Projektarbeit über Dauermagnetwerkstoffe
LV-Nr. 133.016

Verfeinerung von Finite-Element-Gittern

ausgeführt am Institut für Angewandte und Technische Physik
der Technischen Universität Wien

unter der Anleitung von Univ.-Prof. Dr. Josef Fidler
und Dipl.-Ing. Dr. Thomas Schrefl

durch

Werner Scholz
Margaretengürtel 62-64/63/3
1050 Wien
Matr.Nr. 9426502

Wien, Februar 1997

Contents

1 Introduction

Numerical computer simulations of magnetization processes help understanding and improving the properties of magnetic materials. New alloys can be "designed" and their specific behaviour predicted. Their magnetic properties are described by the dynamic (time dependent) micromagnetic Landau-Lifshitz-Gilbert equations. Finite element methods provide a reliable technique for the solution of these partial differential equations. They are very flexible concerning material parameters, desired accuracy, and material inhomogeneities.

The error caused by the discretization of the problem can be estimated during the calculations. It will be small in areas where the magnetization is uniform and larger where the magnetization changes quickly. An appropriate local refinement strategy will add elements in those areas with large errors. Consequently the discretization error will be reduced and more accurate results obtained.

It is the purpose of this paper to present an implementation of a suitable refinement algorithm and an application, the simulation of a hard magnetic cube.

2 Finite element meshes

The sample under consideration has to be descretized in the three dimensions of space. This is achieved by its division into sufficiently small finite elements, which can have the shape of cubes, hexahedrons or tetrahedrons.

Usually the initial triangulation is a uniform mesh, consisting of a suitable number of elements and nodes. If no refinement is applied, this initial mesh is of great importance for the whole simulation. A small number of elements and nodes limits the achievable accuracy and reliability and implies poor approximation near singularities and internal or boundary layers. On the other hand if a finer mesh is chosen, accuracy can be improved significantly but at the same time the systems of equations enlarge, which may lead to intolerably long computation times.

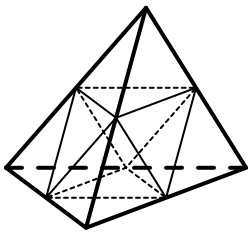
There are several methods to improve the results and almost preserve the calculations' complexity (cf. [Kiku86]). First the nodes of the finite element grid can be relocated which leads to a higher density of nodes in areas with larger integration errors. The number of elements remains fixed but they are deformed. This may lead to undesired shapes and deteriorating convergence rates. As an alternative the degrees of polynomials used for the interpolation of the desired function can be changed.

Finally, the number of unknowns can be optimized by fitting the corresponding discretization to the present approximate solution. In regions where improved accuracy is needed, the underlying discretization mesh is refined and new elements and vertices are inserted. (In this paper vertex and node are used as synonyms.) However, where the solution is expected to be smooth, it is possible to coarsen the mesh. To decide whether an element has to be refined or not the error which originates from the discretization of integrations has to be estimated for each element.

Methods which use this technique are called multi-level methods. They have been proven to be of optimal or nearly optimal complexity for the solution of discrete systems arising from partial differential equations.

3 Refinement

3.1 Regular Refinement



The regular refinement of a tetrahedron called the "father" is done by subdividing it into smaller subtetrahedra called "sons". The first step is to find the midnodes of all edges of the father and to connect those on a common face. Then the four sons at the corners, which are obviously congruent with their father and have equal volume, can be "cut off". The remainder, an octahedron, has to be divided again, but this can be done in several ways. It can be cut along three different planes, which have the shape of parallelograms. Cutting along two of them generates four sons which are no longer congruent with the father. This strategy is equivalent to the insertion of one edge, namely the cutting edge of the above planes. However, the wrong choice can lead to degenerated elements whereas the choice of the shortest of the three possible edges will minimize the maximum measure of degeneracy of the sons ([Bey95], p. 362). If the initial elements have non-obtuse faces, this strategy produces at most three congruence classes.

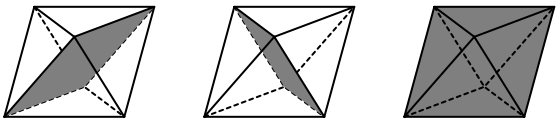


Figure 1: Cut planes of the octahedron

3.2 Irregular Refinement

In order to restrict the refinement of elements to certain areas of the finite element mesh, the triangulation has to be closed after the regular refinement of the desired elements. Otherwise there would be so called hanging nodes, which are difficult to handle in calculations. The "smooth" transition from regular elements to regularly refined elements is done by the irregular elements. They are generated by a procedure called the green closure.

An edge is called refined if at least one of the neighbouring elements which share this edge is refined regularly. This means that a new midnode is inserted in the middle of the edge. Each tetrahedron has 6 edges which can be either refined or not. Therefore there exist $2^6 = 64$ edge refinement patterns. One of them leaves all edges unrefined and a second refines all edges, which occurs if the element itself or all its neighbours are refined regularly. Considering symmetry arguments the remaining 62 patterns can be divided into 9 different types. Using these 9 types, any triangulation can be closed. For practical reasons it is sufficient to use four different types and refine those elements regularly which would need one of the missing refinement rules.

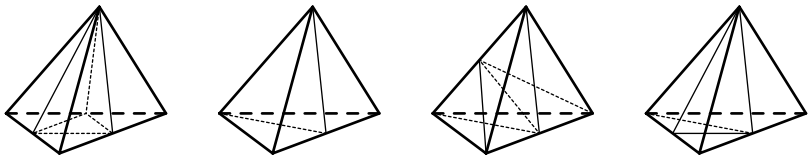


Figure 2: Irregular refinement patterns for the green closure

3.3 The refinement algorithm

As mentioned above the initial triangulation must not include elements with obtuse faces in order to preserve stability. If an appropriate finite element mesh is created, the finite element calculations can be performed to produce the desired results and select certain elements for regular refinement. All selected regular or regularly refined elements are refined as described above. Irregular elements must not be refined because this could lead to degenerated elements and disturb the calculations' stability. As a consequence irregular elements and their "brothers", who belong to the same father, are removed and their father is refined regularly.

After performing this first refinement called the red closure, the triangulation is closed with irregular elements (green closure). If none of the four irregular refinement patterns is applicable for a certain element, it is refined

regularly and its neighbouring elements are new candidates for the green closure.

This procedure is applied until all selected elements are refined and the whole grid is closed properly.

4 AGM^{3D} and its extensions

4.1 Overview

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 grids and manages all data necessary for the calculations. Its main purpose is the refinement of the grid according to the refinement algorithm described in section 3. Consequently only problem dependent parts such as the mathematical model or error estimators have to be implemented. The manual of AGM^{3D} describes all problem independent features in detail. Hence, the following sections describe only extensions and new features.

4.2 User manual

4.2.1 The new magnetization problem

This problem has been implemented only for the cube domain but has no solving capabilities. It just provides a "clean" environment without the original convection diffusion problem supplied by Bey.

4.2.2 New variables

estimator [etamax]

The current error estimator used by the `estimate` command. Default is the etamax estimator (cf. section 4.2.4).

tolerance [0.5]

The parameter used by several estimators. Its exact meaning depends on the estimator defined by the variable `estimator` (cf. section 4.2.4).

4.2.3 New commands

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.

expvertices [<filename>]

Export the real world coordinates of all vertices to a file. If no filename is given, the list of all vertices and their coordinates is printed in the text window of AGM^{3D}. Else the file is created or overwritten and simultaneously the information is printed in the text window (in a more user friendly format, not the format of the output file). In each line of the file there are the x, y and z-coordinates of a vertex with a precision of 4 digits following the decimal point.

Since the program `mesh33` assumes that the line number corresponds with the identifier of the vertex, the lines have to be in the correct order. However, the vertices are stored in a double linked list in the order of their creation. A new vertex is inserted at the beginning and its identifier is derived from the last id which was assigned to a vertex. This number is stored in the variable `VertexCounter` in the data structure of the `multigrid`. It does not indicate the total number of vertices but is a simple counter, which is updated whenever a new vertex is created. This happens for example if the grid is refined. Though, the counter is not decreased if the grid is coarsened and a vertex is removed. As long as the grid is not coarsened the vertices are numbered consecutively. It is assumed that this the case. Then the output starts with the last vertex in the list. It was created first and therefore has id 0. The previous vertex is reached by the `PRED` macro. For security reasons an `assert` command checks if the output is in the correct order and numbered consecutively.

Furthermore `mesh33` expects the centre of the coordinate system to lie inside the domain. By definition vertex no. 0 has the coordinates (0, 0, 0). Consequently all vertices are moved by $(-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$ before being saved to file. However, the unchanged coordinates are indicated in the text window.

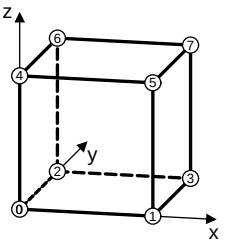


Figure 3: Unit Cube

expelements [<filename>]

Export the identifiers of the four vertices of each element contained in the finest triangulation. If a filename is given the file is created or overwritten. In any case the list is printed in the text window.

The vertices are numbered from 0 onwards by AGM^{3D}. Since the identifiers are expected to start from 1 they are increased by one for the output file. The ids printed in the text window remain unchanged. The order of output lines represents the order of elements in the list they are stored in. The `imperror` command assumes that this order is not changed. There are two reasons why the element's identifier is not taken into account. First, it would be quite inefficient to scan the whole list for each element to get them in the right order. Secondly, there are many "father" elements which are important for the multi-grid algorithm but do not belong to the finest triangulation. Other programs expect the elements of the mesh to be numbered consecutively, which is impossible with the internal ids since these "father" elements have to be left out.

expboundary [<filename>]

Export the vertices and faces of the boundary. If a filename is given, the output file contains information about the vertices and faces on the boundary in a format which can be interpreted by `omega33` and `geomview`. As mentioned above the vertices have to be moved by $(-\frac{1}{2}, -\frac{1}{2}, -\frac{1}{2})$.

imperror <filename>

Import the estimated error and store it in the data structure of the corresponding element. The first token of each line is the element identifier, the second token the error. However, the element id is ignored. The line number of the input file has to correspond with the position of the element in the list of elements. The error in the first line is stored in the first element in the list that was not refined and therefore belongs to the finest triangulation. The error in the second line is stored in the second element in the list that was not refined, and so on. This is exactly the order in which the quadruplets of vertex ids were stored by `expelement` and which has to be preserved. Consequently the grid must not be manipulated between the execution of `expelements` and `imperror`. If the number of lines is smaller than the number of elements contained in the finest triangulation, a warning will appear in the text window.

impsol [<filename>]

Import the magnetization vectors and store them in the data structure of the corresponding vertices. The file structure is described in the source code of `extend.c`.

If no filename is given, a default magnetization will be created: All vertices lying above the plane through $(0, 0, \frac{1}{2})$ parallel to the x-y-plane will be assigned the magnetization vector (-1, 0, 0). All vertices lying in or below that plane will be assigned a magnetization vector (+1, 0, 0). Thus two domains are created. The upper one is magnetized in negative x-direction, the lower in positive x-direction.

expsol <filename>

Export the solution of each vertex to a file. Each line contains the value of magnetization in the three directions of space and the vertex id increased by one. Here the same problems with the sequence of vertices occur as in `expvertices`. While the output file is created or overwritten the solution is also printed in the text window of AGM^{3D}. If a vertex does not have a valid solution, an appropriate message is printed in the text window.

impsigma <filename>

Import the parameter sigma for the `etamax` estimator and store it in the global variable `tolerance`. The value has to be the first token in the fourth line of the input file.

wait

Wait until a key is pressed. This is useful in scripts for AGM^{3D} if the execution of commands should be paused. If `[Ctrl-Brk]` is pressed the script is terminated. Any other key resumes.

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.

4.2.4 Error estimation, refinement and interpolation

For the refinement of the finite element grid certain elements have to be selected and split into smaller elements as described in section 3. 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 value 0.5 can be changed using the variable `tolerance`. If `estimate` is invoked without any arguments the value stored in the variable `tolerance` will be handed over to the estimator. Else the value given as an argument will override the variable `tolerance`. `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 an irregular element being refined.

- **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 grid.

- **abs** estimator

  All elements whose error is greater than the given tolerance will be marked.

- **etamax** estimator

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

- **eps** estimator

  Those elements for which $\sqrt{N \cdot \eta^2} > \texttt{tolerance}$, where η is the error of the current element and N the total number of finite elements, is true, are marked for regular refinement.

- **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 than the allowed `tolerance` the element is marked for refinement.

After execution of `estimate` the desired elements are marked for refinement. 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 a non-magnetized area. The second case should not occur since the length of all magnetization vectors computed by `vecu` have length 1.

4.2.5 Restrictions

Currently only the unit cube is available for the magnetization problem.

4.3 Programmer's manual

4.3.1 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 that 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 4.2.3), 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.

4.3.2 Adding variables and commands

Variables

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

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

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

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);`

5 Application

5.1 Hard magnetic cube

In order to test the new technique the magnetization of a hard magnetic cube has been calculated and the new refinement algorithm applied. The cube's material parameters are typical of a $Nd_2Fe_{14}B$-magnet. The length of its edges is 200 nm. The initial magnetization consists of two domains: The upper half of the cube is magnetized parallel to the x-axis in negative direction, the lower half in positive direction. Consequently there is a domain wall in between. It can be examined in greater detail by refining the mesh in this area.

First a new document (with the name `demo`) is created and initialized with the `magnetization` problem on the `cube` domain using the `multigrid` algorithm.

```
open demo cube magnetization multigrid
```

Then the cube is triangulated using Kuhn's method. (See figure 3 for the corners' numbering.)

```
element 0 1 3 7
element 0 1 5 7
element 0 2 3 7
element 0 2 6 7
element 0 4 5 7
element 0 4 6 7
```

After refining the whole grid two times, the cube consists of $6 \cdot 4^3 = 384$ tetrahedrons and $4^3 + 4^2 + 4 \cdot 5 + 5^2 = 125$ vertices. The initial triangulation and the twice refined mesh are shown in figure 5.1. With `impsol` (cf. section 4.2.3) the two desired domains of magnetization are initialized.

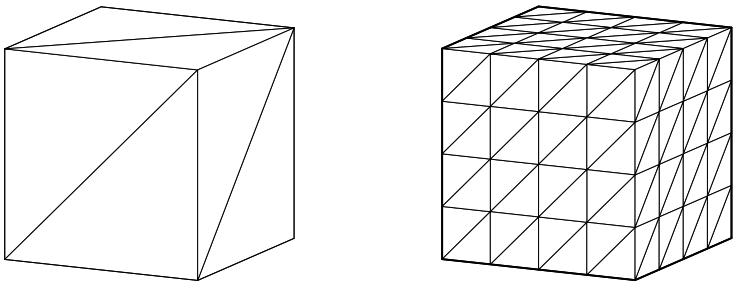


Figure 4: Initial triangulation and second refinement

The initial triangulation should be as coarse as possible, just fine enough to resolve the shape of the domain and the coefficient jumps in the problem under consideration. However, it can be expected that the grid will have to be refined at the domain wall, which will be very thin in a hard magnetic material (typically some nanometres). Therefore we start with a sufficiently fine grid. Now the files containing all information necessary for `vecu` can be generated:

```
expvertices cube.knt
expelements cube.ijk
expboundary cube.off
expsol cube.mag
```

Then `mesh33`, `material2`, `omega33`, `matrix2`, and `vecmesh32` do the preprocessing before `vecu33` can be invoked. Following `vecu`'s calculations the results have to be imported in AGM^{3D}. Figure 7 shows the total energy stored in the magnetization of the cube as a function of the simulation time.

```
impsol cube.inp
imperror cube.ind
```

These commands will assign to each vertex its calculated magnetization vector and to each element its estimated error.

The estimated error has to be evaluated and for each element decided if it has to be refined or not. For this task there are several estimators, which are described in section 4.2.4.

```
set estimator = etamax
set tolerance = 0.5 estimate
```

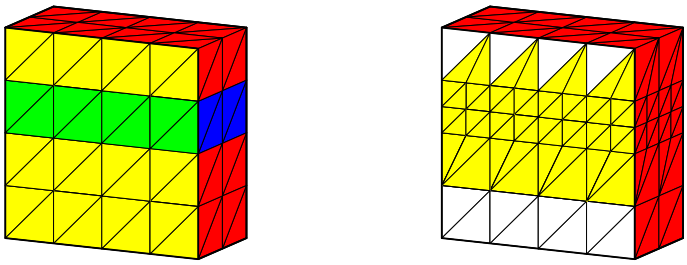


Figure 5: First local refinement around the domain wall

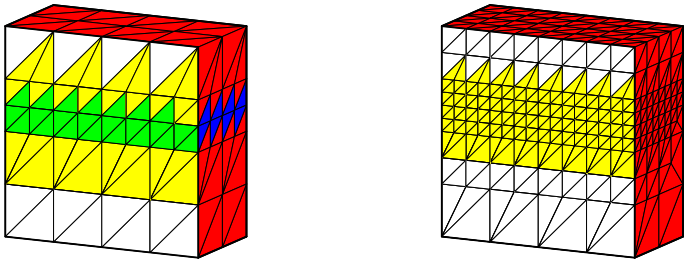


Figure 6: Second local refinement

All marked elements are coloured blue (boundary faces) and green (cut plane). In figure 5 the faces of all marked elements of the left cube appear darker than the others. Then `refine` starts the refinement algorithm described in previous sections and creates the triangulation shown on the right of figure 5. It consists of 318 vertices and 1,312 elements. However, the new vertices contain no valid magnetization vector yet. These have to be created by interpolation with the command `interpolate`.

Finally, the output files can be generated as described above, the new calculation started, and the grid refined again. The result shown in figure 6 is a grid of 1,774 vertices and 8,867 elements.

5.2 Simulation results

The total energy stored in the magnetization of the cube as a function of the simulation (CPU) time (i.e. the time necessary for the simulation) is shown in figure 7.

During the first 320 seconds the total energy decreases very fast and reaches a minimum value of about -0.672. Then it oscillates slightly and remains stable after 1,000 seconds. Another 640 seconds later the finite element mesh is refined locally as shown in figure 5, but this could have already

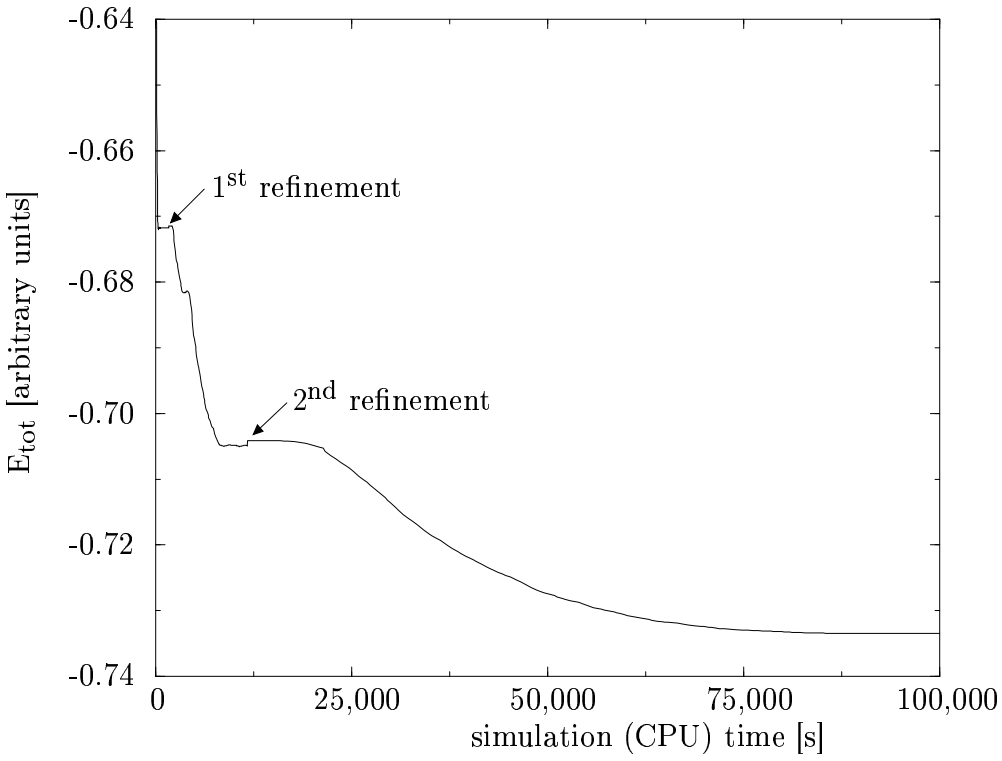


Figure 7: Total energy as a function of the simulation time

been done 1,000 seconds after the beginning of the simulation when the total energy remained constant. Immediately the total energy decreases again. The finer mesh permits a more accurate simulation and a more realistic magnetization, which leads to a lower total energy. After a simulation time of 8,000 seconds it arrives at a minimum of -0.705.

The second refinement is applied after 11,700 seconds. The new magnetization vectors originating from the inserted vertices and the interpolation between their neighbours cause a slightly higher total energy, which can be observed as a small step in figure 7. Resuming the simulation, the total energy remains constant at first. After 20,000 seconds it decreases monotonously to its final value of -0.73374. It is reached after 150,000 seconds

Another interesting aspect is the relation between the simulation time and the simulated time shown in figure 8. It indicates how much time and computing power are necessary for the simulation.

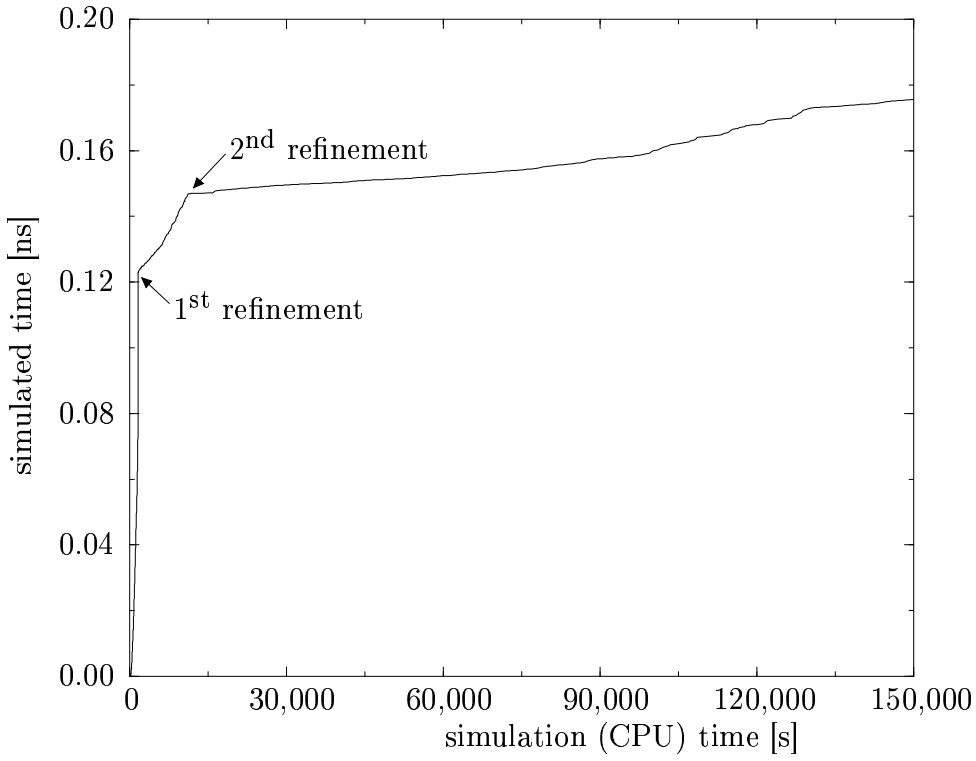


Figure 8: Simulated time as a function of the simulation time

Within some seconds of computing time a simulated time of 0.12 ns has passed. There are two reasons for this behaviour: First, the finite element grid consists of 125 vertices and 384 elements only, which makes the calculations quite simple and fast. Secondly the magnetization and the total energy reach very quickly constant values. Therefore the time step resulting from the discretization of time can be enlarged leading to a rapid development of the simulated time.

After the first local refinement the grid consists of 318 vertices and 1,312 elements. Consequently the simulation becomes more complicated and slower. Finally, 1,774 vertices and 8,867 elements assemble the finite element grid.

Yet, it is not possible to decide if the domain wall is of Bloch or Néel type. Some estimated 50,000 elements would be necessary to find the right answer for this hard magnetic cube.

6 Conclusion

The local refinement of finite element meshes proved to be an appropriate method to increase the accuracy and reliability of numerical simulations based on finite element methods. If the initial mesh is generated carefully AGM^{3D} and its new features provide reliable tools for error estimation and refinement.

An interface between AGM^{3D} and `vecu` has been created by the implementation of new commands. They save all graphical data of the finite element mesh in a format compatible with `vecu` and its preprocessing programs. Moreover they give access to the results obtained by `vecu` and prepare the mesh for further calculations.

The simulation of a hard magnetic cube served as a test for both the program and the technique itself. While the new commands showed their reliability, the program and the technique remain to be "refined". Currently finite element meshes are available only for cubic domains. Yet, AGM^{3D} is supplied with several error estimators and flexible input- and output-routines which need no modification if new domains are added.

Another problem is the fast growing number of elements and vertices if the grid is refined. As a consequence the calculations become very complex and slow. Up to around 2,000 elements results can be obtained within reasonable time. If the grid is refined another time it may end up with some $10,000$ elements. However, as computing power and speed are improving, these limits are pushed forward. Alternatively special computers may be used. Finite element methods require the solution of large systems of linear equations. They can be solved very efficiently with vector computers which are optimized for the manipulation of vectors and matrices. The NEC SX4 B/2 at the Centre for Computing Services at the Technical University Vienna is a good example for this type of machines. It is ranked among the 20 fastest vector computers in the world.

Another problem which has been mentioned in [Bagn91] may occur. The refinement takes place in areas of major interest such as domain walls. It should improve accuracy and reliability of the simulation though it should not influence the sample and its properties. However, it has been observed that domain walls may move from finer areas to coarser areas of the grid. This behaviour can be explained by the exchange energy, which is in a finer grid different from that in a coarser grid and therefore does influence the calculations.

During the simulation of the hard magnetic cube it was not possible to verify these facts but further research work will have to take them into consideration.

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**

[Sch91] 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**