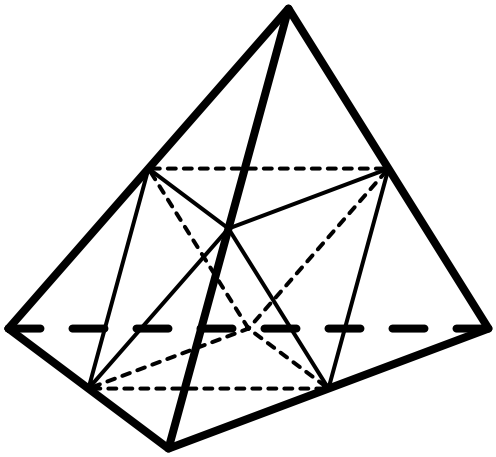
TECHNISCHE UNIVERSITÄT WIEN

IATP/tuwien.at

AGM^{3D} Extensions Manual



Research Group J. Fidler
Institut für Angewandte und Technische Physik
Vienna University of Technology

Werner Scholz
9426502

October 1998

Contents

1 Introduction

The ADAPTIVE GRID MANAGER by Jürgen Bey provides a set of problem independent tools and a complete working environment for the adaptive numerical solution of partial differential equations in three space dimensions.

AGM^{3D} creates the finite element meshes and manages all data necessary for the calculations. Its main purpose is the refinement of the mesh according to the refinement algorithm described in [Bey95]. Consequently only problem dependent parts such as the mathematical model or error estimators have to be implemented. The manual of AGM^{3D} [Bey94] describes all problem independent features in detail. Hence, this manual describes only extensions and new features.

2 User manual

2.1 Compiling AGM^{3D}

Before compiling AGM^{3D} it is necessary to modify the `makefile`. The variable `APPL` gives the name of the application. Currently the convection diffusion (`CD`) and the magnetization application (`MAG`) are available. `BIN` stands for the target directory for the executable and `EXE` for the name of the executable. `b` or `x` is prepended for the batch and X window version, respectively, and the name of the application is appended (e.g. `xagm5MAG`, `bagm5MAG`, `xagm5CD`). If the symbol `EXTEND` is defined using the variable for compiler options `COPTS`, the application independent extensions, which are described in section 2.6, will be compiled and linked. More details on variables and X window libraries can be found in chapter 1.3.2 of the AGM^{3D} manual [Bey94].

The X window version of AGM^{3D} with graphical user interface is compiled with `make xagm` or simply `make`. The function `main` is expected in `mainx.c` in the subdirectory of the selected application.

The batch version version is compiled with `make bagm`. The function `main` is expected in `mainb.c` in the subdirectory of the selected application.

2.2 Running AGM^{3D}

The X window version

The graphical user interface of AGM^{3D} is described in section 1.4 of the AGM^{3D} manual [Bey94]. It has not been modified.

The batch version

Currently, only for the magnetization application a batch version is available.

AGM^{3D} searches for three script files in the current working directory. First, `pre.agm` is executed, then `fem.agm` and finally `post.agm`. The syntax of all standard commands can be found in the AGM^{3D} manual [Bey94] and that of all extended and modified commands in section 2.6 of this manual.

First, the preprocessing commands specified in `pre.agm` (e.g. open a new document, define the FE mesh) are executed. The second script `fem.agm` could do the refinement and coarsening and at the end call itself using the commmand `execute fem.agm`. AGM^{3D} exits this recursive call as soon as the internal `BreakFlag` is set `true`. The `vemcall` command uses this mechanism. Finally, the postprocessing as defined by `post.agm` is done. If one or more script files are missing, AGM^{3D} tries to continue with the next.

As a result, AGM^{3D} stays alive from the beginning until the end of the calculations and mesh manipulation. During idle time, the operating system (e.g. Unix) should set AGM^{3D} in sleeping mode and move it to the swap space on hard disk to save memory. Thus, the current finite element mesh is always available and both refinement and coarsening of the mesh are possible. If the mesh was exported, AGM^{3D} terminated, restarted later on and the new mesh imported, it would not be possible to coarsen the mesh any more, because the multigrid structure has been lost.

Another possiblity is saving the error files, which contain information about the elements to be refined. These files can be reloaded after restarting AGM^{3D}. If the same values for refinement and coarsening tolerance are used after each restart, the complete mesh can be rebuilt. A set of sample scripts is given in figures 1 and 2.

```
open test_document cube magnetization multigrid
impbvertex sim.bve
impivertex sim.ive
impelements sim.ele
```

Figure 1: Sample `pre.agm` script

2.3 The new magnetization problem

This problem has no solving capabilities. It just provides a “clean” environment without the original convection diffusion problem supplied by Bey.

```
set refinetol = 0.1
set coarsentol = 0.01
imperror sim.err.1
estimate
refine
expgeom sim.off.1

set refinetol = 0.05
set coarsentol = 0.002
imperror sim.err.2
estimate
refine
expgeom sim.off.2

expvertices sim.nod.2
expelements sim.ele.2
```

Figure 2: Sample `fem.agm` script

2.4 Domains

There are six different domains available for the magnetization problem (cf. figure 3). In addition it is possible to manipulate domains of any shape, if a tetrahedral finite element mesh is available. As their boundary is covered by triangles this general type of domain is called `triabnd`. It is even possible to refine and coarsen elements of disconnected meshes. This is useful if a finite element mesh of two distinct objects, which have no element and no vertex in common, is used. However, the projection of boundary midnodes to curved surfaces is not possible, since this would require an analytical description.

Cube

The cube is one of the standard domains provided by AGM^{3D}. A detailed description can be found in [Bey94] in section 2.4.4 on page 28.

Hexahedron

The hexahedron has a quadratic base and an aspect ratio (height:base) of 2:1. It has been implemented for testing purposes. In order to manipulate

the mesh of a general hexahedron it is advisable to project the vertices of a hexahedron onto a unit cube and use that domain within AGM^{3D}.

Pentahedron

The pentahedron was created to test the algorithms of AGM^{3D} for compatibility with segments (surface planes) of triangular shape.

Sphere

The sphere is another standard domain provided by AGM^{3D}. Its description can be found in [Bey94] in section 2.4.4 on page 28.

Pipe

The s-shaped pipe is another standard domain provided by AGM^{3D}. It has been introduced in version 1.2 of AGM^{3D} for which no updated documentation is available, yet.

Tube

The cylindrical tube is another standard domain provided by AGM^{3D}. It has been introduced in version 1.2 of AGM^{3D} for which no updated documentation is available, yet.

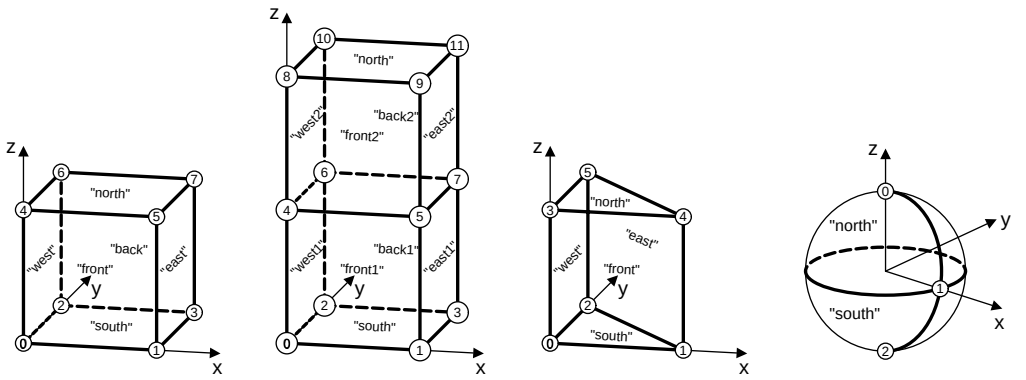
Arbitrary shape

The magnetization problem also provides commands to import domains of any shape and manipulate them. The domain has to be defined in a GeomView OFF file which consists of the boundary nodes and the boundary triangles of the tetrahedral mesh. The inner vertices can be imported using `impivertex` and the elements using `impelements`. There is also a converter (`util/out2off`) for neutral files (as exported by Patran), which generates the GeomView file, a file of the inner vertices and a file of the elements. See the description of `impdomain` in section 2.6 for further details.

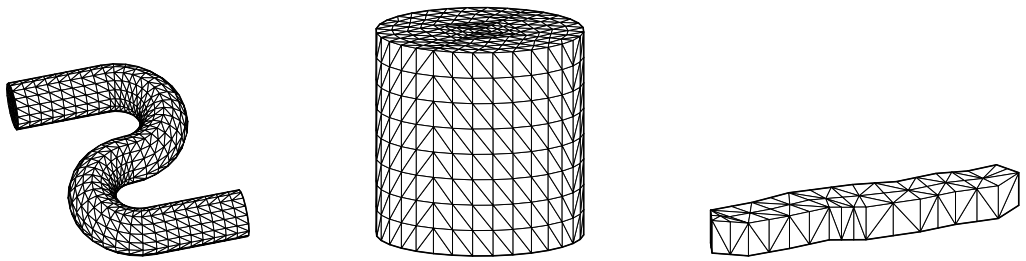
2.5 New variables

estimator [etamax]

The current error estimator used by the `estimate` command. Default is the etamax estimator (cf. section 2.7).



(a) Cube (b) Hexahedron (c) Pentahedron (d) Sphere



(e) Pipe (f) Tube (g) Arbitrary domain

Figure 3: Available domains

refinetol [0.5]

A parameter to select elements for refinement. It is used by several estimators. Its exact meaning depends on the estimator defined by the variable `estimator` (cf. section 2.7).

coarsentol [0.1]

A parameter to select elements for coarsening. It is used by several estimators. Its exact meaning depends on the estimator defined by the variable `estimator` (cf. section 2.7).

2.6 New commands

Application independent commands

These commands are implemented in `extend/extend.c`.

test

This is a dummy command which can be used for anything. Only the function `TestCommand` in `extend.c` has to be adapted to one's needs.

```
Function : TestCommand
Purpose  : Dummy command for testing purposes
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : test
```

impbvertex filename

Create new boundary vertices. This command expects a file in which each line starts with the vertex id. The following columns are expected to be valid arguments for the `bvertex` command. The vertex ids are checked against the ids assigned to the new vertices by AGM^{3D}. The vertices are numbered consecutively starting with id 0.

```
Function : ImpBvertexCommand
Purpose  : Import the real world coordinates of all boundary vertices
           from a file (.bve) with the following structure:
           id {segnum u v}+
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : impbvertex  filename
```

impivertex filename

Create new inner vertices. The first column gives the vertex id. The second, third, and fourth column of the file give the x-, y-, and z-coordinates of the new vertices respectively. The ids given in the file are checked against the ids assigned to the new vertices by AGM^{3D}. Usually, this command is called after initializing a new problem with a certain domain and possibly creating new boundary vertices with `impbvertex`. Consequently, the highest id is given by the number of boundary vertices of the domain (eight in the case of cube and hexahedron, six for a pentahedron, and three for a sphere; cf. fig. 3) plus the number of additional boundary vertices minus one (the lowest id is zero). If the domain is imported by `impdomain`, the number of boundary vertices depends on the domain, of course.

```
Function : ImpIvertexCommand
Purpose  : Import the real world coordinates of all inner vertices
           from a file (.ive) with the following structure:
           id x-coordinate y-coordinate z-coordinate
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : impivertex  filename
```

impelements filename

Create new elements. The first, second, third, and fourth column of the file give the ids of the four vertices of each element (cf. `element` command).

```
Function : ImpElementsCommand
Purpose  : Import the ids of the four vertices of all elements
           which form the finest triangulation
           i.e. those which were not refined any further and therefore
           have no sons to a file (.ele) with the following structure
           id_of_first second third fourth_vertex
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : impelements filename
```

expvertices filename

Export the real world coordinates of all vertices to a file. In each line of the file there are the id, x, y and z-coordinates of a vertex with a precision of 9 digits. The file is created or overwritten as in all other commands for exporting data.

```
Function : ExpVerticesCommand
```

```
Purpose  : Export the real world coordinates of all vertices
           to a file (.knt) with the following structure:
           id x-coordinate y-coordinate z-coordinate
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expvertices filename
```

expelements filename

Export the identifiers of the four vertices of each element contained in the finest triangulation. The file is created or overwritten. In each line of the file there are the id of first, second, third, fourth vertex. The order of output lines represents the order of elements in the list they are stored in.

```
Function : ExpElementsCommand
Purpose  : Export the ids of the four vertices of all elements
           which form the finest triangulation
           i.e. those which were not refined any further and therefore
           have no sons to a file (.ijk) with the following structure
           id_of_first second third fourth_vertex
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expelements filename         cf. ExpVerticesCommand
```

expboundary filename

Export the vertices and faces of the boundary. The output file contains information about the vertices and faces on the boundary in the following format: id of first, second, third vertex of a boundary face.

```
Function : ExpBoundaryCommand
Purpose  : Export the ids of the vertices of all boundary faces to a
           file (.bnd) of the following structure
           id_of_first second third_vertex_of_boundary_face
           ...      number of lines given by number_of_sides
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expboundary  filename        cf. ExpVerticesCommand
```

expgeom filename

Export the finest triangulation in GeomView format. All vertices, but only the boundary triangles are exported.

```
Function : ExpGeomCommand
Purpose  : Export the coordinates of all boundary vertices and the
           ids of the vertices of all boundary faces to a
```

```
           file (.off) of the following structure (geomview format)
           OFF
           number_of_vertices number_of_sides 0
           x-coordinate y-coordinate z-coordinate
           ...      number of lines given by number_of_vertices
           3 id_of_first second third_vertex_of_boundary_face
           ...      number of lines given by number_of_sides
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expgeom filename             cf. ExpVerticesCommand
```

Commands for the magnetization application

impdomain filename

Import a domain of arbitrary shape. The domain has to be defined in a file in GeomView format. Only boundary vertices and triangular elements of the surface may appear in the file. Two lines of comments containing the size of the bounding box guarantee optimal rendering in the X window version. If they are omitted, default values will be used (center in (0,0,0) and a bounding cube with edge length 2).

```
Function : ImpDomainCommand
Purpose  : Import the coordinates of all boundary vertices and the
           ids of the vertices of all boundary faces from a
           file (.off) of the following structure (geomview format)
           OFF
           number_of_vertices number_of_sides 0
           # maximum max_x max_y max_z
           # minimum min_x min_y min_z
           x-coordinate y-coordinate z-coordinate
           ...      number of lines given by number_of_vertices
           3 id_of_first second third_vertex_of_boundary_face
           ...      number of lines given by number_of_sides
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : impdomain  filename
```

impelements filename

Create new elements. The first column gives the element id, the second the property id, the third, fourth, fifth and sixth column of the file give the ids of the four vertices of each element (cf. `element` command).

```
Function : MagImpElementsCommand
Purpose  : Import the ids of the four vertices of all elements
           which form the finest triangulation
```

```
          i.e. those which were not refined any further and therefore
          have no sons to a file (.ele) with the following structure
          elemid propid id_of_first second third fourth_vertex
Input   : int     argc    - number of arguments
          char  **argv    - arguments
Remark  : Syntax : impelements filename
```

imperror filename

Import the estimated error and store it in the data structure of the corresponding element. The first token in each line is the element identifier, the second token the error. Only elements of the finest triangulation, which have no sons, can be assigned an error. Thus, the mesh must not be manipulated between the execution of `expelements` and `imperror`.

```
Function : ImpErrorCommand
Purpose  : Save the estimated error in the data structure of each
           element.
           The element id corresponds to the line number minus 1
           of the file created by ExpElementsCommand
           The following file structure is expected:
           element_id   error
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : imperror filename
```

impsol filename

Import the magnetization vectors and store them in the data structure of the corresponding vertices. In the first column of the input file the vertex id, in the second, third, and fourth the components of the magnetization in x-, y-, and z-direction, respectively, are expected.

```
Function : ImpSolCommand
Purpose  : Save the magnetization vector in the data structure of each
           vertex.
           The following file structure is expected:
           vertex_id Mx My Mz
           ...
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : impsol filename
```

expsol filename

Export the solution of each vertex to a file. Each line contains the normalized magnetization vector and the vertex id increased by one. While the output

file is created or overwritten the solution is also printed in the text window of AGM[3D]for X window. If a vertex does not have a valid solution, a warning is printed in the text window.

```
Function : ExpSolCommand
Purpose  : Export the magnetization stored in the data structure of
           the vertices to a file with the following structure:
           vertex_id  Mx  My  Mz
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expsol filename
```

expelements filename

Export the identifiers of the four vertices of each element contained in the finest triangulation. The file is created or overwritten. In each line of the file there are the element id, property id, id of first, second, third, fourth vertex, 0, 0. The trailing zeros are for compatiblity reasons. The order of output lines represents the order of elements in the list they are stored in. The elements are not necessarily numbered consecutively, because many "father" elements, which have been refined, do not belong to the finest triangulation. Therefore their sons or even grandsons, which have higher ids, are saved in the file.

```
Function : MagExpElementsCommand
Purpose  : Export the ids of the four vertices of all elements
           which form the finest triangulation
           i.e. those which were not refined any further and therefore
           have no sons to a file (.ijk) with the following structure
           elemid propid id_of_first second third fourth_vertex  0 0
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : expelements filename         cf. ExpVerticesCommand
```

vemcall filename

This command could be used to automatically call `vecmesh` and `vecu`. However, this is not very flexible and therefore not recommended.

```
Function : VemcallCommand
Purpose  : Call vecmesh and vecu from AGM
Input    : int     argc    - number of arguments
           char  **argv    - arguments
           char   *argv[1] - name of the simulation (e.g. cube)
           cube.exc        - file containing the exit code
Remark   : Syntax : vemcall cube
```

vertexinfo vnum
elementinfo elnum

These two commands have been improved to display information contained in the data structures of vertices and elements. After the output by the original routines additional information is printed. If the magnetization vector stored in the user data structure of the vertex with id vnum is marked valid, `vertexinfo` displays it. `elementinfo` prints the error stored in the element's data structure if it is marked valid.

```
Function : MagVertexInfoCommand
Purpose  : Overwrite the kernel vertexinfo command
           In addition to the output of the standard vertexinfo command
           print the current solution (magnetization) if it is valid.
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : vertexinfo id

Function : MagElementInfoCommand
Purpose  : Overwrite the kernel elementinfo command
           In addition to the output of the standard vertexinfo command
           print the error if it is valid.
Input    : int     argc    - number of arguments
           char  **argv    - arguments
Remark   : Syntax : elementinfo id
```

2.7 Error estimation, refinement and interpolation

For the refinement of the finite element mesh certain elements have to be selected and split into smaller elements as described in [Bey95] and [Scho97] . The selection is based on an estimated error, which is imported and stored in the data structure of the elements by the `imperror` command. The estimator has the task to decide which elements have to be refined. Currently five estimators are available.

The user can choose one of them by setting the variable `estimator` to the appropriate name. The tolerance has a different meaning for each estimator. Its default values for the refinement (0.5) and coarsening (0.1) tolerance can be changed using the variables `refinetol` and `coarsentol`. If `estimate` is invoked without any arguments the values stored in the variables `refinetol` and `coarsentol` will be handed over to the estimator. Else the values given as a arguments will override the variables. `estimate` invokes the estimator specified by `estimator`, which will mark all elements necessary. If an irregular element would be selected, its father is marked for regular refinement, because Bey's refinement algorithm prohibits the refinement of an irregular element.

```
Function : EstimateCommand
Purpose  : Estimate error and mark elements
Input    : int      argc - number of arguments (incl. its own name)
           char   **argv - array of arguments
Output   : BOOL   - FALSE for any error
Remark   : Syntax : estimate <refinetol>  <coarsentol>
```

- **markall** estimator

  All leaf (i.e. unrefined) elements will be marked for regular refinement. The command `mark all` provides a more convenient way to refine the whole mesh.

- **abs** estimator

  All elements whose error is greater (smaller) than `refinetol` (`coarsentol`) will be marked for refinement (coarsening).

- **etamax** estimator

  If the error η of an element of the finest triangulation complies with $\eta > (\eta_{\max} \cdot$ `refinetol`$)$, it is marked for regular refinement. If it complies with $\eta < (\eta_{\max} \cdot$ `coarsentol`$)$, it is marked for coarsening.

- **eps** estimator

  Those elements for which $\sqrt{N \cdot \eta^2} >$ `refinetol` ($\sqrt{N \cdot \eta^2} <$ `coarsentol`), where η is the error of the current element and N the total number of finite elements, is true, are marked for regular refinement (coarsening).

- **diff** estimator

  First the difference between the two magnetization vectors at the ends of each edge of the current element is calculated. The length of each of these six difference vectors is computed and finally summed up. If the result is greater (smaller) than `refinetol` (`coarsentol`) the element is marked for refinement.

After execution of `estimate` the desired elements are marked for refinement and coarsening. If too few or too many elements are marked or another estimator is chosen, `estimate` can be called again, which will first clear all marks and evaluate the elements' errors again.

The refinement itself can be effected by the command `refine`. It will insert all necessary elements and vertices. However, these newly inserted vertices contain no valid magnetization vector yet. It can be computed with `interpolate` (implemented in `mga.c`). This command interpolates linearly between the magnetization vectors at the ends of the edge which is bisected

by the inserted vertex and stores the result in the new vertex. Finally, the length of the magnetization vector is changed to 1 if it is greater than 10^{-20}. Otherwise it remains unchanged, which leaves it in a rather unphysical state. The magnetization vector can be very short under two circumstances. Either the magnetization vectors that are used for the interpolation are of almost equal length and point in opposite directions or they are very short themselves.

In the first case the new vertex lies between two different domains in a domain wall. This could either be a Bloch wall (magnetization vector rotates in the plane of the wall) or a Néel wall. Since the interpolated vector is very short there is no "domain wall", rather an area with vanishing magnetization. The second case should not occur since the length of all magnetization vectors computed by `vecu` have unit length.

```
Function : InterpolateCommand
Purpose  : Interpolate solution for new vertices
Input    : int      argc - number of arguments (incl. its own name)
           char   **argv - array of arguments
Output   : BOOL   - FALSE for any error
Remark   : Syntax : interpolate
```

2.8 Example scripts

Example scripts for the pentahedron, hexahedron and the arbitrary domain of figure 3 can be found in the subdirectory `demos.mag`. To test these scripts AGM^{3D} has to be compiled for the magnetization application. However, they can be used with both, the X window and batch version.

3 Programmer's manual

3.1 Data structures, control flags and macros

The user data structures and the macros for accessing them are defined in `magproblem.h`, the control flags and their macros in `mag.h`.

Elements

Each element can contain information about the error which was estimated for this element. Whenever a new element is created (function `CreateElement` in `memory.c`), heap space is allocated for the user data structure. For the `magnetization` problem this is one variable of type `COORD` (cf. `misc/misc.h`), which stores the estimated error. However, there is no "default" error for a

new element. Therefore it is marked to contain no valid error. The indicator for this property is the first bit of the element's control word. If it is set (1) the error is valid, if it is cleared (0) the error is not valid. Then the value of the error can be imported using the `imperror` command (cf. section 2.6), which automatically sets the corresponding bit in the control word. The latter is done by the macro `SET_ERROR_OK`. `ERROR_OK` can be used to determine the current status and `RESET_ERROR_OK` to reset the error bit. The error itself can be accessed by the `ERROR` macro.

Vertices

In the user data structure of each vertex a magnetization vector can be stored if a `magnetization` problem is initialized. The third bit of the control word indicates if the values for the magnetization vector are valid or not. There are three macros `SET_SOL_OK`, `RESET_SOL_OK` and `SOL_OK` They work analogously to the corresponding macros for an element's error. The three components of the magnetization vector are accessible by the macros `M1`, `M2` and `M3`. All geometrical and numerical data belonging to one vertex are stored in its data structure. Therefore vertex and node are used equivalently. On the contrary, Bey's convection-diffusion problem of AGM^{3D} does distinguish between nodes and vertices: Geometrical and graphical information is stored in the vertices' data structure whereas numerical data are contained in the nodes' data structure.

3.2 Adding variables and commands

Variables

1. Define the name of the new variable in `extend.h`:
   For example `#define REFINETOL_VARNAME "refinetol"`

2. Define its default value:
   `#define DEFAULT_REFINETOL  0.5`

3. Declare it as an extern variable:
   `extern double   RefineTol;`

4. "Create" it with `Create`*xxx*`Variable` in `extend.c:InitExtension`, where (*xxx*) is the desired type. There are macros for several types defined in `misc/shell.h`.

Commands

1. Define the name of the new command in `extend.h`:
   For example `#define TEST_CMDNAME "test"`.
2. Implement its function in `extend.c`:
   `BOOL TestCommand (int argc, char **argv);`
3. "Create" it with
   `CreateCommand(TEST_CMDNAME,TestCommand,NULL);`

3.3 Files in the `extend` directory

`extend/extend.c`

Herein all commands for data import and export are implemented.

`extend/extend.h`

Definition of new command names, error messages and default values for variables.

3.4 Files in the `mag` directory

`mag/extdom.c`

This file contains the parameter projection functions for boundary midnode projection, boundary segment function for the hexahedron, pentahedron and arbitrary domains and initialization functions for all new domains.

`mag/extdom.h`

Definition of new domain names.

`mag/mag.c`

In this file all estimators, the new `estimate`, `vertexinfo`, `elementinfo` and `interpolate` commands are implemented

`mag/mag.h`

Definition of new command names (implemented in `mag.c`), estimator names and macros for control word access.

`mag/mag.msg`

Error messages of initialization routines.

`mag/magdomain.c`

This is only a copy of `cd/domain.c` to separate the applications (convection diffusion and magnetization) properly.

`mag/magdomain.h`

This is only a copy of `cd/domain.c` for the above reasons.

`mag/magproblem.c`

Various initialization routines.

`mag/magproblem.h`

Macros for acces of new data structures.

`mag/mainb.c`

This is the main program for the batch version of magnetization application.

`mag/mainx.c`

The main program for the X window version is only a copy of `cd/mainx.c`.

3.5 Modifications of other AGM^{3D} source files

In several files, function prototypes, standard include files and options for conditional compilation have been added. The source should now compile without any errors or warnings with the GNU C compiler.

`cd/cd.c`

Added option for conditional compilation of application independent extensions.

`cd/cd.h`

Corrected misspelling (FVAssembleRHSElement instead of FVAssembler-RHSElement).

`cd/domain.h`

Boundary segment functions need the segment id as the first argument. This modification was necessary to cope with arbitrary domains with many boundary segments (=boundary triangles).

`gui/gui.c`

At the end of `GUI_InsertElementsCommand` the call of `GUI_UpdateCommand` has been removed to speed up the import of many elements with `impelements`.

`gui/plot3d.c`

Boundary segment functions need the segment id as the first argument.

`kernel/cmd.c`

Boundary segment functions need the segment id as the first argument. `InsertElementCommand` must not check for `COMPLETE(theMG)`. Then it is possible to handle disjoint meshes.

`kernel/memory.c`

Boundary segment functions need the segment id as the first argument. `CreateMultiGrid` must not complain about unused vertices when handling disjoint meshes.

`kernel/objects.h`

The definition of unsigned long constants (the masks for control word access) was modified for the 64-bit alpha processor.
The `bndsegment` structure was extended to save the coordinates of one corner and the vectors to the other two.

`misc/misc.h`

The definition of unsigned long constants (the masks for control word access) was modified for the 64-bit alpha processor.

References

[Bey94] J. Bey, *AGM^{3D} Manual.* Tübingen, 1994

[Bey95] J. Bey, *Tetrahedral Grid Refinement.* In *Computing 55*, p. 355, Springer, 1995

[Kiku86] N. Kikuchi, *Finite Element Methods in Mechanics.* Cambridge University Press, Cambridge, 1986

[Schr91] T. Schrefl, J. Fidler, *Numerical simulation of magnetization reversal in hard magnetic materials using a finite element method.* J. Magn. Magn. Mater. 111, p. 105, 1992

[Bagn91] A. Bagnérés-Viallix, P. Baras, J.B. Albertini, *2D and 3D calculations of micromagnetic wall structures using finite elments.* IEEE Trans. Magn., Vol. 27, No. 5, 1991

[Scho97] W. Scholz, *Verfeinerung von Finite-Element-Gittern.* Projektarbeit, 1997