Center for Simulation of Advanced Rockets

University of Illinois at Urbana-Champaign

Rocmop Developer’s Guide

Center for Simulation of Advanced Rockets
University of Illinois at Urbana-Champaign
2270 Digital Computer Laboratory
Urbana, IL

Title:	Rocmop Developer's Guide
Author:	Phillip Alexander and Pornput Suriyamongkol (revision 3)
Subject:	This document describes the design of the Rocmop module
Revision:	3.0
Revision History	Revision 0: Initial Release: March 2005 Revision 1: Reformatted: December 2005 Revision 2: Documentation Week Update: December 2006 Changed to Microsoft Word Format Revision 3: Reorganization: April 2007 Distinguished between Rocmop1 and Rocmop2
Effective Date:	04/13/2007

0.0 Introduction and Related Documents

Rocmop is a Roccom module which improves the quality of unstructured tetrahedral volume meshes through nodal repositioning, a process referred to as *mesh smoothing*. Mesquite is an externally developed serial mesh-smoothing utility which Rocmop instantiates to improve each pane of the mesh in isolation. A simple averaging scheme realigns shared nodes at pane interfaces. See the Rocmop User's Guide for a more in-depth introduction to Rocmop. The Roccom User's Guide is another invaluable to any developer working within the Roccom framework. Another useful resource is the Rocmap User's and Developer's guide which describes many of the classes used by Rocmop.

1.0 Algorithm and Features

Rocmop enables parallel on-line mesh optimization without inducing topological changes on the mesh. This form of mesh optimization, in which nodal repositioning is the only operation, is referred to as *smoothing.* In Rocmop, smoothing is performed in two stages. First, each individual pane is smoothed via an externally developed serial package, Mesquite. The default Rocmop1 requires that this pane include ghost information in the form of a complete five-section *pconn* attribute as described in the Roccom User's Guide. If Rocmop2 is used, it constructs this information itself *ignoring any existing pconn.* When Mesquite is called to optimize the mesh, the boundary of the pane is fixed and nodes on the volume interior are repositioned (see Figure 1). Due to the inclusion of the ghost part of the mesh in the serial step, nodes at real pane boundaries often receive different optimized positions across their incident meshes. Rocmop uses a simple averaging scheme to realign these nodes as shown in Figure 2 below.

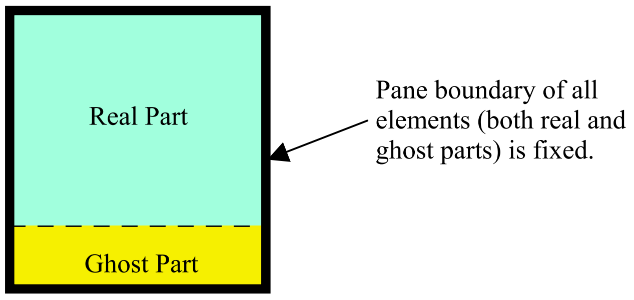
Real Part
Ghost Part
Pane boundary of all elements (both real and ghost parts) is fixed.

Figure 1 Pane boundary is fixed during the smoothing

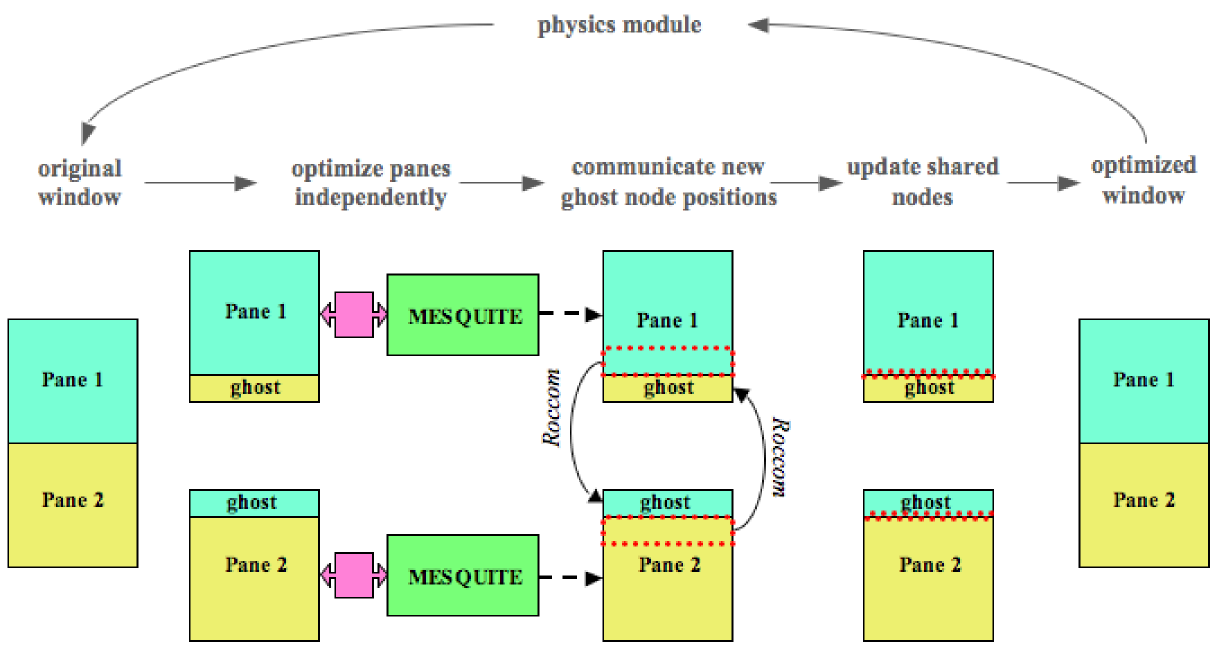

Figure 2 Rocmop smoothing scheme

2.0 Rocmop Implementation

This section explains the implementation of the algorithm in previous section. Rocmop smoothes a mesh inside *Rocmop::smooth( )* (see section 3.3 for more details). Rather than directly smoothing and updating nodal coordinates the input mesh, Rocmop maintains its own buffer window (*_buf_window*). It then smoothes the buffer window and returns a displacement array, which, when added to the original coordinates, produces a smoothed mesh.

Figure 3 shows typical smoothing procedure inside *smooth( )*. On the first call to *smooth( )*, Rocmop clones the input window (*_usr_window*) and store it in its buffer window (*_buf_window*). If Rocmop2 is used, 2nd order ghost layer in *_buf_window* is constructed at this time. Having these windows set up, Rocmop smoothes *_buf_window* and returns the displacements to the caller. The caller then adds these displacements to its nodal coordinates producing smoothed mesh. On subsequent calls to *smooth( )*, Rocmop updates nodal coordinates in _*buf*_window, then performs smoothing and returns the displacement array.

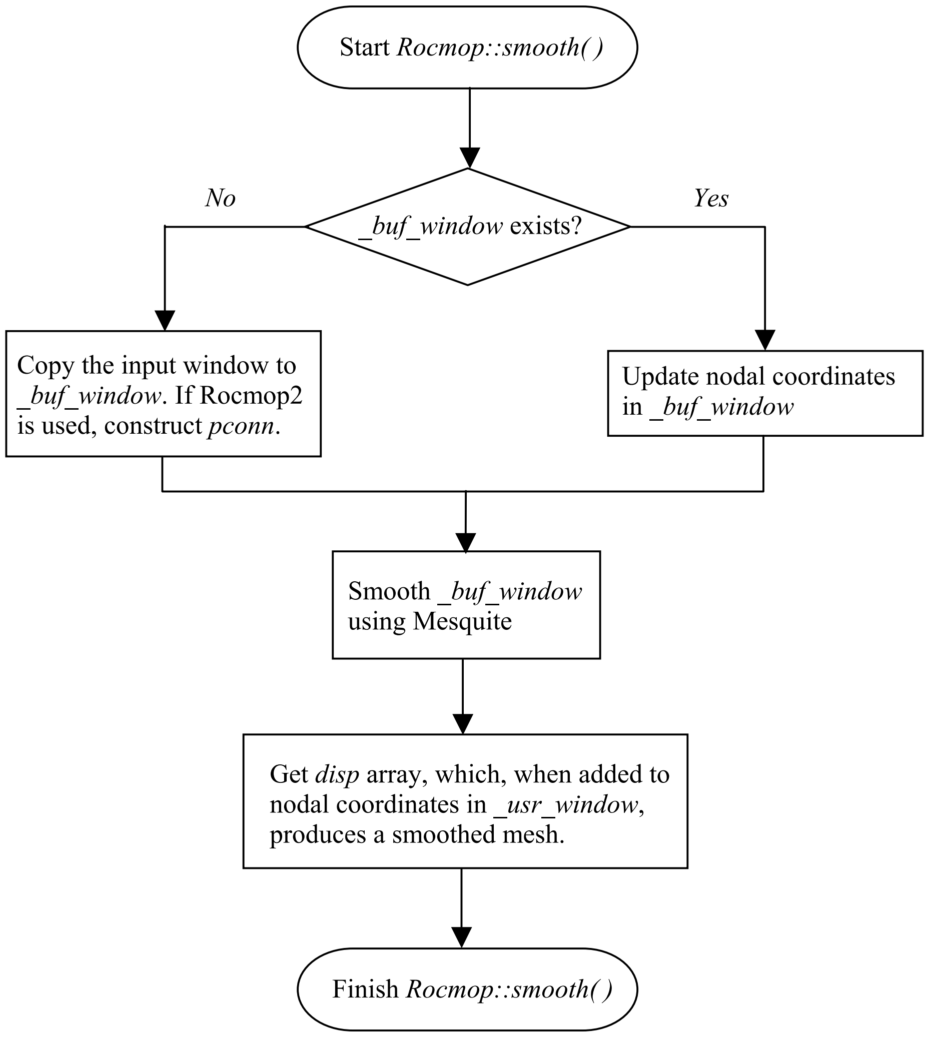
Start *Rocmop::smooth( )*
_buf_window exists?
No
Yes
Copy the input window to *_buf_window*. If Rocmop2 is used, construct *pconn*.
Update nodal coordinates in *_buf_window*
Smooth *_buf_window* using Mesquite
Get *disp* array, which, when added to nodal coordinates in *_usr_window*, produces a smoothed mesh.
Finish *Rocmop::smooth( )*

Figure 3 Smoothing process in Rocmop::smooth( ) function

Currently Rocmop takes its input mesh from Rocflu. Usually, the nodal coordinates of the ghost part from Rocflu are incorrect. Thus Rocmop returns displacements of ghost nodes consisting of 2 quantities: 1) coordinate correction and 2) displacements to smooth them. As a result, these displacements can be very large (especially when the coordinates of the input ghost nodes are significantly off) and do not respect the maximum allowable displacement. Figure 4 shows how the displacements are determined in Rocmop2 after buffer window is smoothed by Mesquite.

Steps	*disp* Array	
	Real Part	Ghost Part
1. Get displacements of real nodes	smoothed - original coordinates	-
2. Scale displacements of real nodes if necessary	* scale if necessary	-
3. Update smoothed NC of real nodes	+ original coordinates	-
4. Update smoothed NC of ghost nodes	-	Rocmap::update_ghost(*disp*)
5. Get displacments	- original coordinates	- original coordinates

Figure 4 Algorithm to determine displacements in Rocmop2

Although Figure 4 shows a reasonable algorithm, it is not implemented in Rocmop1. The algorithm that Rocmop1 currently uses is shown in Figure 5. It should be noted that this algorithm would result in incorrect coordinates of ghost nodes when the displacements are scaled down to not exceed the maximum allowable displacement.

Steps	*disp* Array	
	Real Part	Ghost Part
1. Get displacements	smoothed - original coordinates	smoothed - original coordinates
2. If any displacement of real nodes > the limit Scale down the whole *disp* array	* scale if necessary	* scale if necessary

Figure 5 Algorithm to determine displacements in Rocmop1

3.0 Organization

3.1 Rocmop's Classes

Rocmop: the main smoothing class, Rocmop implements the Roccom module interface and is in charge of using Mesquite to smooth panes in isolation as well as realigning shared nodes through its own member functions. Rocmop has a variety of run-time options which affect its operation. These options are described in the Rocmop User's Guide.

MesqPane: implements the Mesquite mesh interface for Roccom's pane class.

3.2 External Classes

Rocmap::Pane_communicator: abstracts inter-pane communication by providing reduction operations for communicating information on shared nodes. This class is also used to update values stored on ghost nodes or cells with the current value on the corresponding real mesh entities. Pane_communicator requires a full five-section *pconn* attribute, which is why Rocmop also requires this information.

Rocmap::Pane_dual_connectivity: the list of nodes incident on each element is readily available through mesh connectivity tables. This class provides a list of elements incident on each node, information required by Mesquite.

Rocmap::Pane_boundary: used to determine which nodes are on the boundary of each pane. Mesquite is not allowed to reposition these nodes when smoothing the mesh.

3.3 Important Rocmop Subroutines

check_dispalcements(
COM::Attribute *w_disp)

Calculates the maximum nodal displacement across the entire window, and adds this value to a running total. If the running total exceeds the set displacement threshold, then reset the running total to 0 and returns **true** else, returns **false.**

Parameter	***Description***
w_disp	A nodal attribute with three components of type COM_DOUBLE.

constrain_displacements(
COM::Attribute * w_disp)

If necessary, scales a nodal displacement attribute so that the maximum allowable displacement is not exceeded for any real node.

determine_pane_border( )

Determines which nodes and elements are on the border of the pane and stores this information into *is_pane_bnd_node* and *is_pane_bnd_elem* data structures.

determine_shared_border( )

Examines the *pconn* to determine which nodes are shared with a neighboring pane. Stores the result in *is_shared_node* data structure.

mark_elems_from_nodes(
std::vector<std::vector<bool> > &marked_nodes,
std::vector<std::vector<bool> > &marked_elems)

Takes a Boolean per-node property, and determines which elements contain a node which has that property. This is used, for example, to determine which elements touch the physical surface of a pane. The outer vector corresponds to local panes, the inner vector to real nodes or real elements.

print_extremal_dihedrals(
COM::Window* window)

Prints the minimum and maximum dihedral angle across the entire window as well as the pane id of that element, and the id of the element where the extreme value occurs.

set_value(
const char* opt,
const void* value)

Used to set Rocmop's run-time parameters.

Parameter	***Description***
opt	Name of the option to modify (see the table below)
value	New value of the option

Rocmop options accessible through *set_value( )* include:

Option	***Value Type***	***Values***	***Description***
verbose	int	Integers ≥ 0	Verbosity level, used for debugging. Default value is 0, the lowest level
method	int	0	Perform volume smoothing on an unstructured tetrahedral volume mesh. Currently no other valid options
lazy	int	0,1	Check mesh quality before smoothing?
tol	float	(0.0,180.0)	If lazy option is set, mesh only smoothed if maximum dihedral angle exceeds *tol*
maxdisp	float	≥ 0.0	Scale nodal displacement returned by Rocmop so that no node is displaced more than *maxdisp*. If set to 0.0, no scaling is performed
inverted	int	0,1	Set to 1 if tetrahedral node ordering is inverted according to the standards provided in the Roccom User's Guide

smooth(
const COM::Attribute *pmesh,
COM::Attribute *disp)

Smoothes the mesh associated with *pmesh*, and returns the displacement to the new nodal coordinates. See the Rocmop User's Guide for a description of the run-time options which affect how this process.

Parameter	***Description***
Pmesh	The Roccom *pmesh* attribute of the mesh to be smoothed. See the Roccom User's guide for a description of the *pmesh* attribute
disp	A nodal buffer with three components of type COM_DOUBLE. At exit, it contains a displacement, which, when added to the current nodal coordinates of the mesh, produces a smoothed mesh

3.4 Important MesqPane Subroutines

MesqPane implements the Mesquite interface defined in

`Rocmop/External/mesquite_0_9_5/include/MeshInterface.hpp`

The majority of the interface functions are basic mesh operations which are easily understood from source code comments. However, there are a few exceptions which are described here. Rocmop currently uses MesqPane_95.h and MesqPane_95.C which implement Mesquite v. 0.95 interface. Though not fully tested, MesqPane_1_1.h and MesqPane_1_1.C implement Mesquite v.1.1.4 interface and are stored in `Rocmop/include/etc` and `Rocmop/src/etc` respectively.

void invert( )

Inverts all of the elements in a tetrahedral mesh by swapping the 2nd and 4th node ids in the connectivity table. This function is used to repair Rocflu's meshes which use a different tetrahedron node ordering standard than Roccom.

void init( )

Allocates memory, builds dual connectivity tables, and determines the panes border nodes.

void tag_create(
const msq_std::string& tag_n,
TagType type,
unsigned length,
const void* default_value,
MsqError &err)

Similar to Roccom's attributes, Mesquite uses *tags* to store mesh information associated with mesh entities. This function creates a new tag with the given name, creates a vector for storing its data, and adds the tag to a mapping from tag names to tag objects.

4.0 Data Structures

Other than scalar values and pointers to objects of externally defined classes, Rocmop contains only one interesting data structure, which is used to store boolean information on a node-by-node basis:

```
std::<vector<std::vector<bool> > _is_shared_node;
```

The outer vector indexes local panes, while the interior vector indexes the nodes on that pane. Other class members which use the same data structure include the following:

```
_is_shared_elem
_is_phys_bnd_node
_is_phys_bnd_elem
_is_pane_bnd_node
_is_pane_bnd_elem
```

5.0 Mesquite Usage in Rocmop

Mesquite is a complicated package which provides a wide variety of smoothing algorithms, quality measures, and termination criteria. In the interest of reducing the computational intensity of mesh smoothing, Rocmop's only uses Mesquite to perform a single iteration of a Feasible Newton solver to improve the mesh, using the inverse mean ratio as a quality measure. This design choice induces limitations which must be taken into account when using the module, both in terms of the meshes which it will accept, and in terms of the degree of optimization achievable.

Although Mesquite provides untangling routines, Rocmop does not make use of them. Therefore, the mesh passed to Mesquite should not contain any inverted elements. The smoothing package will detect these elements and not perform any optimization on the mesh. Element inversion is already a simulation-halting event, so this design choice seems reasonable.

The choice of only using a single iteration of the Feasible Newton solver also has ramifications. This algorithm improves mesh quality monotonically, so we do not need to worry that early termination will degrade the discretization. However, cases have been observed in which the mesh deforms so rapidly that Mesquite is unable to move interior nodes quickly enough to keep ahead of the moving physical boundary, eventually leading to mesh deformation. This situation is overcome by either taking shorter time steps, or through multiple calls to Rocmop at each time step.

Little attention has been paid to the node-realignment step of the smoothing process mentioned earlier in this paper. The method is literally two steps: average shard node positions across pane boundaries and then update the positions of the corresponding ghost nodes. Despite the simplicity of the algorithm, acceleration of quality degradation at pane boundaries has not been observed, although the possibility of such behavior remains a concern.