OOMMF
User's Guide

August 12, 1998

Abstract

The Object Oriented Micromagnetic Computing Framework (OOMMF) project in the Mathematical and Computational Sciences Division (MCSD) of ITL/NIST, is aimed at developing portable, extensible public domain micromagnetic programs and tools. The end goal is a completely functional, user-friendly micromagnetic code, with a well documented, flexible programmer's interface that will allow developers outside the OOMMF project to swap their own code in and out as desired. The code is written primarily in C++ with a Tcl/Tk (and in the future, possibly OpenGL) interface. Target systems include a wide range of Unix platforms, Windows NT, and Windows '95.

Contents

Disclaimer ii

1 Overview of OOMMF 1

2 Installation 3
2.1 Requirements . . . 3
2.2 Basic Installation . . . 3
2.2.1 Download . . . 3
2.2.2 Configuration . . . 4
2.2.3 Compiling and Linking . . . 5
2.2.4 Installing . . . 5
2.2.5 Using OOMMF Software . . . 5
2.3 Advanced Installation Options . . . 5
2.3.1 Managing OOMMF platform names . . . 6
2.3.2 Reducing Disk Space Usage . . . 7
2.3.3 Microsoft Windows Options . . . 7

3 Quick Start: Example OOMMF Session 9

4 OOMMF Launcher/Control Interface: mmLaunch 12

5 Micromagnetic Problem Editor: mmProbEd 13

6 Vector Field Display: mmDisp 14

7 The 2D Micromagnetic Solver: mmSolve2D 15

8 OOMMF Batch System 16
8.1 Batch solver . . . 16
8.1.1 Usage . . . 16
8.1.2 Output . . . 17
8.1.3 Programmer's interface . . . 18
8.2 Batch scheduler . . . 18
8.2.1 Usage . . . 19
8.2.2 Overview . . . 19
8.2.3 Batch master and slave specifics . . . 19
8.2.4 Batch task scripts . . . 20

8.2.5 Sample task scripts 22

9 File Formats **26**
9.1 Problem specification format (MIF) 26
9.2 Data table format (ODT) 32
9.3 Vector field formats (OVF and SVF) 33
9.3.1 The OVF format 34
9.3.2 The SVF format 38

10 Credits **40**

Disclaimer

Recipients of this software assume all responsibilities associated with its operation, modification, and maintenance.

This software was produced by NIST, an agency of the U.S. government, and is by statute not subject to copyright in the United States.

Commercial equipment and software referred to on these pages is identified for informational purposes only, and does not imply recommendation of or endorsement by the National Institute of Standards and Technology, nor does it imply that the products so identified are necessarily the best available for the purpose.

1 Overview of OOMMF

The goal of the OOMMF[1] (Object Oriented Micromagnetic Computing Framework) project in ITL/NIST is to develop a portable, extensible public domain micromagnetic program and associated tools. This code will form a completely functional micromagnetics package, but will also have a well documented, flexible programmer's interface so that people developing new code can swap their own code in and out as desired. The main contributors to OOMMF are Mike Donahue, and Don Porter.

In order to allow a programmer not familiar with the code as a whole to add modifications and new functionality, we feel that an object oriented approach is critical, and have settled on C++ as a good compromise with respect to availability, functionality, and portability. In order to allow the code to run on a wide variety of systems, we are writing the interface and glue code in Tcl/Tk[2]. This enables our code to operate across a wide range of Unix platforms, Windows NT, and Windows '95.

The code may actually be modified at 3 distinct levels. At the top level, individual programs interact via well-defined protocols across network sockets. One may connect these modules together in various ways from the user interface, and new modules speaking the same protocol can be transparently added. The second level of modification is at the Tcl/Tk script level. Some modules allow Tcl/Tk scripts to be imported and executed at run time, and the top level scripts are relatively easy to modify or replace. At the lowest level, the C++ source is provided and can be modified, although at present there is no documentation detailing this process.

The first portion of OOMMF released was a magnetization file display program called mmDisp. An alpha release[3] of the complete OOMMF project is now available. This includes a problem editor, a 2D micromagnetic solver, and several display widgets, including an updated version of mmDisp. The solver can be controlled by an interactive interface (Sec. 7), or through a sophisticated batch control system (Sec. 8).

The solver is based on a micromagnetic code that Mike Donahue and Bob McMichael had previously developed. It utilizes a heavily damped Landau-Lifshitz-Gilbert ODE solver to relax 3D spins on a 2D mesh of square cells, using FFT's to compute the self-magnetostatic (demag) field. Anisotropy, applied field, and initial magnetization can be varied pointwise, and arbitrarily shaped elements can be modeled. We are currently working on a full 3D version of this code suitable for modeling layered materials.

[1] http://math.nist.gov/oommf/
[2] http://www.scriptics.com/
[3] http://math.nist.gov/oommf/software.html

If you want to receive email notification of updates to this project, register your email address with the "μMag Announcement" mailing list:

http://www.ctcms.nist.gov/~rdm/email-list.html.

2 Installation

2.1 Requirements

OOMMF software is written in C++ and Tcl. It uses the Tcl-based Tk Windowing Toolkit to create graphical user interfaces which are portable to many varieties of Unix as well as Microsoft Windows 95/NT.

Tcl and Tk must be installed before installing OOMMF. Tcl and Tk are available for free download from Scriptics Corporation[4]. We recommend the latest stable versions of Tcl and Tk (8.0p2 for both as of August 12, 1998). OOMMF requires at least Tcl version 7.5 and Tk version 4.1 on Unix platforms, and requires at least Tcl version 7.6 and Tk version 4.2 on Microsoft Windows platforms. OOMMF software does not support any alpha or beta versions of Tcl/Tk.

A Tcl/Tk installation includes two shell programs. The names of these programs may vary depending on the Tcl/Tk version and the type of platform. The first shell program contains an interpreter for the base Tcl language. In the OOMMF documentation we refer to this program as `tclsh`. The second shell program contains an interpreter for the base Tcl language extended by the Tcl commands supplied by the Tk toolkit. In the OOMMF documentation we refer to this program as `wish`. Consult your Tcl/Tk documentation to determine the actual names of these programs on your platform (for example, `tclsh80.exe` or `wish4.2`).

To build the OOMMF software from source code, you will need a C++ compiler capable of compiling C++ templates, as well as other software development utilities for your platform.

2.2 Basic Installation

Follow the instructions in the following sections, in order, to prepare OOMMF software for use on your computer.

2.2.1 Download

The latest release of the OOMMF software may be retrieved from the OOMMF download page[5]. Each release is available in two formats. The first format is a gzipped tar file containing an archive of all the OOMMF source code. The second format is a `.zip` compressed archive containing pre-compiled executables for Microsoft Windows 95/NT running on an x86-based microprocessor system and compatible with Tcl/Tk version 8.0 in addition to all

[4] http://www.scriptics.com/

[5] http://math.nist.gov/oommf/software.html

the OOMMF source code. Other release formats, including pre-compiled executables for Microsoft Windows 95/NT running on a Digital Alpha Systems RISC-based microprocessor system, and/or compatible with Tcl/Tk version 7.6/4.2 will be made available on request.

For the first format, unpack the distribution archive using gunzip and tar:

```
gunzip -c oommf10a980812.tar.gz | tar xvf -
```

For the other format(s), you will need a utility program to unpack the `.zip` archive. This program must preserve the directory structure of the files in the archive, and it must be able to generate files with names not limited to the traditional MSDOS 8.3 format. Some old versions of pkzip unpacking utility programs do not have these properties. One utility program which is known to work is UnZip[6]. Using your utility, unpack the `.zip` archive, e.g.

```
unzip oommf10a980812.zip
```

2.2.2 Configuration

After downloading and unpacking the OOMMF software distribution, all the OOMMF software is contained in a subdirectory named `oommf`. Start a command line interface (a shell on Unix, or the MS-DOS Prompt on Microsoft Windows), and change the working directory to the directory `oommf`. Then run the Tcl script `platform.tcl`:

```
tclsh platform.tcl
```

(Remember, the Tcl shell program may not have the name `tclsh` on your platform. It may be something like `tclsh8.0` or `tclsh76.exe`. Consult your Tcl documentation, and use the name which works on your computer.)

The output of the script `platform.tcl` is the platform name which the OOMMF software uses to describe your computer. For example, when you run `platform.tcl` on a Silicon Graphics machine, the output is the string `sgi`. The platform name is used by OOMMF in index and configuration files and to name directories so that a single OOMMF installation can support multiple platform types.

Say the output of `platform.tcl` is `foo`. Then you must edit the file `config/cache/foo.tcl` to describe your computing environment to OOMMF. Follow the editing instructions found within the file.

If the output of `platform.tcl` indicates that multiple platform names are compatible with your computer, or if you wish for OOMMF software to use a different platform name for your computer and others like it, see "Managing OOMMF platform names" (Sec. 2.3.1) for further instructions.

[6] http://www.cdrom.com/pub/infozip/UnZip.html

2.2.3 Compiling and Linking

If you downloaded a distribution with pre-compiled executables, skip this section.

The compiling and linking of the C++ portions of the OOMMF software are guided by the application `pimake` ("Platform Independent Make") which is distributed as part of the OOMMF software (in the directory `app/pimake/`). `pimake` is similar in operation to the Unix utility program `make`, but it is written entirely in Tcl so that it will run anywhere Tcl is installed. Just as `make` is controlled by rules in files named `Makefile` or `makefile`, `pimake` is controlled by rules in files named `makerules.tcl`. `pimake` takes an optional argument on its command line which names the target to build. To build all the OOMMF software, run `pimake` in the root directory of the OOMMF distribution.

```
tclsh app/pimake/pimake.tcl
```

When `pimake` calls on a compiler or other software development utility, the command line is printed, so that you may monitor the build process. Assuming a proper configuration for your platform, `pimake` should be able to compile and link all the OOMMF software without error. If `pimake` reports errors, please send the output to the OOMMF developers.

2.2.4 Installing

The current alpha release of the OOMMF software does not support an installation procedure. For now, simply run the executables from the directories in which they were unpacked/built.

2.2.5 Using OOMMF Software

To start using OOMMF software, run the OOMMF bootstrap application `oommf`. This may be launched from the command line interface:

```
tclsh app/oommf/oommf.tcl
```

Or if you prefer, you may now use whatever graphical "point and click" interface your operating system provides for launching programs to launch the OOMMF bootstrap application `./app/oommf/oommf.tcl`. By default, the OOMMF bootstrap application will start up a copy of the OOMMF application **mmLaunch** (Sec. 4) in a new window.

2.3 Advanced Installation Options

The following sections provide instructions for some additional installation options.

2.3.1 Managing OOMMF platform names

OOMMF software classifies computing platforms into different types using the scripts in the directory `config/names` relative to the root directory of the OOMMF distribution. Each type of computing platform is assigned a unique name. These names are used as directory names and in index and configuration files so that a single OOMMF installation may contain platform-dependent sections for many different types of computing platforms.

To learn what name OOMMF software uses to refer to your computing platform, run the Tcl script `platform.tcl` in the OOMMF root directory.

Changing the name OOMMF assigns to your platform To change the name OOMMF software uses to describe your platform from `foo` to `bar`, simply rename the file `config/names/foo.tcl` to `config/names/bar.tcl` and rename the file `config/cache/foo.tcl` to `config/cache/bar.tcl`.

After renaming your platform type, you must recompile your executables.

Adding a new platform type If the script `platform.tcl` reports the platform name `unknown`, then none of the scripts in `config/names/` recognizes your platform type. To add the platform name `foo` to OOMMF's vocabulary of platform names, create the file `config/names/foo.tcl`. The simplest way to proceed is to copy an existing file in the directory `config/names` and edit it to recognize your platform.

The files in `config/names` include Tcl code like this:

```
Oc_Config New _ [file rootname [file tail [info script]]] {
    # In this block place the body of a Tcl proc which returns 1
    # if the machine on which the proc is executed is of the
    # platform type identified by this file, and which returns 0
    # otherwise.
    #
    # The usual Tcl language mechanism for discovering details
    # about the machine on which the proc is running is to
    # consult the global Tcl variable 'tcl_platform'.  See the
    # existing files for examples, or contact the OOMMF developers
    # for further assistance.
}
```

After creating the new file `config/names/foo.tcl`, you also need to create a file `config/cache/foo.tcl`. A reasonable starting point is to copy the file `config/cache/unknown.tcl` for editing. Contact the OOMMF developers for assistance.

Please consider contributing your new platform recognition and configuration files to the OOMMF developers for inclusion in future releases of OOMMF software.

Resolving platform name conflicts If the script `platform.tcl` reports "Multiple platform names are compatible with your computer", then there are multiple files in the directory `config/names/` which return 1 when they run on your computer. For each compatible platform name reported, edit the corresponding file in `config/names/`, so that only one of them returns 1 on your computer. Experimenting using `tclsh` to probe the Tcl variable `tcl_platform` should assist you in this task. If that fails, you can explicitly assign a platform type corresponding to your computing platform by matching its hostname. For example, if your machine's host name is `foo.bar.net`:

```
Oc_Config New _ [file rootname [file tail [info script]]] {
    if {[string match foo.bar.net [info hostname]]} {
        return 1
    }
    # Continue with other tests...
}
```

Contact the OOMMF developers if you need further assistance.

2.3.2 Reducing Disk Space Usage

To delete the intermediate files created when building the OOMMF software from source code, use `pimake` to build the target `objclean` in root directory of the OOMMF distribution.

```
tclsh app/pimake/pimake.tcl objclean
```

The `pimake` target `clean` will also remove the OOMMF executables. The `pimake` target `distclean` will remove all files created by the compile and link steps, leaving the OOMMF files as they were distributed, except for your configuration edits.

2.3.3 Microsoft Windows Options

This section lists the installation options for Microsoft Windows only.

Adding an OOMMF shortcut to your desktop Right mouse click on the desktop to bring up the configuration dialog, and select **New**—**Shortcut**. Enter the command line necessary to bring up OOMMF, e.g.,

```
tclsh80 c:\oommf\app\oommf\oommf.tcl
```

Click **Next¿** and enter `OOMMF` for the shortcut name. Select **Finish**.

At this point the shortcut will appear on your desktop with either the tclsh or wish icons. Right mouse click on the icon and select **Properties**. Select the **ShortCut** tab, and bring up **Change Icon...**. Under **File Name:** enter the OOMMF icon file, e.g.,

```
C:\oommf\app\oommf\oommf.ico
```

Click **OK**. Back on the **Shortcut** tab, change the **Run:** selection to Minimized. Click **OK** to exit the Properties dialog box. Double clicking on the OOMMF icon should now bring up the OOMMF application **mmLaunch**.

Setting the TCL_LIBRARY environment variable If you encounter difficulties during OOMMF start up, you may need to set the environment variable TCL_LIBRARY.

On Windows NT Bring up the Control Panel (for example, by selecting **Settings—Control Panel** off the Start menu), and select **System**. Go to the **Environment** tab, and enter TCL_LIBRARY as the Variable, and the name of the directory containing `init.tcl` for the Value, e.g.,

```
%SystemDrive%\Program Files\Tcl\lib\tcl8.0
```

Click **Set** and **OK** to finish.

On Windows 95 Edit the file `autoexec.bat`. Add a line such as the following:

```
set TCL_LIBRARY=C:\Program Files\Tcl\lib\tcl8.0
```

3 Quick Start: Example OOMMF Session

STEP 1: Start up the mmLaunch window.

- At the prompt, when you are in the OOMMF root directory, type tclsh app/oommf/oommf.tcl
- After a (possibly long) time, you will see a small window labeled mmLaunch. It will come up in background mode, so you will get another prompt in your original window, even before the mmLaunch window appears.

STEP 2: Gain access to other useful windows.

- On mmLaunch window, check localhost box, causing a menu of user account boxes to appear. Then check the box corresponding to the account you want to compute on. This gives a menu of options:
  - **mmProbEd:** to grab/modify a problem
  - **mmSolve2D:** to control the solver
  - **mmDisp:** to display vector fields
  - **mmGraph:** to form x-y plots
  - **mmArchive:** to save vector field data (primitive)
  - **mmDataTable:** to display current values of variables
- Click on **mmDisp**, **mmGraph**, and/or **mmDataTable**, depending on what form of output you want.

STEP 3: Load a problem.

- On mmLaunch window, click on the **mmProbEd** button.
- On mmProbEd window, under File menu, click Open. A "Load MIF File" window will appear.
- On this window:
  - Double click in the "Path" subwindow to change directories. Several sample problems can be found in the directory `oommf/app/mmpe/examples`.
  - To load a problem, double click on a `*.mif` file (e.g., prob1.mif) from the list under the "Filter:" subwindow.
  - Modify the problem as desired by clicking on buttons from the main mmProbEd window (e.g., **Material Parameters**), and fill out the pop-up forms. A completely new problem may be defined this way.

STEP 4: Initialize the solver.

- On mmLaunch window, click on the **mmSolve2D** button.
- Then, when after a (possibly long) time the process appears on the mmLaunch window, check the box next to mmSolve2D. An mmSolve2D window will appear.
- On mmSolve2D window:
  - Check Problem Description under Inputs.
  - Check mmProbEd under Source Threads.
  - Click **LoadProblem**.
  - Wait for more buttons to appear.
  - Check Scheduled Outputs.
  - For each desired output (TotalField, Magnetization, and/or DataTable), specify the frequency of update:
    - Check desired output. This will exhibit the possible output destinations in Destination Threads.
    - Check the box next to the desired Destination Thread. This will exhibit "Schedule" options.
    - Choose a schedule:
      - **every_Iteration:** fill in number and check the box.
      - **every_Equilibrium:** fill in number and check the box.
      - **Interactive:** whenever you click corresponding Interactive output button.

STEP 5: Start the calculation.

- On the mmSolve2D window, start the calculation with **Run** or **Relax**.
- Either before or after this, if you requested mmDataTable output, check the boxes for the desired quantities on the mmDataTable window under the Data menu, so that they appear and are updated as requested in your schedule.
- Similarly, check the box for the desired X-Axis and Y-Axis variables on the mmGraph window(s) under the X-Axis and Y-Axis menus.

STEP 6: Perform midcourse controls as desired.

- On the mmSolve2D window, buttons can stop and restart the calculation:
  - **Reset:** Return to beginning of problem.
  - **LoadProblem:** Restart with a new problem.

  - **Run:** Apply a sequence of fields until all complete.
  - **Relax:** Apply a single field and run the ODE to equilibrium.
  - **Pause:** Click anytime to stop the solver. Restart with run.
  - **Field-:** Apply the previous field again.
  - **Field+:** Apply the next field in the list.
- Output options can be changed and new output windows opened.

STEP 7: Exit OOMMF.

- On the mmSolve2D window, stop the simulation with **Pause** or **Exit**.
- Use the File menu on each window to exit.

4 OOMMF Launcher/Control Interface: mmLaunch

Full documentation not yet available. See the Quick Start example (Sec. 3) for a step by step walk through an example OOMMF session.

5 Micromagnetic Problem Editor: mmProbEd

Documentation not yet available.

6 Vector Field Display: mmDisp

Documentation not yet available.

7 The 2D Micromagnetic Solver: mmSolve2D

Documentation not yet available.

8 OOMMF Batch System

The OOMMF Batch System (OBS) consists of a command line interface to the OOMMF solver (`batchsolve.tcl`), and several Tcl scripts that provide support for complex job scheduling (`batchmaster.tcl`, `batchslave.tcl`, and `batchtask.tcl`).

Unlike the rest of the OOMMF package, the OBS is meant to be driven primarily from the command line or shell (batch) script, for example,

```
oommf.tcl <appname> [args ...]
```

where `oommf.tcl` is the OOMMF bootstrap program, and <`appname`> is an application name known to the OOMMF project. If the bootstrap program is not in your execution path, then you will have to provide the path to `oommf.tcl`. Also, `oommf.tcl` is itself a Tcl script, and so must be interpreted by a Tcl shell (typically `tclsh` or `tclsh80`). The header in `oommf.tcl` contains lines that, under Unix, automatically launch `tclsh`. The registry in Windows may be used to provide similar functionality. If the automatic Tcl shell launch mechanism is not working, you will need to supply the Tcl shell on the command line, e.g.,

```
tclsh80 <path to oommf.tcl> <appname> [args ...]
```

8.1 Batch solver

The batch solver script `batchsolve.tcl` provides a simple command line interface to the OOMMF micromagnetic solver engine.

8.1.1 Usage

```
oommf.tcl batchsolve [[-file] <filename.mif>] [-no_interface] \
     [-start_paused] [-end_paused] [-end_exit]
```

where

[**-file**] <**filename.mif**> Immediately load and run the specified MIF file. The `[-file]` tag may be omitted if the first character of `filename` is not a hyphen.

-no_interface Don't export an interactive interface. Currently this is the only network-active code inside `batchsolve.tcl`. (However, this file may be sourced from inside another script (e.g., `batchslave.tcl`) that could provide network access to the solver code without going through the interactive interface.)

-start_paused Pause solver after loading problem.

-end_paused Pause solver and enter event loop at bottom of `batchsolve.tcl` rather than just falling off the end (the effect of which will depend on whether or not Tk is loaded).

-end_exit Explicitly call exit at bottom of `batchsolve.tcl`.

The input `filename.mif` file is a standard OOMMF MIF (Micromagnetic Input Format) file, such as produced by the problem editor application (`mmpe.tcl`). The batch solver will search several directories for this file, including the current working directory, the `data` and `scripts` subdirectories, and parallel directories relative to the OOMMF `mmsolve` and `mmpe` application directories. Refer to the `mif_path` variable in `batchsolve.tcl` for the complete list.

If the **interactive interface** is active, then `batchsolve.tcl` will register itself with the OOMMF account server and will show up in the OOMMF launcher application threads list. From there you can bring the interactive interface up and connect display widgets (in the same manner as for the interactive solver application, `mmsolve`). The interface also allows you to pause, un-pause, and terminate the current simulation. If you need more interactive control, use `mmsolve` instead of `batchsolve.tcl`.

Use the `-start_paused` switch to monitor the progress of `batchsolve.tcl` from the very start of a simulation. With this switch the solver will be paused immediately after loading the specified MIF file, so you can bring up the interactive interface and connect display widgets before the simulation is begun. Start the simulation by selecting the *Run* command from the interactive interface.

The `-end_paused` switch will insure that the solver does not automatically terminate after completing the specified simulation. This is not generally useful, but may find application when `batchsolve.tcl` is called from inside a Tcl-only wrapper script.

8.1.2 Output

The output may be changed by a Tcl wrapper script (see Sec. 8.1.3), but the default behavior is to write tabular text data and the magnetization state at equilibrium for each field step. The tabular data are appended to the file *basename*`.odt`, where *basename* is specified in the input MIF file. See the routine `GetTextData` in `batchsolver.tcl` for current details, but at present the output consists of solver iteration count, nominal applied field $\mathbf{B}$, reduced magnetization $\mathbf{m}$, and total energy. This output is in the ODT file format.

The magnetization data are written to a series of OVF (OOMMF Vector Field) files, *basename*`.field`*nnnn*`.omf`, where *nnnn* starts at `0000` and is incremented at each field step. (The ascii header inside each file records the nominal applied field at that step.) These files are viewable using the `mmdisp` application.

8.1.3 Programmer's interface

In addition to directly launching `batchsolve.tcl` from the command line, it may also be sourced from inside another Tcl script, e.g., `batchslave.tcl`, that provides additional control structures. There are several variables and routines inside `batchsolve.tcl` that may be accessed and redefined from such a wrapper script to provide enhanced functionality.

Global variables

mif This is a Tcl handle to a global `mms_mif` object that is used to initialize and reset the solver by the default `batchsolver.tcl` routines.

solver Handle to the `omf_solver` object.

search_path Directory search path use by the `FindFile` proc (cf. Sec. 8.1.3).

Refer to the documentation on these objects for details on their manipulation.

Batchsolve procs The following Tcl procedures are designed for external use and/or redefinition:

SolverTaskInit Called at the start of each task.

BatchTaskRelaxCallback Called at each equilibrium state reached in the simulation.

SolverTaskCleanup Called at the conclusion of each task.

FindFile Finds a file using the directory search path specified by the global variable `search_path`. The default `SolverTaskInit` proc uses this routine to locate the requested input MIF file.

The first and third of these accept an arbitrary argument list (`args`), which is copied over from the `args` argument to `BatchTaskRun` and `BatchTaskLaunch`. Typically one copies the default procs (as needed) into a task script, and makes appropriate modifications. You may (re-) define these procs either before or after sourcing `batchsolve.tcl`. See Sec. 8.2.5 for example scripts.

8.2 Batch scheduler

The OBS provides support for complex scheduling of multiple batch jobs, via the two files `batchmaster.tcl` and `batchslave.tcl`, plus a user defined task script. The task script may be modeled after the included `simpletask.tcl` or `multtask.tcl` sample scripts.

8.2.1 Usage

```
oommf.tcl batchmaster <task_script.tcl> \
                      [server_host [server_port]]
```

where

<task_script.tcl> is the user defined task (job) definition file,

server_host specifies the network address for the master to use (default is *localhost*),

server_port is the port address for the master (default is *0*, which selects an arbitrary open port).

The required <`task_script.tcl`> should be based on the example scripts in Sec. 8.2.5. If slaves are to be run on remote machines, then `server_host` **must** be set to the local machine's network name, and the `$TaskInfo` methods `AppendSlave` and `ModifyHostList` will need to be called from inside the task script. See Sec. 8.2.4 for details.

8.2.2 Overview

The OBS has been designed to control multiple sequential and concurrent micromagnetic simulations, but the scheduling scripts `batchmaster.tcl` and `batchslave.tcl` are completely general and may be used to schedule other types of jobs as well.

When `batchmaster.tcl` is run, it sources `task_script.tcl`, which should modify the global object `$TaskInfo` to inform the master what tasks to perform and optionally how to launch slaves to perform those tasks. This is detailed in Sec. 8.2.4. After this, `batchmaster.tcl` launches all the specified slaves, initializes each with a slave initialization script, and then feeds tasks sequentially from the task list to the slaves. When a slave completes a task it reports back to the master and is given the next unclaimed task. If there are no more tasks, the slave is shut down. When all the tasks are complete, the master prints a summary of the tasks and exits.

8.2.3 Batch master and slave specifics

The communication protocol between `batchmaster.tcl` and `batchslave.tcl` is currently evolving and is not described here. Check the source code for the latest details.

The command line to start the master script is shown in Sec. 8.2.1 above. The slave will be launched by the master using instructions specified in the task script, which should take the form

```
oommf.tcl batchslave $server(host) $server(port) $slaveid \
          $passwd [aux_script [aux_args]]
```

The arguments (`$server(host)` through `$passwd`) are provided by the master and should be specified in the task script using the `%connect_info` percent substitution token. The `aux_script` is a script for the slave to source before processing any commands from the master. Typically this will be the micromagnetic batch mode script, `batchsolve.tcl`. The `aux_args` are additional arguments that will be passed to `aux_script`.

8.2.4 Batch task scripts

The master script creates an instance of a `BatchTaskObj` object with the name `$TaskInfo`. The task script uses method calls to this object to set up tasks to be performed. The only required call is to the `AppendTask` method, e.g.,

```
$TaskInfo AppendTask A "BatchTaskRun taskA.mif"
```

This method expects two arguments, a label for the task (here "A") and a script to accomplish the task. The script will be passed across a network socket from the master to the slave, and then the script will be interpreted by the slave. (In particular, keep in mind that the file system seen by the script will be that of the machine on which the slave process is running.)

This example uses the default `batchsolve.tcl` procs to run the simulation defined by the `taskA.mif` MIF file. If you want to make changes to the MIF problem specifications on the fly, you will need to modify the default procs. This is done by creating a slave initialization script, via the call

```
$TaskInfo SetSlaveInitScript { <insert script here> }
```

The slave initialization script does global initializations, and usually redefines the `SolverTaskInit` proc, and optionally the `BatchTaskRelaxCallback` and `SolverTaskCleanup` procs as well. `SolverTaskInit` is called by `BatchTaskRun` (inside `batchsolve.tcl`) at the start of each task, `BatchTaskRelaxCallback` is executed when each equilibrium is reached, and `SolverTaskCleanup` is called at the end of each task. The first and third are passed the arguments that were passed to `BatchTaskRun`. A simple `SolverTaskInit` proc could be

```
proc SolverTaskInit { args } {
   global mif basename outtextfile
   set A [lindex $args 0]
   set outbasename "$basename-A$A"
```

```
    $mif SetA $A
    $mif SetOutBaseName $outbasename
    set outtextfile [open "$outbasename.odt" "a+"]
    puts $outtextfile [GetTextData header \
  "Run on $basename.mif, with A=[$mif GetA]"]
 }
```

This proc receives the exchange constant `A` for this task on the argument list, and makes use of the global variables `mif` and `basename`. (Both should be initialized in the slave initialization script outside the `SolverTaskInit` proc.) It then stores the requested value of `A` in the `mif` object, sets up the base filename to use for output, and opens a text file to which tabular will be appended. The handle to this text file is stored in the global `outtextfile`, which is closed by the default `SolverTaskCleanup` proc. A corresponding task script could be

```
$TaskInfo AppendTask "A=13e-12 J/m" "BatchTaskRun 13e-12"
```

which runs a simulation with `A` set to 13×10^{-12} J/m. This example is taken from the `multitask.tcl` script in Sec. 8.2.5.

If you want to run more than one task at a time, then the `$TaskInfo` method `AppendSlave` will have to be invoked. This takes the form

```
$TaskInfo AppendSlave <spawn count> <spawn command>
```

where <`spawn command`> is the command to exec to launch the slave process, and <`spawn count`> is the number of slaves to launch with this command. (Typically <`spawn count`> should not be larger than the number of processors on the system.) The default value for this item (which gets overwritten with the first call to `$TaskInfo AppendSlave`) is

```
 1 {exec %oommf batchslave -notk %connect_info batchsolve.tcl}
```

This uses the OOMMF bootstrap program to launch the `batchslave` application, with connection information provided by the master, and using the auxiliary script `batchsolve.tcl`. The `batchmaster` script provides several percent-style substitutions useful in slave launch scripts: %oommf, %connect_info, %oommf_root, and %%. The first is an absolute path to the OOMMF bootstrap program on the master machine, the second is connection information needed by the `batchslave` application, the third is the path to the OOMMF root directory on the master machine, and the last is interpreted as a single percent.

To launch slaves on a remote host, use `rsh` in the spawn command, e.g.,

```
 1 {exec rsh foo oommf.tcl batchslave -notk %connect_info \
      batchsolve.tcl}
```

```
# FILE: simpletask.tcl
#
# This is a sample batch task file.  Usage example:
#
#        oommf.tcl batchmaster simpletask.tcl

# Form task list
$TaskInfo AppendTask A "BatchTaskRun taskA.mif"
$TaskInfo AppendTask B "BatchTaskRun taskB.mif"
$TaskInfo AppendTask C "BatchTaskRun taskC.mif"
```

Figure 1: Sample task script `simpletask.tcl`.

This example assumes `oommf.tcl` is in the execution path on the remote machine `foo`. In addition, you will have to add the machine `foo` to the host connect list with

```
$TaskInfo ModifyHostList +foo
```

and `batchmaster` must be run with the network interface specified as the server host (instead of the default `localhost`), e.g.,

```
oommf.tcl batchmaster multitask.tcl bar
```

where `bar` is the name of the local machine.

This may seem a bit complicated, but the examples in the next section should make things clearer.

8.2.5 Sample task scripts

The first sample task script (Fig. 1) is a simple example that runs the 3 micromagnetic simulations described by the MIF files `taskA.mif`, `taskB.mif` and `taskC.mif`. The example uses the default slave launch script, so a single slave is launched on the current machine, and the 3 simulations will be run sequentially. Also, no slave init script is given, so the default procs in `batchsolve.tcl` are used. Output will be magnetization states and tabular data for each equilibrium state, stored in files on the local machine with base names as specified in the MIF files.

The second task script (Fig. 2) is a more complicated example running concurrent processes on two machines. This script should be run with the command

```
oommf.tcl batchmaster multitask.tcl foo
```

where `foo` is the name of the local machine.

Several Tcl variables (`RMT_MACHINE` through `A_list`) are setup at the top of the `multitask.tcl` script; these are used farther down in the script. The remote machine is specified as `bar`, which is used in the `$TaskInfo AppendSlave` and `$TaskInfo ModifyHostList` commands.

There are two `AppendSlave` commands, one to run two slaves on the local machine, and one to run a single slave on the remote machine (`bar`). The latter changes to a specified working directory before launching the `batchslave` application on the remote machine. (For this to work you must have `rsh` configured properly. In the future it will be possible to launch remote commands using the OOMMF account server application, thereby lessening the reliance on system commands like `rsh`.)

After this the slave init script is defined. The Tcl `regsub` command is used to place the task script defined value of `BASEMIF` into the init script template. The init script is run on the slave when the slave is first brought up. It first reads the base MIF file into a newly created `mms_mif` instance. (The MIF file needs to be accessible by the slave process, irrespective of which machine it is running on.) Then replacement `SolverTaskInit` and `SolverTaskCleanup` procs are defined. The new `SolverTaskInit` interprets its first argument as a value for the exchange constant `A`. Note that this is different from the default `SolverTaskInit` proc, which interprets its first argument as the name of a MIF file to load. But with this task script, a MIF file is read once when the slave is brought up, and then each task redefines only the value of `A` for the simulation (and corresponding changes to the output filenames and data table header).

Finally, the Tcl loop structure

```
foreach A $A_list {
    $TaskInfo AppendTask "A=$A" "BatchTaskRun $A"
}
```

is used to build up a task list consisting of one task for each value of `A` in `A_list` (defined at the top of the task script). For example, the first value of `A` is 10e-13, so the first task will have the label `A=10e-13` and the corresponding script is `BatchTaskRun 10e-13`. The value 10e-13 is passed on by `BatchTaskRun` to the `SolverTaskInit` proc, which has been redefined to process this argument as the value for `A`, as described above.

There are 6 tasks in all, and 3 slave processes, so the first three tasks will run concurrently in the 3 slaves. As each slave finishes it will be given the next task, until all the tasks are complete.

```
# FILE: multitask.tcl
#
# This is a sample batch task file.  Usage example:
```

```
#
#        oommf.tcl batchmaster multitask.tcl hostname [port]
#
# Task script configuration
set RMT_MACHINE    bar
set RMT_OOMMF      "~/mag/oommf/app/oommf/oommf.tcl"
set RMT_WORK_DIR   "~/mag/oommf/app/mmsolve/data"
set BASEMIF taskA
set A_list { 10e-13 10e-14 10e-15 10e-16 10e-17 10e-18 }

# Slave launch commands
$TaskInfo ModifyHostList +$RMT_MACHINE
$TaskInfo AppendSlave 1 "exec %oommf batchslave -notk \
        %connect_info batchsolve.tcl"
$TaskInfo AppendSlave 1 "exec rsh $RMT_MACHINE \
        cd $RMT_WORK_DIR \\\;\
        $RMT_OOMMF batchslave -notk %connect_info batchsolve.tcl"

# Slave initialization script (with batchsolve.tcl proc
# redefinitions)
set init_script {
    # Initialize solver. This is run at global scope
    set basename __BASEMIF__      ;# Base mif filename (global)
    mms_mif New mif
    $mif Read [FindFile ${basename}.mif]
    # Redefine TaskInit and TaskCleanup proc's
    proc SolverTaskInit { args } {
        global mif outtextfile basename
        set A [lindex $args 0]
        set outbasename "$basename-A$A"
        $mif SetA $A
        $mif SetOutBaseName $outbasename
        set outtextfile [open "$outbasename.odt" "a+"]
puts $outtextfile [GetTextData header \
"Run on $basename.mif, with A=[$mif GetA]"]
        flush $outtextfile
    }
    proc SolverTaskCleanup { args } {
```

```
        global outtextfile
        close $outtextfile
    }
}
# Substitute $BASEMIF in for __BASEMIF__ in above script
regsub -all -- __BASEMIF__ $init_script $BASEMIF init_script
$TaskInfo SetSlaveInitScript $init_script

# Create task list
foreach A $A_list {
    $TaskInfo AppendTask "A=$A" "BatchTaskRun $A"
}
```

Figure 2: Advanced sample task script `multitask.tcl`.

9 File Formats

9.1 Problem specification format (MIF)

Micromagnetic simulations are specified to the OOMMF solver using files conforming to the OOMMF *Micromagnetic Input Format* (MIF). This is also the native file type for the OOMMF problem editor, `mmProbEd` (Sec. 5). All values in this file are in SI units.

A sample MIF file is presented in Fig. 3. The first line of a MIF file must be of the form "`#` MIF x.y", where x.y represents the format revision number. After this, any line ending in a backslash, '\', is joined to the succeeding line before any other processing is performed. Lines beginning with a '`#`' character are comments and are ignored. Blank lines are also ignored.

All other lines must consist of a *Record Identifier* followed by a parameter list. The Record Identifier is separated from the parameter list by one or more ':' and/or '=' characters. Whitespace and case is ignored in the Record Identifier field.

The parameter list must be a proper Tcl list. The parameters are parsed (broken into separate elements) following normal Tcl rules; in short, items are separated by whitespace, except as grouped by double quotes and curly braces. The grouping characters are removed during parsing. Any '`#`' character that is found outside of any grouping mechanism is interpreted as a comment start character. The '`#`' and all following characters on that line are interpreted as a comment.

Order of the records in a MIF file is unimportant, except as explicitly stated below. If two or more lines contain the same Record Identifier, then the last one takes precedence (except for Field Range records, of which there may be several active). All records are required unless listed as optional. Some of these record types are not yet available from `mmProbEd`, however your may edit a MIF file by hand and feed it to `mmSolve2D` with the `filesource.tcl` application.

For convenience, the Record Identifier tags are organized into several groups; these groups correspond to the buttons presented by `mmProbEd`. We follow this convention below.

Material parameters

- **#Material Name:** This is a convenience entry in `mmProbEd`; inside the MIF file it is a comment line. It relates a symbolic name (e.g., Iron) to specific values to the next 4 items. Ignored by `mmSolve` applications.
- **Ms:** Saturation magnetization in A/m.
- **A:** Exchange stiffness in J/m.

- **K1:** Crystalline anisotropy constant in J/m^3.
- **Anisotropy Type:** Crystalline anisotropy type; One of < uniaxial | cubic >.
- **Anisotropy Dir1:** Directional cosines of first crystalline anisotropy axis, taken with respect to the coordinate axes (3 numbers). Optional; Default is 1 0 0.
- **Anisotropy Dir2:** Directional cosines of second crystalline anisotropy axis, taken with respect to the coordinate axes (3 numbers). Optional; Default is 0 1 0.

  For uniaxial materials it suffices to specify only Anisotropy Dir1. For cubic materials one should also specify Anisotropy Dir2; the third axis direction will be calculated as the cross product of the first two. The anisotropy directions will be automatically normalized if necessary, so for example 1 1 1 is valid input (it will be modified to .5774 .5774 .5774). For cubic materials, Dir2 will be adjusted to be perpendicular to Dir1 (by subtracting out the component parallel to Dir1).
- **Anisotropy Initialization:** Method to use to set up directions of anisotropy axes, as a function of spatial location; This is a generalization of the Anisotropy Dir1/2 records. The value for this record should be one of < Constant | UniformXZ | UniformS2 >. Constant uses the values specified for Anisotropy Dir1 and Dir2, with no dispersion. UniformXY ignores the values given for Anisotropy Dir1 and Dir2, and randomly varies the anisotropy directions uniformly in the xy-plane. UniformS2 is similar, but randomly varies the anisotropy directions uniformly on the unit sphere (S^2). This record is optional; the default value is Constant.

Demag specification

- **Demag Type:** Specify algorithm used to calculate self-magnetostatic (demagnetization) field. Must be one of
  - **3dSlab:** Calculate the in-plane field components using offset blocks of constant (volume) charge. Fields components parallel to the z-axis are calculated using squares of constant (surface) charge on the upper and lower surfaces of the sample.
  - **3dCharge:** Calculate the in-plane field component using rectangles of constant (surface) charge on each cell. The z-components of the field are calculated in the same manner as for the 3dSlab approach.
  - **FastPipe:** Algorithm suitable for simulations that have infinite extent in the z-direction. This is a 2D version of the 3dSlab algorithm.

– **None:** No demagnetization. Fastest but least accurate method.

The 3dSlab and 3dCharge methods require the Part Thickness (cf. the Part Geometry section) be set. All algorithms use Fast Fourier Transform (FFT) techniques to accelerate the calculations.

Part geometry

- **Part Width:** Nominal part width (x-dimension) in meters. Will be adjusted to an integral multiple of Cell Size.
- **Part Height:** Nominal part height (y-dimension) in meters. Will be adjusted to an integral multiple of Cell Size.
- **Part Thickness:** Part thickness (z-dimension) in meters. Required for 3d models.
- **Cell Size:** In-plane (xy-plane) dimension of base calculation cell. This cell is a rectangular brick, with square cross-section and thickness given by Part Thickness.
- **Part Shape:** Optional. In-plane part shape, which must be one of the following:
  - **Rectangle:** The sample completely fills the area specified by Part Width and Part Height. (Default.)
  - **Ellipse:** The sample (or the magnetically active portion thereof) is an ellipse inscribed into the rectangular area specified by Part Width and Part Height.
  - **Oval:** Shape is a rounded rectangle, where each corner is replaced by a quarter circle with radius r, where r is given by a required second parameter.
  - **Mask:** Shape is determined by a 2-color bitmap file, the name of which is specified as a second parameter. The given filename must be accessible to the `mmSolve` application. At present the bitmap file must be a PPM (portable pixmap format) file, but it wouldn't be too difficult to add GIF support. The pixels in the file must map one-to-one to the cells in the simulation. The magnetically active cells are the ones which are black in the bitmap.

Initial magnetization

- **Init Mag:** Name of routine to use initial the simulation magnetization directions (as a function of position), and routine parameters, if any. Optional, with default Random. The list of routines is long, and it is easy to add new ones. See the file `maginit.cc` for details. A few of the more useful routines are:

- **Random**: Random directions on the unit sphere. This is somewhat like a quenched thermal demagnetized state.
- **Cant**: Uniform magnetization in the direction indicated by the two additional parameters, θ and ϕ, where the first is the angle from the z-axis (in degrees), and the second is the angle from the x-axis (in degrees) of the projection onto the xy-plane.
- **Vortex**: Fits an idealized vortex about the center of the sample.
- **vioFile**: One additional parameter required, specifying the name of a VIO file to use to initialize the magnetization. The grid in the input file will be scaled as necessary to fit the grid in the current simulation. The file must be accessible to the intended `mmSolve` application. This Init Mag type should eventually be replaced with an omfFile Init Mag type.

Experiment parameters

- **Field Range Count:** Number of Field Ranges specified (integer). Optional, with default value 1. If used, this record should precede all Field Range records.
- **Field Range:** Specifies a range of applied fields that are stepped though in a linear manner. The parameter value is a list of 7 numbers, the starting field Bx By Bz in Tesla, the stopping field Bx By Bz in Tesla, and an integer number of steps to take between the starting and stopping fields (inclusive). There should be as many of these records as specified by the Field Range Count record. They will be stepped through in order of appearance. This record is optional, with a default value of 0 0 0 0 0 0 1.
- **Field Type:** External field routine and parameters, if any. This is optional, with default Uniform. At most one record of this type is allowed, but the Multi type may be used to apply a collection of fields. The nominal applied field (NAF) is stepped through the Field Ranges as describe above, and made available to the external field routines which use or ignore it as appropriate.

  The following routines are available:

  - **Uniform**
    Applied field is uniform with value specified by the NAF.
  - **Ribbon relcharge x0 y0 x1 y1 height**
    Charge "Ribbon," lying perpendicular to the xy-plane. Here relcharge is the charge strength relative to Ms, and (x0,z0), (x1,z1) are the endpoints of the

ribbon (in meters). The ribbon extends height/2 above and below the calculation plane. This routine ignores the NAF.

- **Tie rfx rfy rfz x0 y0 x1 y1 ribwidth**
  The points (x0,y0) and (x1,y1) define (in meters) the endpoints of the center spine of a rectangular ribbon of width ribwidth lying in the xy-plane. The cells with sample point inside this rectangle see an applied field of (rf0,rf1,rf2), in units relative to Ms. (If the field is large, then the magnetizations in the rectangle will be "tied" to the direction of that field.)
- **File filename multiplier**
  Field read in from a file. The "multiplier" field is optional, defaulting to 1. The input file may be any of the vector field file types recognized by mmDisp. The input dimensions will be scaled as necessary to exactly match the simulation grid, with zeroth order interpolation as necessary. This routine ignores the NAF.
- **Multi routinecount **
  **param1count name1 param1 param2 ... **
  **param2count name2 param1 param2 ... **
  ...
  Allows a conglomeration of several field type routines. All entries must be on the same logical line, i.e., end physical lines with '\' continuation characters as necessary. Here routinecount is the number of routines, and param1count is the number parameters (including name1) needed by the first routine, etc.

All dimensions are in meters. The coordinates in the simulation run from (0,0,0) to (Part Width, Part Height, Part Thickness).

Output specification

- **Base Output Filename:** Default base name used to construct output filenames.

Miscellaneous

- **Converge Torque Value:** Nominal value to use as a stopping criterion: When the maximum torque seen by any spin in the simulation is smaller than this value, it is assumed that a relaxed state has been reached for the current applied field. This is a dimensionless value, relative to Ms. Optional; default value is 10^{-4}.
- **Randomizer seed:** Value with which to seed random number generator. Optional. Default value is 0, which uses the system clock to generate a semi-random seed.

```
# MIF 1.1
#
# All units are SI.
#
######################## MATERIAL PARAMETERS ##############################
Ms:  800e3                 # Saturation magnetization in A/m.
A:   13e-12                # Exchange stiffness in J/m.
K1:  0.5e3                 # Anisotropy constant in J/m^3.
Anisotropy Type: uniaxial  # One of <uniaxial|cubic>.
Anisotropy Dir1: 1 0 0     # Directional cosines wrt to coordinate axes

######################## DEMAG SPECIFICATION ##############################
Demag Type: 3dSlab   # One of <3dSlab|2dSlab|3dCharge|FastPipe|None>.

########################## PART GEOMETRY ##################################
Part Width:     0.25e-6    # Nominal part width in m
Part Height:    1.0e-6     # Nominal part height in m
Part Thickness: 1e-9       # Part thickness in m.
Cell Size:      8.1e-9     # Cell size in m.
#Part Shape:    # One of <Rectangle|Ellipse|Oval|Mask>.  Optional.

####################### INITIAL MAGNETIZATION #############################
Init Mag: Cant 90 45 # Initial magnetization routine and parameters

####################### EXPERIMENT PARAMETERS #############################
Field Range Count: 2  # Number of Field Range lines.
Field Range: -.05 -.01 0. .05 .01 0. 100 # Start_field Stop_field Steps
Field Range: .05 .01 0. -.05 -.01 0. 100
Field Type: Multi 4 \
 7 Ribbon 1 0 1.0e-6 0.25e-6 1.0e-6 1e-9 \
 7 Ribbon 1 0 0     0.25e-6 0     1e-9 \
 9 Tie 100 0 0 0.12e-6 0.5e-6 0.13e-6 0.5e-6 8.1e-9
 1 Uniform \
# The above positions ribbons of positive charge along the upper
# and lower edges with stength Ms, applies a large (100 Ms) field
# to the center cell, and also applies a uniform field across the
# sample stepped from (-.05,-.01,0.) to (.05,.01,0.) (Tesla), and
# back, in approximately 0.001 T steps.
```

```
###################### OUTPUT SPECIFICATIONS ############################
Base Output Filename: samplerun

########################## MISCELLANEOUS ##############################
Randomizer seed: 1   # Value to seed random number generator with.
Converge Torque Value: 1e-5  # Stopping criterion, relative to Ms.
```

Figure 3: Example MIF file.

9.2 Data table format (ODT)

Textual output from `mmSolve` applications that is not of the vector field variety is output in the *OOMMF Data Table* (ODT) format. This is an ASCII file format, with column information in the header and one line of data per record. Any line ending in a '\' character is joined to the succeeding line before any other processing is performed. Any leading '#' characters on the second line are removed.

As with the OVF format (Sec. 9.3.1), all non-data lines begin with a '#' character, comments with two '#' characters. (This makes it easier to import the data into external programs, for example, plotting packages.) An example is shown in Fig. 4.

The first line should be the file type descriptor

```
# ODT 1.0
```

This is followed by comments and any of the following 5 recognized descriptor tag lines:

- **Table Start:** Optional, used to segment a file containing multiple data table blocks. Anything after the colon is taken as an optional lable for the corresponding data table block.
- **Title:** Optional; everything after the colon is interpreted as a title for the table.
- **Columns:** Required. One parameter per column, designating the label for that column. Spaces may be embedded in a column label by using the normal Tcl grouping mechanisms (i.e., double-quotes and braces).
- **Units:** Optional. If given, it should have one parameter for each column, giving a unit label for the corresponding column.

```
# ODT 1.0
# Table Start
# Title: This is a small sample ODT file.
#
## This is a sample comment.  You can put anything you want
## on comment lines.
#
# Columns: Iteration "Applied Field"  {Total Energy}    Mx
# Units:      {}          "mT"           "J/m^3"       "A/m"
              103          50            0.00636      787840
             1000          32            0.00603      781120
            10300       -5000            0.00640     -800e3
# Table End
```

Figure 4: Sample ODT file.

- **Table End**: Optional, no parameters. Should be used in conjuction with a corresponding Table Start record.

Data may appear anywhere after its Columns descriptor record and before any Table End line, with one record per line. The data should be numeric values separated by whitespace.

9.3 Vector field formats (OVF and SVF)

Vector field files specify vector quantities (e.g., magnetization or magnetic flux density) as a function of spatial position. This type of file is produced by the micromagnetic solver program (Section 7) when "Total Field" or "Magnetization" output is selected. It is also the input data type read by the vector field display program, mmDisp (Section 6). OOMMF currently supports two vector field formats: the *OOMMF Vector Field* (OVF) format and the *Simple Vector Field* (SVF) format. The OVF format is the preferred format used by all OOMMF code. It supports both rectangular and irregular meshes, in binary and ASCII. The SVF format is provided to supply a simple interface to outside programs. (A third format, the *Vector Input/Output* (VIO) format, is used by some precursors to the OOMMF code. Its use is deprecated, and should not be used for new code.)

The recommended file extensions for OVF files are .omf for magnetization files, .ohf for magnetic field (**H**) files, .obf for magnetic flux density (**B**) files, or .ovf for generic files. SVF files should be given a .svf extension.

9.3.1 The OVF format

A commented sample OVF file is provided in Fig. 5. These files have an ASCII header and trailer, and a data block that may be either ASCII or binary. All non-data lines begin with a '`#`' character; the double '`##`' marks the start of a comment, which continues until the end of the line. There is no line continuation character. Lines starting with a '`#`' but containing only whitespace characters are ignored.

All non-empty non-comment lines in the file header are structured as field+value pairs. The field tag consists of all characters after the initial '`#`' up to the first colon (':') character. Case is ignored, and all space and tab characters are eliminated. The value consists of all characters after the first colon, continuing up to a '`##`' comment designator or the end of the line.

The first line of a OVF file should be a file type identification line, having the form

```
# OOMMF: rectangular mesh v1.00
```

where the value "rectangular mesh v1.00" identifies subformat type and revision. Only the field tag "OOMMF" is used at present.

After this the file is conceptually broken into Segment blocks, and each Segment block is composed of a (Segment) Header block and a Data block. Every block begins with "`#` Begin: <block type>" line, and ends with a corresponding "`#` End: <block type>" line. The number of Segment blocks is specified in the

```
# Segment count: 1
```

line. Currently only 1 segment is allowed. This may be changed in the future to allow for multiple images per file. This is followed by

```
# Begin: Segment
```

to start the first segment.

Segment Header block The Segment Header block start is marked by the line "`#` Begin: Header" and the end by "`#` End: Header". Everything between these lines should be either comments or one of the following file descriptor lines. They are order independent. All are required unless otherwise stated. Numeric values are floating point values unless "integer" is explicitly stated.

- **title:** Long file name or title.

- **desc:** Description line. Use as many as desired. Description lines may be displayed by postprocessing programs, unlike comment lines which are ignored by all automated processing.

- **meshunit:** Fundamental mesh spatial unit, treated as a label. The comment marker '`##`' is not allowed in this label. Example value: "nm".

- **valueunit:** Fundamental field value unit, treated as a label. The comment marker '`##`' is not allowed in this label. Example: "kA/m."

- **valuemultiplier:** File data values are multiplied by this to get true values in units of "valueunit." This simplifies the use of normalized values in the data block.

- **xmin, ymin, zmin, xmax, ymax, zmax:** Six separate lines, specifying the bounding box for the mesh, in units of "meshunit." This may be used by display programs to limit the display area, and may be used for drawing a boundary frame if "boundary" is not specified.

- **boundary:** List of *(x,y,z)* triples specifying the vertices of a boundary frame.

- **ValueRangeMaxMag, ValueRangeMinMag:** The maximum and non-zero minimum field magnitudes in the data block, in the same units as used in the data block. These are for optional use as hints by postprocessing programs; for example, mmDisp will not display any vector with magnitude smaller than ValueRangeMinMag.

- **meshtype:** Grid structure; should be either "rectangular" or "irregular." Irregular grid files should specify "pointcount" in the header; rectangular grid files should specify instead "xbase, ybase, zbase," "xstepsize, ystepsize, zstepsize," and "xnodes, ynodes, znodes."

- **pointcount:** Number of data sample points/locations, i.e., nodes (integer). For irregular grids only.

- **xbase, ybase, zbase:** Three separate lines, denoting the position of the first point in the data section, in units of "meshunit." For rectangular grids only.

- **xstepsize, ystepsize, zstepsize:** Three separate lines, specifying the distance between adjacent grid points, in units of "meshunit." For rectangular grids only.

- **xnodes, ynodes, znodes:** Three separate lines, specifying the number of nodes along each axis (integers only). For rectangular grids only.

Data block The data block start is marked by a line of the form

```
# Begin: data <representation>
```

where <representation> is one of "text", "binary 4", or "binary 8". Text mode uses the ASCII specification, with individual data items separated by an arbitrary amount of whitespace (spaces, tabs and newlines). Comments are not allowed inside binary mode data blocks, but are permitted inside text data blocks.

The binary representations are IEEE floating point in network byte order (MSB). To insure that the byte order is correct, and to provide a partial check that the file hasn't been sent through a non 8-bit clean channel, the first datum is a predefined value: 1234567.0 (Hex: 49 96 B4 38) for 4-byte mode, and 123456789012345.0 (Hex: 42 DC 12 21 83 77 DE 40) for 8-byte mode. The data immediately follow the check value.

The structure of the data depends on whether the "meshtype" declared in the header is "irregular" or "rectangular". For irregular meshes, each data element is a 6-tuple, consisting of the x, y and z components of the node position, followed by the x, y and z components of the field at that position. Ordering among the nodes is not relevant. The number of nodes is specified in the "pointcount" line in the segment header.

For rectangular meshes, data input is field values only, in x, y, z component triples. These are ordered with the x index incremented first, then the y index, and the z index last. This is nominally Fortran order, and is adopted here because commonly x will be the longest dimension, and z the shortest, so this order is more memory-access efficient than the normal C array indexing of z, y, x. The size of each dimension is specified in the "xnodes, ynodes, znodes" lines in the segment header.

In any case, the first character after the last data item should be a newline, followed by

```
# End: data <representation>
```

where <representation> must match the value in the "Begin: data" line. This is followed by a

```
# End: segment
```

line that ends the segment, and hence the file.

```
# OOMMF: rectangular mesh v1.00
#
## This is a comment.
## No comments allowed in the first line.
#
```

```
# Segment count: 1   ## Number of segments.  Should be 1 for now.
#
# Begin: Segment
# Begin: Header
#
# Title: Long file name or title goes here
#
# Desc: 'Description' tag, which may be used or ignored by postprocessing
# Desc: programs. You can put anything you want here, and can have as many
# Desc: 'Desc' lines as you want.  The ## comment marker is disabled in
# Desc: description lines.
#
# meshunit: nm   ## Fundamental mesh measurement unit.  Treated as a label
#
# meshtype: rectangular
# xbase: 0.      ## (xbase,ybase,zbase) is the position, in
# ybase: 0.      ## 'meshunit', of the first point in the data
# zbase: 0.      ## section (below).
#
# xstepsize: 20. ## Distance between adjacent grid pts.: on the x-axis,
# ystepsize: 10. ## 20 nm, etc.  The sign on this value determines the
# zstepsize: 10. ## grid orientation relative to (xbase,ybase,zbase).
#
# xnodes: 200    ## Number of nodes along the x-axis, etc. (integers)
# ynodes: 400
# znodes:   1
#
# xmin:    0.    ## Corner points defining mesh bounding box in
# ymin:    0.    ## 'meshunit'.  Floating point values.
# zmin:  -10.
# xmax: 4000.
# ymax: 4000.
# zmax:   10.
#
# valueunit: kA/m  ## Fundamental field value unit, treated as a label
# valuemultiplier: 0.79577472  ## Multiply file values by this to get
#                              ## true value in 'valueunits'.
#
```

```
# ValueRangeMaxMag:  1005.3096  ## These are in file value units, and
# ValueRangeMinMag:  1e-8       ## are used as hints (or defaults) by
#     ## postprocessing programs.  The mmdisp program ignores any points
#     ## with magnitude smaller than ValueRangeMinMag, and uses
#     ## ValueRangeMaxMag to scale inputs for display.
#
# End: Header
#
## Anything between '# End: Header' and '# Begin: data text',
## '# Begin: data binary 4' or '# Begin: data binary 8' is ignored.
##
## Data input is in 'x-component y-component z-component' triples,
## ordered with x incremented first, then y, and finally z.
#
# Begin: data text
1000 0 0 724.1 0. 700.023
578.5 500.4 -652.36
<...data omitted for brevity...>
252.34 -696.42 -671.81
# End: data text
# End: segment
```

Figure 5: Commented OVF sample file.

9.3.2 The SVF format

The Simple Vector Field (SVF) format is a simple ASCII format supporting irregularly sampled data. It is intended as an aid for importing data from non-OOMMF programs, and is backwards compatible with the format used for problem submissions for the first μMag standard problem[7].

A sample SVF file is shown in Fig. 6. Any line beginning with a '#' character is a comment, all others are data lines. Each data line is a whitespace separated list of 6 elements: the x, y and z components of a node position, followed by the x, y and z components of the field at that position. Input continues until the end of the file is reached.

It is recommended (but not required) that the first line of an SVF file be

[7] http://www.ctcms.nist.gov/~rdm/stdprob_1.html

```
# SVF-02
## File: sample.svf
## Boundary-XY: 0.0 0.0 1.0 0.0 1.0 2.0 0.0 2.0 0.0 0.0
## Grid step: .25 .5 0
#  x      y      z        m_x      m_y      m_z
  0.01   0.01   0.01   -0.35537  0.93472 -0.00000
  0.01   1.00   0.01   -0.18936  0.98191 -0.00000
  0.01   1.99   0.01   -0.08112  0.99670 -0.00000
  0.50   0.50   0.01   -0.03302  0.99945 -0.00001
  0.99   0.05   0.01   -0.08141  0.99668 -0.00001
  0.75   1.50   0.01   -0.18981  0.98182 -0.00000
  0.99   1.99   0.01   -0.35652  0.93429 -0.00000
```

Figure 6: Example SVF file.

```
# SVF-02
```

This will aid automatic file type detection. Also, three special (extended) comments are recognized by the mmDisp display program:

```
## File: <filename or extended filename>
## Boundary-XY: <boundary vertex pairs>
## Grid step: <cell dimension triple>
```

All these lines are optional. The "File" provides a preferred (possibly extended) filename to use for display identification. The "Boundary-XY" line specifies the ordered vertices of a bounding polygon in the xy-plane. If given, mmDisp will draw a frame using those points to ostensibly indicate the edges of the simulation body. Lastly, the "Grid step" line provides three values representing the average x, y and z dimensions of the volume corresponding to an individual node (field sample). It is used by mmDisp to help scale the display.

10 Credits

The main contributors to this document are Michael J. Donahue (michael.donahue@nist.gov) and Donald G. Porter (donald.porter@nist.gov), both of ITL/NIST. Section 3 is based on notes from Dianne P. O'Leary (oleary@cs.umd.edu).

The OOMMF[8] code is being developed mainly by Michael Donahue and Donald Porter. Robert D. McMichael (rmcmichael@nist.gov) made contributions to the early development of the 2D micromagnetic solver. Jason Eicke (jeicke@seas.gwu.edu) is responsible for the problem editor, and has worked on the self-magnetostatic module of the micromagnetic solver.

[8] http://math.nist.gov/oommf/