 8. Output meshes: (a) – after simple refinement, (b) – after refinement with prescribed density at the curvature

7.2. Solution of Poisson's equation

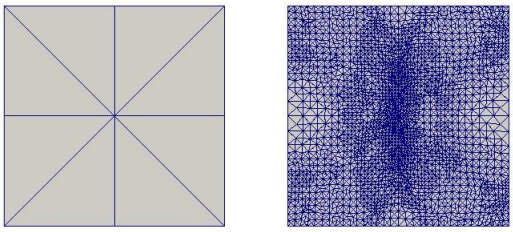
As an example of usage of GetFEM++ library extended with MAdLib, the results obtained from adaptive solution of Poisson problem are presented.

$$\Delta u = f$$

The source function:

$$f(x) = 150e^{-100((x-0.5)^2+(y-0.5)^4)}$$

is defined in the square domain ($[0;1]\times[0;1]$); Homogenous Dirichlet boundary conditions are imposed on the boundary. Initial coarsed mesh is presented in figure 9.



(a) (b)

Fig. 9. Initial mesh (a), refined mesh (b)

After adaptation procedure an output mesh is generated, as it may be seen in figure 9. The solution and error distribution are presented in figures 10 and 11. One may observe that the higher the gradient of the solution is, the higher the density of mesh is in the region.

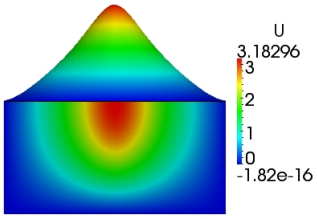


Fig. 10. Solution of Poisson's equation

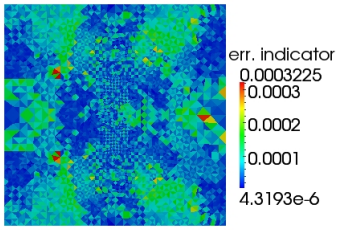


Fig. 11. Error distribution obtained for refined mesh

7.3. Shear plate cantilever

A geometric model and initial mesh for the cantilever are presented in figure 12. The curved edge is represented by an arc with radius $r = 0.25$ and angle $\alpha = \frac{\pi}{2}$. The problem description is given below:

$$E = 2\cdot 10^8\,[\mathrm{kPa}]$$

$$\nu = 0.25$$

$$h = 0.01\,[\mathrm{m}]$$

$$P = 400\,[\mathrm{kN}] - \text{net force}$$

The right edge of the cantilever is uniformly loaded with net force P.

Due to the GetFEM++ standard element error indicator based on normal derivative, the equivalent stress field, on which the indicator was computed, was approximated with linear triangular elements.

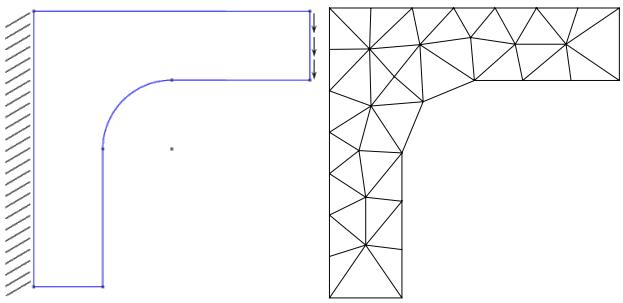


Fig. 12. Geometric model and input mesh

One may observe that the initial mesh does not reflect the curvature introduced in the geometric model. While performing the mesh adaptation procedure, it is expected that mesh will be denser in the region of maximum stress concentration and also that the output mesh will be more adjusted to the underlying geometric model, as one might seen in the example 7.1. It should be emphasized that no special mesh size field was applied in the curved region. The resultant meshes obtained for four steps of the analysis are presented in figure 13. Results obtained for initial and final mesh are presented in figure 14.

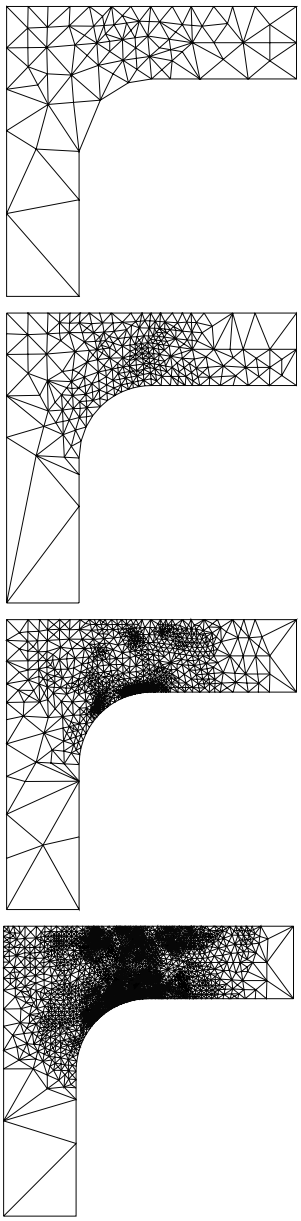


Fig. 13. Refined mesh after each ataptation step

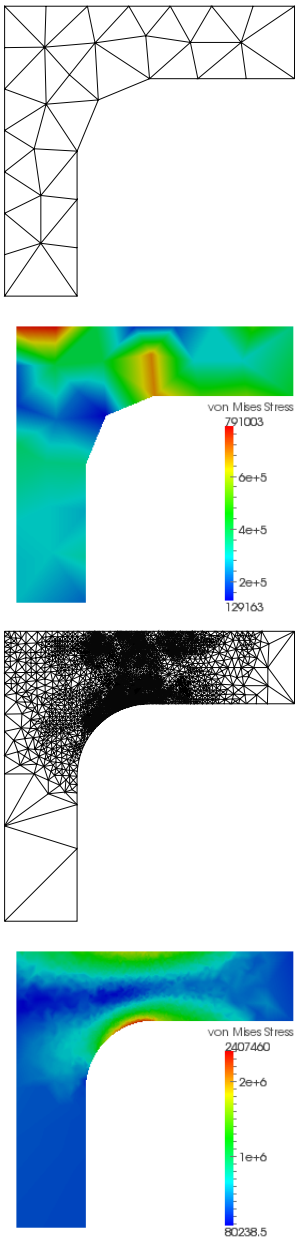


Fig. 14. Results obtained for initial coarsed mesh and final mesh

7.4. Moving objects

As the last example a mesh adaptation for moving objects will be presented. In the section 6 the general concept of handling mesh motion was introduced. It was mentioned that MAdLib is capable of supporting motion of mesh boundaries including internal ones. Here a different case of moving object is considered. Instead of moving mesh boundaries (that is geometric), we move an abstract point which indicates the center of mesh refinement zone. A general image representing this situation is presented in figure 15.

Two examples will be given, one in 2D and the other in 3D. For both examples the procedure for dealing with moving points is analogous and is explained here. A NURBS curve (modeled using openNURBS SDK (openNURBS 2012)) is defined along which the point is moved. A mesh size field is prescribed in the specified radius from the point. The obtained results are presented in figures 16 and 17 appropriately.

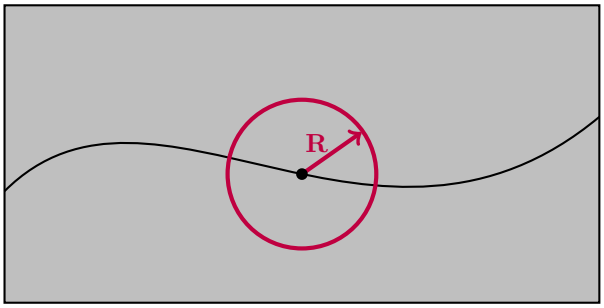


Fig. 15. Geometric point moving along defined curve. In the radius from the point a mesh size is defined

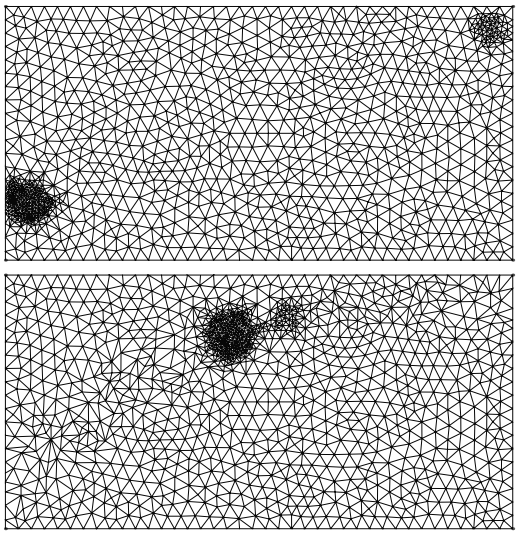


Fig. 16. Mesh refinement around a moving points in 2D domain

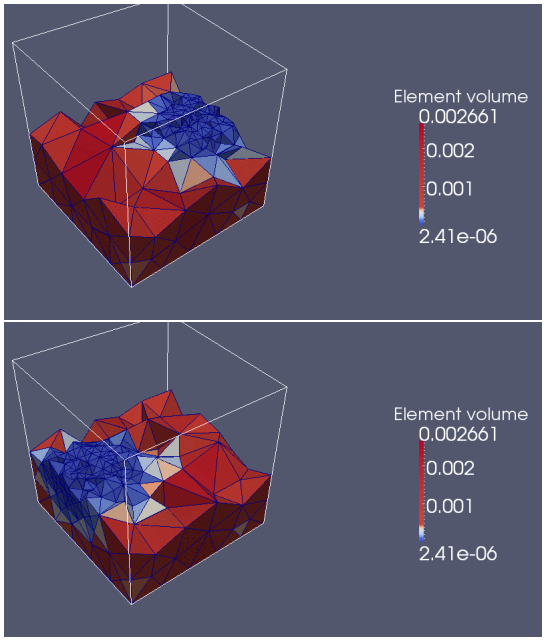


Fig. 17. Mesh refinement around a moving point in 3D domain

8. CONCLUSIONS

An overview of extending finite element library with adaptation package was presented. Basic code examples of MAdLib usage were given, introducing the library fundamental capabilities. Coupling GetFEM++ with MAdLib enhanced the capabilities of handling geometric model in adaptation procedure and introduced support for mesh motion. Presented examples showed the basic capabilities of coupled libraries.

As it was previously mentioned it is possible to use various types of error estimators, therefore investigation of other types of error estimation used with GetFEM++ and MAdLib would be an interesting problem to carry.

Increasing development of parallel platforms makes necessary to implement computational frameworks using parallelism to improve their computational speed. It is worth noting that both libraries may be used in parallel environments (GetFEM++ 2010; Sheel and Remacle 2010), what is a strong advantage of these packages. However only sequential cases were investigated and presented in this paper.

Acknowledgements

Scientific research has been carried out as a part of the project "Innovative recourses and effective methods of safety improvement and durability of buildings and transport infrastructure in the sustainable development" financed by the European Union from the European Fund of Regional Development based on the Operational Program of the Innovative Economy.

References

Ainsworth M., Oden J.T. 2000, *A Posteriori Error Estimation in Finite Element Analysis*. Wiley-Blackwell.

Bank R., Sherman A., and Weiser A. 1983, *Some Refinement Algorithms And Data Structures For Regular Local Mesh Refinement*. Scientific Computing IMACS.

Borouchaki H., Hecht F., Frey P.J. 1998, *Mesh Gradation Control*. International Journal for Numerical Methods in Engineering.

Compère G., Remacle J.-F., Jansson J., Hoffman J. 2010, *A mesh adaptation framework for dealing with large deforming meshes*, International Journal for Numerical Methods in Engineering, 82(7):843–867.

Compère G. and Remacle J.-F., Marchandise E. 2008, *Transient Mesh Adaptivity with Large Rigid-Body Displacements*. Springer, IMR, pp. 213–230.

GetFEM++ Homepage 2010, `http://download.gna.org/getfem/html/homepage/index.html`.

Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities 2010, `http://geuz.org/gmsh/`.

Hoffmann C.M. 1989, *Geometric and Solid Modeling*. Morgan Kaufmann Pub, San Mateo, California.

MAdLib: an open source Mesh Adaptation Library 2010, `http://sites.uclouvain.be/madlib/`.

SSCILIB (Small Scientific Library), `http://sscilib.sourceforge.net/`.

Mesh Quality Improvement Toolkit 2009, `http://trilinos.sandia.gov/packages/mesquite/`.

openNURBS SDK 2012, `http://www.rhino3d.com/opennurbs`.

Putanowicz R. 2011, *Grounds for the selection of software components for building FEM simulation systems for coupled problems*, Mechanics and Control, 30, pp. 234–244.

Sheel T.K., Remacle J.-F. 2010, *Madlib – mesh adaptation library: An efficient parallel mesh adaptation algorithm.*. LAP LAMBERT Academic Publishing.

Zavattieri P.D., Dari E.A., Buscaglia G.C. 1996, *Optimization Strategies In Unstructured Mesh Generation*. Int. J. Numer. Meth. Engng..

Zienkiewicz O.C., Taylor R.L., Zhu J.Z. 2005, *The Finite Element Method: Its Basis and Fundamentals*. ButterHeinem ST, 6th Revised Edition.