GiD Mesh Library: AdvancingFrontTetrahedra Mesher 1.0.0 Module

GiDML_AdvancingFrontTetrahedraMesher module

1 Introduction 4
2 Module description 5
2.1 General overview 6
2.2 Forced nodes 7
2.3 Mesh size information 8
3 Module functions 9
3.1 Module and version information 10
3.2 GiDML_AdvancingFrontTetrahedraMesher 12
3.3 GiDML_AdvancingFrontTetrahedraMesher_CheckConsistency 13
3.4 GiDML_AdvancingFrontTetrahedraMesher_GetErrorString 14
3.5 GiDML_AdvancingFrontTetrahedraMesher_DeleteGiDMLOutputContent 15
4 Input data 16
4.1 Spatial dimension 17
4.2 Input nodes 18
4.2.1 Module attributes for input nodes 19
4.3 Input faces 20
4.4 Input elements 21
4.4.1 Module attributes for input elements 22
4.5 Input parameters 23
5 Output data 24
5.1 Output nodes 25
5.2 Output elements 26
6 Example 27
6.1 Mesh of a cube 28
6.2 Mesh of a cube with a forced node 29
7 Terms of use of the module 31

GiDML_AdvancingFrontTetrahedraMesher module

Welcome to your new documentation space!

Use it to **organize any information that people need to find and reference easily**, like your organisation's travel policies, design guidelines, or marketing assets.

To start, you might want to:

- **Customise this overview** using the **edit icon** at the top right of this page.
- **Create a new page** by clicking the **+** in the space sidebar.

Tip: Add the label *featured* to any pages you want to appear in the **Featured pages** list below.

Search

Featured Pages

Filter by label

There are no items with the selected labels at this time.

Recently Updated

Input nodes
Jan 22, 2024 • contributed by Abel Coll Sans

Mesh of a cube with a forced node
Jan 17, 2024 • contributed by Abel Coll Sans

Mesh of a cube
Jan 17, 2024 • contributed by Abel Coll Sans

Output elements
Jan 17, 2024 • contributed by Abel Coll Sans

Output nodes
Jan 17, 2024 • contributed by Abel Coll Sans

NEED INSPIRATION?

Check out our guide on building better documentation to learn best practices for creating and organizing documents in Confluence.

Introduction

This is the documentation of the module of the GiDMeshLibrary **GiDML_AdvancingFrontTetrahedraMesher**. It refers to its **version 1.0.0**.

Additional information can be found at https://www.gidsimulation.com/gid-for-science/gid-plus/gidml

For any comment or suggestion, please contact gidml@cimne.upc.edu.

Module description

General overview

The **GiDML_AdvancingFrontTetrahedraMesher** module is an unstructured volume mesh generator based on the advancing front meshing technique. It gets a triangle mesh defining the contours of a volume (these triangles must be a water-tight definition of the contour of the volume) and returns a tetrahedra mesh of it. Because of the nature of the algorithm, the volume mesh is always constrained to the triangles defining the contours of the volume.

Its fields of use are wide, considering all the numerical methods working with unstructured tetrahedra: Computational Fluid Dynamics, Structural Analysis, Fluid Structure Interaction, etc...

One of its main characteristics is the high quality of the tetrahedra generated, compared with other meshing methods.

It can be downloaded from the website http://www.gidhome.com/gidml

This document focus on how to use the module, rather than how the advancing front algorithm works internally.

Forced nodes

The **GiDML_AdvancingFrontTetrahedraMesher** module is able to generate volume meshes including nodes in specific positions in space inside the volume. The coordinates of these positions must be indicated in the input data.

Mesh size information

There are some parameters to control the mesh size inside the volume such as the general size, or the sizes transition factor. Furthermore, a background mesh can be used to define a distribution of sizes in different regions of the space ocuppied by the volume to be meshed. In this case, a specific size is set in each element of the background mesh, and the volume mesher will use that mesh for the mesh generation in that region.

Module functions

Module and version information

These are the module functions related to the module information (name and version number).

GiDML_AdvancingFrontTetrahedraMesher_GetModuleName

Declaration:
const char *GiDML_AdvancingFrontTetrahedraMesher_GetModuleName();
Definition:
This function provides with the name of this module. It may be usefull for checking input data read from a .gidml file, for instance (see Files).
Parameters:
No parameters for this function.

Returned value:
The name of the GiDML_AdvancingFrontTetrahedraMesher module is returned in a *const char** format. It is the value defined in the gidml_advancingfront_tetrahedra_mesher.h file, under the *#define GiDML_ADVANCINGFRONT_TETRAHEDRA_MESHER_MODULE_NAME*.

GiDML_AdvancingFrontTetrahedraMesher_GetModuleVersion

Declaration:
const char *GiDML_AdvancingFrontTetrahedraMesher_GetModuleVersion();
Definition:
This function provides with the version of this module.
Parameters:
No parameters for this function.

Returned value:
The version of the GiDML_AdvancingFrontTetrahedraMesher module is returned in a *const char** format. It is the value defined in the gidml_advancingfront_tetrahedra_mesher.h file, under the *#define GiDML_ADVANCINGFRONT_TETRAHEDRA_MESHER_MODULE_VERSION*.

GiDML_AdvancingFrontTetrahedraMesher_GetModuleFormatVersion

Declaration:
const char *GiDML_AdvancingFrontTetrahedraMesher_GetModuleFormatVersion();
Definition:
This function provides with the version of the format used by this module to write the input and output data in .gidml files (see Files).
Parameters:
No parameters for this function.

Returned value:
The version of the GiDML_AdvancingFrontTetrahedraMesher format is returned in a *const char** format. It is the value defined in the gidml_advancingfront_tetrahedra_mesher.h file, under the *#define GiDML_ADVANCINGFRONT_TETRAHEDRA_MESHER_FORMAT_VERSION*.

GiDML_AdvancingFrontTetrahedraMesher_GetVersion_Number

Declaration:
double GiDML_AdvancingFrontTetrahedraMesher_GetVersion_Number(const char* version);
Definition:
This function provides with a number (in double format) corresponding to the version provided by the GiDML_AdvancingFrontTetrahedraMesher_GetModuleFormatVersion and GiDML_AdvancingFrontTetrahedraMesher_GetModuleVersion functions (in const char* format). It is guaranteed that the number obtained from a newer version is greater than the obtained from an older one. This is useful to detect if a module version is newer or older than another.
Parameters:
Only one parameter is required by this function: the version id (in *const char** format) from which the number is required.

Returned value:
The number associated to the version is returned in double format.

GIDML_AdvancingFrontTetrahedraMesher_GetMinimumVersionToSaveInput

Declaration:
const char* GIDML_AdvancingFrontTetrahedraMesher_GetMinimumVersionToSaveInput(const GiDMLInput_Handle hdl_gin);
Definition:
This function provides with the oldest number of format version in which the data included in the GiDMLInput handle can be saved. In some occasions it is useful to save the data in the .gidml file in older versions, to allow programs linked with older versions of the module run some cases.
Parameters:
Only one parameter is required by this function which is the GiDMLInput handle where the data is.

Returned value:
The older version in which the data included in the GiDMLInput handle is returned in const char* format.

GIDML_AdvancingFrontTetrahedraMesher_GetMinimumVersionToSaveOutput

Declaration:
const char* GIDML_AdvancingFrontTetrahedraMesher_GetMinimumVersionToSaveOutput(const GiDMLOutput_Handle hdl_gout);
Definition:
This function provides with the oldest number of format version in which the data included in the GiDMLOutput handle can be saved. In some occasions it is useful to save the data in the .gidml file in older format versions, to allow programs linked with older versions of the module run some cases.
Parameters:
Only one parameter is required by this function which is the GiDMLOutput handle where the data is.

Returned value:
The older version in which the data included in the GiDMLOutput handle is returned in const char* format.

GiDML_AdvancingFrontTetrahedraMesher

This is the function calling to the mesh generator itself.

Declaration:
int GiDML_AdvancingFrontTetrahedraMesher(const GiDMLInput_Handle hdl_gin, GiDMLOutput_Handle hdl_gout);
Definition:
This function is the unstructured tetrahedra mesher itself.
Parameters:
The function receives the input data handle hdl_gin with the corresponding volume boundaries, parameters, etc... and returns the final mesh in the output data handle hdl_gout.

Returned value:
This function returns an error_id (in integer format) that can be processed by GiDML_AdvancingFrontTetrahedraMesher_GetErrorString function (see https://gidsimulation.atlassian.net/wiki/spaces/GIDMLAF/pages/2577137724). If all the meshing process has finalized successfully, it returns 0.

GiDML_AdvancingFrontTetrahedraMesher_CheckConsistency

Declaration:
int GiDML_AdvancingFrontTetrahedraMesher_CheckConsistency(const GiDMLInput_Handle hdl_gin);
Definition:
This function performs light checks to the input data to ensure it is right for generating the mesh with the GiDML_AdvancingFrontTetrahedraMesher function (see https://gidsimulation.atlassian.net/wiki/spaces/GIDMLAF/pages/2577072207). For instance: if there are no faces in the input data defining the contours of the volumes, the module could not generate the mesh.
Parameters:
The function receives the input data handle hdl_gin which is the candidate to be used for generating the mesh using the GiDML_AdvancingFrontTetrahedraMesher function.

Returned value:
This function returns an error_id (in integer format) that can be processed by GiDML_AdvancingFrontTetrahedraMesher_GetErrorString function (see https://gidsimulation.atlassian.net/wiki/spaces/GIDMLAF/pages/2577137724). If all the data inside the GiDMLInput structure are coherent, it returns 0.

GiDML_AdvancingFrontTetrahedraMesher_GetErrorString

Declaration:
const char* GiDML_AdvancingFrontTetrahedraMesher_GetErrorString(const int error_id);
Definition:
This function provides with the meaningful information corresponding to an error_id returned by the functions GiDML_AdvancingFrontTetrahedraMesher, GiDML_AdvancingFrontTetrahedraMesher_CheckConsistency.
Parameters:
The function receives as parameter an error_id in integer format.

Returned value:
The function returns the error message corresponding to the error_id in const char* format. They are listed hereafter:

error_id	message
0	"Everything is ok"
1	"Error in advancingfront tetrahedra mesher"
2	"There is no GiDMLInput handle"
3	"There are no nodes inside GiDMLInput handle"
4	"There are no faces inside GiDMLInput handle"
5	"Contour element of volume is bad"
6	"Error entering point in octree while meshing"
7	"Error in advancingfront tetrahedra mesher processing the input data"
8	"User stopped the meshing process"
9	"Couldn't mesh at this location."
10	"Couldn't mesh: not enough space in octtree -2-"
11	"Error TakeFaceOut face -1-"
12	"Error creating face -1-"
13	"Error forcing points to volume mesh"
14	"Unknown error in advancingfront tetrahedra mesher"

In the header file gidml_advancingfront_tetrahedra_ mesher.h file this list of messages is also present.

GiDML_AdvancingFrontTetrahedraMesher_DeleteGiDMLOutputContent

Declaration:
int GiDML_AdvancingFrontTetrahedraMesher_DeleteGiDMLOutputContent(GiDMLOutput_Handle hdl_gout);
Definition:
This function deletes the content of the GiDOutput structure corresponding to the GiDMLOutput_Handle created by the GiDML_AdvancingFrontTetrahedraMesher module. As the module has created these data and filled the GiDMLOutput with them, it is the responsible to delete them.

Note that the GiDMLOutput structure corresponding to the handle is not deleted. It should be deleted using the GiDML_IO_DeleteGiDMLOutputHandle function from the GiDML_IO module (see GiDML IO module).
Parameters:
The function receives the output data handle hdl_gout.

Returned value:
This function returns 0 if all the deletion process has been done with no problems, and 1 if there has been some problem.

Input data

Spatial dimension

The GiDML_AdvancingFrontTetrahedraMesher module is always working in 3D (spatial dimension equal to 3).

Input nodes

All the nodes involved in the input data must be introduced in the list of nodes coordinates of the GiDMLInput handle, following the standard API functions of GiDML_IO module (see GiDML IO module). It has to be noted that the GiDML_AdvancingFrontTetrahedraMesher works always in 3D (3 coordinates per node).

Note that all the nodes involved in the input data must be entered. Those are:

- the ones belonging to the triangles of the contours of the volumes (these are mandatory)
- the ones belonging to the background mesh used for mesh size information (not mandatory)
- the ones which must be forced nodes in the final mesh (not mandatory)

Note that the order to enter the nodes coordinates in the coordinates vector of the input data must be this: nodes from triangles of the contour, forced nodes (if they exist) and nodes of background mesh (if they exist).

In the connectivities information of the elements of the GiDMLInput structure, as well as in possible reference to nodes in the attributes or parameters, the id of the node refers always to the position of the node in this list of nodes coordinates (beginning from 0 position).

Module attributes for input nodes

The module attributes for the input nodes for the GiDML_AdvancingFrontTetrahedraMesher module are listed hereafter.

Note that, for using the API functions, all of them are of entity type *GIDML_NODE* and type of attribute *GIDML_MODULE_ATTRIBUTE*.

Name	Value type	Description	Possible values
GiDML_ADVANCINGFRONT_NODES_SIZES	GIDML_TYPE_DOUBLE	This attribute indicates mesh size required in the nodes present in the input data.	If the value is 0.0 of negative, no mesh size is considered in that node.

Note that these attributes are not mandatory for the mesh generator. By default, if no attribute is set, no specific size assigned to any node

Input faces

All the faces involved in the input data must be introduced in the faces connectivities of the GiDMLInput handle, following the standard API functions of GiDML_IO module (see GiDML IO module). The GiDML_AdvancingFrontTetrahedraMesher works only with **3 nodes Triangle** faces in the input data.

Note that all the faces defining the contours of the volume to be meshed must be entered, and these faces must represent a water-tight definition of the contours of the volume. These contour elements are also needed to be oriented towards the inner part of the volume

The connectivities of the triangles are provided with the id of the corresponding nodes which is the position of the node in this list of nodes coordinates (beginning from 0 position).

Input elements

The GiDML_AdvancingFrontTetrahedraMesher only get input elements to provide mesh size information for the final mesh. These are the elements of the background mesh used for the mesh size information. It accepts **4 nodes Tetrahedra** and **8 nodes Hexahedra** types of elements.

The connectivities of the elements are provided with the id of the corresponding nodes which is the position of the node in the list of nodes coordinates (beginning from 0 position).

Note that the possible reference to an element from any attribute refers always to the position of the element in this list of elements (beginning from 0 position).

Module attributes for input elements

There is only one module attribute for the input elements for the GiDML_AdvancingFrontTetrahedraMesher module, which indicates the mesh size assigned to them.

Name	Value type	Description	Possible values
GiDML_ADVANCINGFRONT_BACKGROUND_ELEMENTS_SIZES	GIDML_TYPE_DOUBLE	This attribute corresponds to the mesh size assigned to the elements.	If the value is 0.0 or negative, the size is not taken into account.

Note that, for using the API functions, this attribute is of entity type *GIDML_ELEMENT* and type of attribute *GIDML_MODULE_ATTRIBUTE*.

This attribute is not mandatory for the mesh generator. By default, if it is not set, the input elements are not considered in the GiDML_AdvancingFrontTetrahedraMesher module.

Input parameters

This section refers to the parameters from the GiDML_AdvancingFrontTetrahedraMesher module format version 1.0 on.

Parameters

The input scalar parameters for the GiDML_AdvancingFrontTetrahedraMesher module are listed hereafter. Note that all the parameters are stored as doubles, however, they can be set as integers also using the API functions. Values indicated as default are the ones taken if the parameter is not set.

Name	Description	Possible values
GiDML_ADVANCINGFRONT_GENERAL_MESH_SIZE	This is the general mesh size, which will be applied to the volume mesh if there is no background mesh sizes information.	If the value is 0.0 (default value), the size is not taken into account and a portion of the bounding box of the volume is taken.
GiDML_ADVANCINGFRONT_TETRAHEDRA_SIZE_TRANSITION_FACTOR	This parameter controls whether the transitions between different element sizes are slow or fast. In models with different mesh sizes assigned, a low value of this parameter will imply meshes with more elements than using a high value.	This value must be a real number between 0.0 and 1.0. The default value is 0.6.
GiDML_ADVANCINGFRONT_BOUNDARY_WEIGHTED_TRANSITION	This parameter is used to control the transition of sizes pattern.	If this parameter is set (value equal to 1), the transition size follows in a more parallel way the elements in the boundary of the mesh; otherwise (if the parameter is equal to 0) the size transition is more uniform along the mesh.
GiDML_ADVANCINGFRONT_AVOID_BOUNDARY_ELEMENTS	This parameter force the mesher not to generate any element with all its nodes in the contours of the volume.	If the value is 0,0 (default value), the mesher can generate elements with no restriction of this type. If the value is 1.0, the mesh returned is ensured not to have any element with all its nodes in the contour of the volume.
GiDML_ADVANCINGFRONT_DONT_RETURN_IF_NEGATIVE_JACOBIANS	This parameter indicates if the resulting mesh should be returned or not, in case some of its elements has negative jacobians.	If the value if 0.0 (default value), the resulting mesh is always returned in the output data. If it is 1.0, the resulting mesh is not returned in the output data if some element has negative jacobian.
GiDML_ADVANCINGFRONT_NUMBER_OF_FORCED_NODES	This indicates the number of forced nodes in the input data.	If the value is 0.0 (default value), there is no forced node to be considered.

Output data

Output nodes

All the nodes involved in the output data are provided in the list of nodes coordinates of the GiDMLOutput handle. These are the nodes from the tetrahedra generated by the mesher. Note that the first nodes appearing in this list are the ones from the input data which are constrained (the ones of the input boundaries of the volume, and the forced nodes in case there are some of them).

It has to be noted that the GiDML_AdvancingFrontTetrahedraMesher works always in 3D (3 coordinates per node).

The GiDML_AdvancingFrontTetrahedraMesher module is not returning any **user attribute** in the output nodes.

Output elements

All the volume elements involved in the output data are provided in the elements' connectivities of the GiDMLOutput handle. These are the linear tetrahedra (4 nodes) generated by the mesher.

The GiDML_AdvancingFrontTetrahedraMesher module is not returning any **user attribute** in the output elements.

Example

Mesh of a cube

In this example we are generating a mesh of a cube of 10 units of length size. The contour mesh of the cube is harcoded in this example. The following code is providing to the mesher (as input) the contour mesh of the volume to be meshed (made of 3 node-triangles), and the general element size desired for the volume mesh as a parameter.

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_advancingfront_tetrahedra_mesher.h"
#define NAME_OF_MY_PROGRAM "MY_PROGRAM" //identifier of who is creating the input data
int main() {
  //in this example a cube which contour is defined by 12 faces (3 node-triangles) is used
  const double coordinates[24]={0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,0,10,10};
  const int faces_conectivities[36]=
{0,1,3,1,3,2,1,2,5,2,5,6,0,4,3,3,4,7,4,5,7,5,6,7,0,1,4,1,4,5,3,2,7,7,2,6};
  //note that the triangle faces defining the contour of the volume are needed to be oriented
coherently (towards inner or outer part of it).

  //Create GiDMLInput Handle.
  const int n_dimension = 3; //space dimension of the data
  const char *module_name = GiDML_AdvancingFrontTetrahedraMesher_GetModuleName();
  const char *module_format_version = GiDML_AdvancingFrontTetrahedraMesher_GetModuleFormatVersion();
//module data
  GiDMLInput_Handle hdl_input=GiDML_IO_NewGiDMLInputHandle(module_name, module_format_version,
NAME_OF_MY_PROGRAM, n_dimension);
  //the module name and format are interesting in case that the input is saved/read to/from a auxiliary
file, in order to identify its use

  //Fill GiDMLInput Handle with data
  //Nodes
  const int number_of_nodes = 8;
  GiDML_IO_SetNodesCoords(hdl_input, number_of_nodes, n_dimension,number_of_nodes,coordinates); //nodes
data

  //Faces
  const int number_of_faces = 12;
  const ElemType faces_element_type = GID_TRIANGLE_ELEMENT;
  const int nnode_faces = 3;
  GiDML_IO_SetFaces(hdl_input, number_of_faces, faces_conectivities, faces_element_type, nnode_faces);
//faces data

  //Parameters
  //add the parameter 'general_mesh_size'
  const double general_size = 2.0;
  GiDML_IO_SetParameter(hdl_input,"GiDML_ADVANCINGFRONT_GENERAL_MESH_SIZE", general_size);

  //Check the that the Input format is correct.
  int error_returned = GiDML_AdvancingFrontTetrahedraMesher_CheckConsistency(hdl_input);

  //GiDMLOutput Handle.
  GiDMLOutput_Handle hdl_output = GiDML_IO_NewGiDMLOutputHandle();
  if ( error_returned == 0){
    //Call the mesher.
    error_returned = GiDML_AdvancingFrontTetrahedraMesher(hdl_input,hdl_output);
  }

  if ( error_returned ){
    const char *error_message = GiDML_AdvancingFrontTetrahedraMesher_GetErrorString(error_returned);
    //somehow show this error message...
  } else {
    //Get the tetrahedra mesh
    const int number_of_nodes_in_tetra_mesh = GiDML_IO_GetNumberOfNodes( hdl_output );
    const int num_of_tetras = GiDML_IO_GetNumberOfElements( hdl_output );
    double *coords = NULL;
    GiDML_IO_GetNodesCoords( hdl_output , coords);
    int *tetras_connectivity = NULL;
    int fail=GiDML_IO_GetElements( hdl_output,tetras_connectivity);
  }

  //Delete data from GiDMLOutput
  GiDML_AdvancingFrontTetrahedraMesher_DeleteGiDMLOutputContent(hdl_output);

  //Delete data structures inside GiDMLInput and GiDMLOutput structures
  GiDML_IO_DeleteGiDMLInputHandle(hdl_input);
  GiDML_IO_DeleteGiDMLOutputHandle(hdl_output);

  return 0;
}
```

Mesh of a cube with a forced node

This example is basically the same as Mesh of a cube , but forceing the mesher to include an inner point inside the cube. In this case it is the point with coordinates (0.2, 0.2, 0.2), and it has no mesh size associated.

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_advancingfront_tetrahedra_mesher.h"
#define NAME_OF_MY_PROGRAM "MY_PROGRAM" //identifier of who is creating the input data
int main() {
  //in this example a cube which contour is defined by 12 faces (3 node-triangles) is used
  const double coordinates[27]=
{0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,0,10,10,0.2,0.2,0.2};
  //note that the last three coordinates are the ones of the forced node.
  const int faces_conectivities[36]=
{0,1,3,1,3,2,1,2,5,2,5,6,0,4,3,3,4,7,4,5,7,5,6,7,0,1,4,1,4,5,3,2,7,7,2,6};
  //note that the triangle faces defining the contour of the volume are needed to be oriented
coherently (towards inner or outer part of it).

  //Create GiDMLInput Handle.
  const int n_dimension = 3; //space dimension of the data
  const char *module_name = GiDML_AdvancingFrontTetrahedraMesher_GetModuleName();
  const char *module_format_version = GiDML_AdvancingFrontTetrahedraMesher_GetModuleFormatVersion();
//module data
  GiDMLInput_Handle hdl_input=GiDML_IO_NewGiDMLInputHandle(module_name, module_format_version,
NAME_OF_MY_PROGRAM, n_dimension);
  //the module name and format are interesting in case that the input is saved/read to/from a auxiliary
file, in order to identify its use

  //Fill GiDMLInput Handle with data
  //Nodes
  const int number_of_nodes = 8;
  GiDML_IO_SetNodesCoords(hdl_input, number_of_nodes, n_dimension,number_of_nodes,coordinates); //nodes
data

  //Faces
  const int number_of_faces = 12;
  const ElemType faces_element_type = GID_TRIANGLE_ELEMENT;
  const int nnode_faces = 3;
  GiDML_IO_SetFaces(hdl_input, number_of_faces, faces_conectivities, faces_element_type, nnode_faces);
//faces data

  //Parameters
  //add the parameter 'general_mesh_size'
  const double general_size = 2.0;
  GiDML_IO_SetParameter(hdl_input,"GiDML_ADVANCINGFRONT_GENERAL_MESH_SIZE", general_size);

  //add the parameter indicating number of forced points
  const int n_forced_points=1;
  GiDML_IO_SetParameter(hdl_input, "GiDML_ADVANCINGFRONT_NUMBER_OF_FORCED_NODES", n_forced_points);

  //Check the that the Input format is correct.
  int error_returned = GiDML_OctreeTetrahedraMesher_CheckConsistency(hdl_input);

  //GiDMLOutput Handle.
  GiDMLOutput_Handle hdl_output = GiDML_IO_NewGiDMLOutputHandle();
  if ( error_returned == 0){
    //Call the mesher.
    error_returned = GiDML_OctreeTetrahedraMesher(hdl_input,hdl_output);
  }

  if ( error_returned ){
    const char *error_message = GiDML_AdvancingFrontTetrahedraMesher_GetErrorString(error_returned);
    //somehow show this error message...
  } else {
    //Get the tetrahedra mesh
    const int number_of_nodes_in_tetra_mesh = GiDML_IO_GetNumberOfNodes( hdl_output );
    const int num_of_tetras = GiDML_IO_GetNumberOfElements( hdl_output );
    double *coords = NULL;
    GiDML_IO_GetNodesCoords( hdl_output , coords);
    int *tetras_connectivity = NULL;
    int fail=GiDML_IO_GetElements( hdl_output,tetras_connectivity);
  }

  //Delete data from GiDOutput
  GiDML_AdvancingFrontTetrahedraMesher_DeleteGiDMLOutputContent(hdl_output);

  //Delete data structures inside GiDMLInput and GiDMLOutput structures
  GiDML_IO_DeleteGiDMLInputHandle(hdl_input);
  GiDML_IO_DeleteGiDMLOutputHandle(hdl_output);

  return 0;
}
```

Terms of use of the module

CIMNE is the propietary of this module. Any use of it involves the signment of an agreement with CIMNE.

Please, contact gidml@cimne.upc.edu if you are interested in integrating the GiDML_AdvancingFrontTetrahedraMesher module inside your software.