Rocface Users Guide (Version 3.0)

Xiangmin Jiao

Revised: 04/15/2005

1 Overview

Rocface is a service utility for transferring data between surface meshes in a fashion that is numerically accurate and physically conservative. Rocface transfers data in two steps. First, it constructs a common refinement of two surface meshes using a sophisticated algorithm described in Jiao's thesis. This step is done sequentially and is typically done off-line. Second, it transfers data using a least squares formulation that minimizes errors in the L_2 norm. The user also has the option of using a simple interpolation scheme for the second step, but it will not be conservative.

Rocface supports both node-centered and face-centered data, and can transfer between these two types of data in any combination. It supports both multi-block structured meshes and unstructured surface meshes with mixed elements. For unstructured meshes, both first-order (3-node triangles and 4-node quadrilaterals) and second-order (6-node triangles and 8- or 9-node quadrilaterals) elements are supported.

Rocface was implemented in C++ using Roccom's developers interface. Interprocessor communication is done through MPI.

2 Rocface API

Rocface should typically be called through Roccom by an orchestration module (such as Rocman in Rocstar3 of CSAR). An application does not interact directly with Rocface, except that they must register windows with Roccom. Note that Rocface can only take nodal and elemental data attributes with contiguous layouts (i.e., no staggered layout). See Roccom Users Guide for how to create windows. In this section, we will describe only the function interface to be called from an orchestration module.

Rocface provides subroutines Rocface_load_module and Rocface_unload_module. Each takes a window name as an input. Typically, the name is "RFC". These two routines are the only ones that are not called through COM_call_function. All the interface functions of Rocface are registered with Roccom as a member function of a Rocface object, which encapsulates all the internal context of Rocface. In the following, we omit the Rocface object in the argument list.

2.1 Overlaying Meshes

2.1.1 Construction of Overlay

Because the overlay algorithm is sequential, the construction of the overlay is typically done off-line using the surfdiver utility described in Section 4. In a sequential run, a user may want to construct the overlay on-line using the following interface.

overlay(const COM::Attribute *mesh1, const COM::Attribute *mesh2, const MPI_Comm *comm=NULL, const char *path=NULL, const double *tol1=NULL, const double *tol2=NULL)

The first two arguments are two Roccom objects referring to the mesh data of two windows. Only these two arguments are mandatory and are sufficient for most cases. The third argument comm is an MPI communicator, which should be the same as the communicator on which the two input windows are distributed. The argument path is the path where the output files should go. The remaining two arguments control the tolerancing schemes used in the overlay algorithm, and they should be left blank.

A user can clean up the memory allocated for an overlay using the following interface.

clean_overlay(const char *win1, const char *win2)

It takes the window names of the two meshes as input.

2.1.2 I/O of Overlay

After constructing the overlay, a user can write it out into binary files using the interface write_overlay_sdv.

write_overlay_sdv(const COM::Attribute *mesh1, const COM::Attribute *mesh2, const char *prefix1=NULL, const char *prefix2=NULL)

The first two arguments are two Roccom objects referring to the mesh data of two windows. Only these two arguments are mandatory and are sufficient for most cases. The third and fourth arguments are the prefixes that should be used for the output files. The default are the window names.

The overlay files can be read in using the following interface.

read_overlay_sdv(const COM::Attribute *mesh1, const COM::Attribute *mesh2, const MPI_Comm *comm, const char *prefix1=NULL, const char *prefix2=NULL)

It takes an additional argument comm before the prefixes. This routine need to be called in parallel simulations, and comm should be the communicator on which the windows are defined.

2.2 Data Transfer

After constructing an overlay or loading it in from files, the user can invoke data transfer for two attributes using the interface least_squares_transfer.

least_squares_transfer(const COM::Attribute *att1, COM::Attribute *att2, const int *ord=NULL, double *tol=NULL, int *iter=NULL, const int *v=NULL)

Again, the first two arguments correspond to the source and target attributes, respectively, and only these two are mandatory. The remaining arguments typically should not be used. The third argument specifies what order of accuracy Roccom should use for quadrature rules, and the default is 2. The tol and iter arguments specify the convergence criteria of the linear solver that Rocface uses. The last argument specifies the verbose level, and the default is zero.

If a user would like to use interpolation for speed, the following interface can be used instead, whose argument list is a subset of that of least_squares_transfer.

interpolate(const COM::Attribute *att1, COM::Attribute *att2, const int *v=NULL)

3 Compiling Rocface

Rocface can be compiled using only the C++ compilers that reasonably conform to the ISO/IEC C++ standard with supports for namespace, template, and STL. Rocface is known to compile with g++-2.91 or later on various platforms (including Linux, Sun, SGI Origin 2000, IBM SP/2), SGI CC-7.30 and KAI CC-4.0 on SGI Origin 2000, and the native C++ compiler (xlC) on IBM SP.

The makefile of Rocface includes the common makefile of Roccom for implicit rules and automatic generation of dependencies. It requires GNU Make (gmake). On the supported platforms, including Origin 2000 (e.g., modi4.ncsa), IBM SP (e.g., Blue Horizon), and Linux workstations (e.g., Turing), a user can compile Rocface by simply run "gmake" in the root directory of Rocface. To enable debugging, pass " DEBUG=1" to gmake as command line option. You can also choose to use the built-in dummy MPI in Roccom passing "DUMMY_MPI=1" to gmake.

4 Surfdiver

Surfdiver is a preprocessor of Rocface for generating a common refinement of two surface meshes for Rocstar. It takes two ASCII interface meshes files as input and generates binary overlay files for Rocface. These binary outputs are compatible across all platforms that use either big- or small-endian. Therefore, you can run surfdiver on any of your favorite platform and then take the output to the machine to run Rocstar.

4.1 Compilation

Surfdiver uses Roccom and Rocface, so you must compile Roccom and Rocface before you can build Surfdiver. Because surfdiver is serial, we recommend compile it using the built-in dummy MPI in Roccom. If you have clean source directories of Roccom and Rocface, you can build surfdiver in the following two steps:

1. In Roccom's root directory, build Roccom using command "gmake DUMMY_MPI=1".
2. In Rocface's root directory, invoke the command "gmake DUMMY_MPI=1 surfdiver", which will build the Rocface library and surfdiver.

4.2 Preprocessing for Rocstar3

In Rocstar3, each fluid and solid module provides its own utility to generate an interface mesh file for surfdiver. These files have the suffix ".im". The file format is given in Appendix **??**.

To run surfdiver, copy the interface mesh file of fluids to "fluid.im", copy that of solids to "solid.im", and then run "surfdiver fluid solid" (or "surfdiver solid fluid"; if one fails, try the other). This will generate one ".sdv" file for each fluid or solid pane.

Note that if you built surfdiver without the "DUMMY_MPI=1" option, you may need to use mpirun when running surfdiver.

5 Advanced Tuning for Feature Detection

Rocface automatically detects the corners and ridges of a surface mesh. For most models, the default parameters set by Rocface would work. For some complex models, one can control the parameters of feature detection by providing a control file <window name>.fc. This file should have five lines:

1. The first line contains four parameters for face angle: cosine of the upper bound, cosine of the lower bound, the signal-to-noise ratio, and cosine of an open-end of a 1-feature.

2. The second line contains three parameters for angle defect: upper bound, lower bound, and signal-to-noise ratio.

3. The third line contains three parameters for the turning angle: cosine of the upper bound, cosine of the lower bound, and the signal-to-noise ratio.

4. The fourth line contains four parameters controling the filteration rules: the minimum edge length for open-ended 1-features, whether to apply the long-falseness filteration rule, whether to apply the strong-end filtration rule, and whether to snap 1-features of input meshes on top of each other.

5. The fifth line controls the verbosity level. The default value is one. Setting vebosity level to greater than one will instruct Rocface to write out HDF files "*_fea.hdf", which contain the feature information and are very helpful for fine tuning feature detection.

The following is a sample control file containing the default values.

```
0.76604444 0.98480775  3 0.98480775 # Feature angles
1.3962634  0.314159265 3            # Angle defects
0.17364818 0.96592583  3            # Turning angles
6 1 0 0                            # Filteration rules
2                                  # Verbosity level
```

If the control file is missing, then the default values (as shown above) will be used. These default values should work for most cases. For some other cases, it typically suffices to adjust the signal-to-noise ratios (the third parameters of the first three lines) and the minimum edge length (first parameter of line 4) should suffice.

5.1 RSRM Dataset

The RSRM dataset has many fine details and some weak-ended features, and hence requires tuning the feature-detection parameters. As of April 15, 2005, the following control parameters are known to work for both fluids and solids meshes of the RSRM dataset:

```
0.5        0.6         3   0.17365   # Feature angles
1.3962634  0.314159265 3             # Angle defects
0.         0.5         3             # Turning angles
6 1 1 0                              # Filteration rules
1                                    # Verbosity level
```

To run surfdiver for Rocstar3, create a file Rocman/RocfluRocfrac/ifluid.fc and a file Rocman/RocfluRocfrac/isolid.fc containing these lines, and then invoke

surfdiver Rocflu/Rocin/ifluid_in_00.000000.txt Rocfrac/Rocin/isolid_in_00.000000.txt Rocman/RocfluRocfrac/

5.2 Fine-Tuning Parameters

If it ever becomes necessary to fine-tune the feature-detection parameters, adjusting only the parameters of feature angles (i.e., the first line of the .fc files) typically suffices. The following procedure is useful in determining the proper values of feature angles:

1. Find the ifluid_fea*.hdf and isolid_fea*.hdf files in the output directory. These files are generated by surfdiver if the ".fc" files are present and the verbosity level in these files are greater than 1.
2. Convert these HDF files into Tecplot files using the utility hdf2plt. Typically, the commands look like (note that the quotation marks are important):
   (a) hdf2plt "ifluid_fea*.hdf" ifluid.plt
   (b) hdf2plt "isolid_fea*.hdf" isolid.plt
3. Load the ifluid.plt and isolid.plt into two separate Tecplot sessions. Look at the contour of "frank" (stands for feature rank) to eyeball the discrepancies of the detected features in the input meshes.
4. Use the "probe" tool of Tecplot to look at the values of "fangle" (stands for feature angles), and adjust the feature-angle thresholds based on the values "fangle" of false features. Typically, if some edges are marked as features in correctly, then one should increase the feature-angle thresholds; if some feature edges are missing, then one should decrease the feature-angle thresholds.

6 Test Problems

Rocface includes a series of test problems in the directory test. These problems are meant to test the robustness of the mesh overlay algorithm. You can run all the cases at once by invoking "gmake testall".