GiD Mesh Library: Image2Mesh 1.1.0 Module

Introduction

This is the documentation of the module of the GiDMeshLibrary **GiDML_Image2Mesh**. It refers to its **version 1.1.0**.

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_Image2Mesh** module is a volume mesh generator directly from 3D images. It generates unstructured tetrahedra from a 3D grid of voxels, in which the color of the material is coded. It refines the mesh adaptively in the regions where there are different materials, in order to well capture the interface between them.

It can be used in different fields, considering for instance the automatic mesh generation of **3D medical images** or **tomography** images for materials among others.

Its main characteristics are:

- very **fast and robust**: it generates always a mesh from the 3D image. It is based on an octree technology, which makes it extremely fast, also taking advantage on a parallel implementation following shared memory paradigm.
- **Configurable resolution** of the final mesh: one can arrive to the level of resolution of the voxel size in the final mesh, but it can be limited in order not to reach too large meshes. Depending on the level of resolution, some details of the initial 3D image may be skipped or not.
- Ensures **periodicity** in the final mesh if it is required by the user. In that case, the information of the relationship between periodic nodes and elements is computed and provided automatically.

The meshing algorithm followed is based in the PhD thesis:
A.Coll, "**Robust volume mesh generation for non-watertight geometries**". PhD thesis, Polytechnical University of Catalonia (UPC-BarcelonaTech), Spain, 2014.

It can be downloaded from the website https://www.gidsimulation.com/gid-for-science/gid-plus/gidml

Requirements for the input 3D image

In the current version of the module only two different materials are accepted. They are enough to separate two regions, namely interior and exterior, or two different materials in one sample. This kind of images are normally obtained from a segmentation process applied previously to the original dataset.

Regarding the geometry of the dataset it is not required that image grid has the same number of voxels at each direction, the same applies to the size of the voxels, ie the voxel is not required to be isotropic.

Periodicity in the final mesh

These kinds of meshers (from 3D images) are often used for obtaining Representative Volume Elements (RVE) for materials properties simulations. In these cases, it is common to require a periodicity in the geometrical model, both for geometry and material properties.

For this purpose an specific parameter ("GIDML_IMG2MESH_PERIODIC_MESH") can be set in the input data, in order to ensure the final mesh is periodic in geometry and material. That is:

- there are nodes in the opposite faces of the image aligned with the Cartesian axis that can be related one 2 one
- these nodes have the same material.

The relationship between periodic nodes is get by the 2 nodes edges provided in the output data (see Output edges).

It has to be considered that, although the simulation may require a periodic mesh, the 3D image may not be periodic concerning the materials (it is always periodic in terms of geometry, as it is defined in a regular Cartesian grid). In that cases, the mesher is changing arbitrarialy one of the nodes colors in order to reach the material periodicity.

Module functions

All the functions described hereafter are declared in the header file gidml_from_image_2_mesh.h.

Module and version information

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

GiDML_Image2Mesh_GetModuleName

Declaration:
const char *GiDML_Image2Mesh_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_Image2Mesh module is returned in a *const char** format. It is the value defined in the gidml_from_image_2_mesh.h file, under the *#define GiDML_IMAGE2MESH_MODULE_NAME*.

GiDML_Image2Mesh_GetModuleVersion

Declaration:
const char *GiDML_Image2Mesh_GetModuleVersion();
Definition:
This function provides with the version of this module.
Parameters:
No parameters for this function.

Returned value:
The version of the GiDML_Image2Mesh module is returned in a *const char** format. It is the value defined in the gidml_from_image_2_mesh.h file, under the *#define GiDML_IMAGE2MESH_MODULE_VERSION*.

GiDML_Image2Mesh_GetModuleFormatVersion

Declaration:
const char *GiDML_Image2Mesh_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_Image2Mesh format is returned in a *const char** format. It is the value defined in the gidml_from_image_2_mesh.h file, under the *#define GiDML_IMAGE2MESH_FORMAT_VERSION*.

GiDML_Image2Mesh_GetVersion_Number

Declaration:
double GiDML_Image2Mesh_GetVersion_Number(const char* version);
Definition:
This function provides with a number (in double format) corresponding to the version provided by the GiDML_Image2Mesh_GetModuleFormatVersion and GiDML_Image2Mesh_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_Image2Mesh_GetMinimumVersionToSaveInput

Declaration:
const char* GiDML_Image2Mesh_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_Image2Mesh_GetMinimumVersionToSaveOutput

Declaration:
const char* GiDML_Image2Mesh_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_Image2Mesh_CheckModuleAndVersion

Declaration:
int GiDML_Image2Mesh_CheckModuleAndVersion(const GiDMLIO_Handle hdl_gio);
Definition:
This function checks whether the module the data in the handle is for is the same as GiDML_Image2Mesh module, and if its format version is the same as the one in the module.This function is useful when reading data from .gidml files which were written with different versions of the module.
Parameters:
The handle of the GiDMLInput or GiDMLOutput data is required as the only parameter for the function.

Returned value:
The function returns the following integer values depending on the case:

- 0 if module and version are the same
- -1 if module name is not the same
- -2 if module format version in hdl_gio is higher than the module one
- 2 if module format version in hdl_gio is lower than the module one

GiDML_Image2Mesh_TransformGiDMLInputFromOlderVersion

Declaration:
int GiDML_Image2Mesh_TransformGiDMLInputFromOlderVersion(const char* older_version, const char* newer_version, GiDMLInput_Handle hdl_gin);
Definition:
This function transform the input data inside the handle, which was written in an older format, to a newer format of the GiDML_Image2Mesh module. This function is useful when working with .gidml files written in other versions than the current of the module.
Parameters:
The function parameters are the older and newer versions identifiers (in const char* format), and the handle containing the data.

Returned value:
The function returns 0 if it has been able to transform data, and 1 if not.

Depending on the versions managed, it may be the case that the data has been transformed to another version, but not the required one (it is transformed progressively from one version to the required one). In that case the function also returns 1.

GiDML_Image2Mesh_TransformGiDMLInputToOlderVersion

Declaration:
int GiDML_Image2Mesh_TransformGiDMLInputToOlderVersion(const char* newer_version, const char* older_version, GiDMLInput_Handle hdl_gin);
Definition:
This function transform the input data inside the handle, which was written in a newer format, to an older format of the GiDML_Image2Mesh module. This function is useful when working with .gidml files written in other versions than the current of the module.
Parameters:
The function parameters are the older and newer versions identifiers (in const char* format), and the handle containing the data.

Returned value:
The function returns 0 if it has been able to transform data, and 1 if not.

Depending on the versions managed, it may be the case that the data has been transformed to another version, but not the required one (it is transformed progressively from one version to the required one). In that case the function also returns 1.

GiDML_Image2Mesh_TransformGiDMLOutputFromOlderVersion

Declaration:
int GiDML_Image2Mesh_TransformGiDMLOutputFromOlderVersion(const char* older_version, const char* newer_version, GiDMLOutput_Handle hdl_gout);
Definition:
This function transform the output data inside the handle, which was written in an older format, to a newer format of the GiDML_Image2Mesh module. This function is useful when working with .gidml files written in other

versions than the current of the module.
Parameters:
The function parameters are the older and newer versions identifiers (in const char* format), and the handle containing the data.

Returned value:
The function returns 0 if it has been able to transform data, and 1 if not.

Depending on the versions managed, it may be the case that the data has been transformed to another version, but not the required one (it is transformed progressively from one version to the required one). In that case the function also returns 1.

GiDML_Image2Mesh_TransformGiDMLOutputToOlderVersion

Declaration:
int GiDML_Image2Mesh_TransformGiDMLOutputToOlderVersion(const char* newer_version, const char* older_version, GiDMLOutput_Handle hdl_gout);
Definition:
This function transform the output data inside the handle, which was written in a newer format, to an older format of the GiDML_Image2Mesh module. This function is useful when working with .gidml files written in other versions than the current of the module.
Parameters:
The function parameters are the older and newer versions identifiers (in const char* format), and the handle containing the data.

Returned value:
The function returns 0 if it has been able to transform data, and 1 if not.

Depending on the versions managed, it may be the case that the data has been transformed to another version, but not the required one (it is transformed progressively from one version to the required one). In that case the function also returns 1.

GiDML_Image2Mesh

This is the function calling to the mesh generator itself.

Declaration:
int GiDML_Image2Mesh(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 color (material) information of the pixels, 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_Image2Mesh_GetErrorString function (see GiDML_Image2Mesh_GetErrorString). If all the meshing process has finalized successfully, it returns 0.

GiDML_Image2Mesh_CheckConsistency

Declaration:
int GiDML_Image2Mesh_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_Image2Mesh function (see GiDML_Image2Mesh). For instance: if there is no color information for the pixels, or if the grid parameters for the image are not defined, 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_Image2Mesh function.

Returned value:
This function returns an error_id (in integer format) that can be processed by GiDML_Image2Mesh_GetErrorString function (see GiDML_Image2Mesh_GetErrorString). If all the meshing process has finalized successfully, it returns 0.

GiDML_Image2Mesh_GetErrorString

Declaration:
const char* GiDML_Image2Mesh_GetErrorString(const int error_id);
Definition:
This function provides with the meaningful information corresponding to an error_id returned by the functions GiDML_Image2Mesh, GiDML_Image2Mesh_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 image2mesh mesher"
2	"There is no GiDMLInput handle"
3	"There are no list of voxels colors inside GiDMLInput handle"
4	"There is no information about origin, delta or npoints inside GiDMLInput handle"
5	"Error processing the input data"
6	"Error coloring tetrahedra"
7	"Unknown error in image2mesh mesher"

In the header file gidml_from_image_2_mesh.h file this list of messages is also present.

GiDML_Image2Mesh_DeleteGiDMLOutputContent

Declaration:
int GiDML_Image2Mesh_DeleteGiDMLOutputContent(GiDMLOutput_Handle hdl_gout);
Definition:
This function deletes the content of the GiDMLOutput structure corresponding to the GiDMLOutput_Handle created by the GiDML_Image2Mesh 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

All the input data needed for the GiDML_Image2Mesh module are parameters, as the mesher is not receiving any mesh as input. Basically, the input parameters are to define the regular voxels grid (which is the 3D image) with the corresponding colors, and some other parameter to set some of the options of the mesher.

Input parameters

This section refers to the parameters from GiDML_Image2Mesh module format version 1.2 on.

Parameters

The input scalar parameters for the GiDML_Image2Mesh 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_IMG2MESH_NUMBER_POINTS_X	This parameter indicates the number of points of the 3D image in x direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_NUMBER_POINTS_Y	This parameter indicates the number of points of the 3D image in y direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_NUMBER_POINTS_Z	This parameter indicates the number of points of the 3D image in z direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_SPACING_X	This parameter indicates the spacing between points of the 3D image in x direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_SPACING_Y	This parameter indicates the spacing between points of the 3D image in y direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_SPACING_Z	This parameter indicates the spacing between points of the 3D image in z direction.	Value greater than 0. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_ORIGIN_COORDINATE_X	This parameter indicates the value of the lower x coordinate of the points.	Any real value. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_ORIGIN_COOR	This parameter indicates the value of the lower y coordinate of the points.	Any real value. There are no default values for this parameter. It must always be entered.

DINATE_Y		
GIDML_IMG2MESH_ORIGIN_COORDINATE_Z	This parameter indicates the value of the lower z coordinate of the points.	Any double value. There are no default values for this parameter. It must always be entered.
GIDML_IMG2MESH_MINIMUM_ELEMENT_SIZE	This parameter indicates the minimum element size desired for the output mesh. It has to be considered that the final mesh will have this size if the colors distribution of the 3D image make the mesher refines until that level.	Value equal or greater than 0. If it is 0, the refinement process may arrive to a resolution of the 3D image voxel size in the output mesh.
GIDML_IMG2MESH_PERIODIC_MESH	This parameter indicates if the final mesh must be periodic or not. It has to be considered that the periodicity is conceived by opposite faces of the 3D image aligned to the Cartesian axes.	A value equal to 0 (default value) indicates that the mesh will be not periodic, otherwise it will be.
GIDML_IMG2MESH_RETURN_INTERFACE_TRIANGLES	This parameter indicates if the returned mesh should contain or not the triangles interface between tetrahedra of different color.	A value equal to 0 (default value) implies not to return the triangles in the output mesh. If the parameter takes other values, the triangles are returned.

Parameters vectors

The input vector parameters for the GiDML_Image2Mesh module are listed hereafter. Note that the dimension of the vector must be set in the API functions.

Name	Value type	Array dimension	Description	Possible values
GIDML_IMG2MESH_INPUT_NODES_COLORS	GIDML_TYPE_INTEGER	Number of points in the input data (which results from multiplying the number of points in each direction)	This array contains a color (material) for each node. The colors in the array are ordered in memory such that the x-dimension is running faster, then follows the y-dimension (rows) and finally the z-dimension (slices).	If the value is 0, the node is considered as background color.

Output data

The output data from the GiDML_Image2Mesh module is basically the volume tetrahedra mesh generated from the 3D image represented in the input data. Depending on the input parameters, this output data may also contain triangle faces and line element edges.

The GiDML_Image2Mesh module is not returning any **user attribute** in the 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 elements, faces and edges that the output mesh should have. It has to be noted that the GiDML_Image2Mesh works always in 3D (3 coordinates per node).

Module attributes for the output nodes

The GiDML_Image2Mesh module is providing one module attribute for the output nodes, corresponding to the color of the nodes of the mesh.

Name	Value type	Description	Possible values
GIDML_IMG2MESH_NODES_COLORS	GIDML_TYPE_INTEGER	This attribute indicates the color of each node.	The values of the colors corresponds to the colors of GIDML_IMG2MESH_INPUT_NODES_COLORS parameters vector from the input data. In case the node is onto an interface between two volumes, the id is greater than the maximum volume color.

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

Output elements

All the volume elements involved in the output data are provided in the elements' connectivities of the GiDMLOutput handle. These are the tetrahedra generated by the mesher from the 3D image.

The output elements are linear tetrahedra (4 nodes).

Module attributes for the output elements

The GiDML_Image2Mesh module is providing one module attribute for the output elements, corresponding to the color of the elements of the mesh.

Name	Value type	Description	Possible values
GIDML_IMG2MESH_ELEMENTS_COLORS	GIDML_TYPE_INTEGER	This attribute refers to the color of each tetrahedron, which indicates the volume it is into.	The values of the colors corresponds to the colors of the GIDML_IMG2MESH_INPUT_NODES_COLORS parameters vector from the input data.

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

Output faces

All the faces involved in the output data are provided in the faces connectivities of the GiDMLOutput handle. These are the faces contour of each volume tetrahedra mesh.

The output faces are linear triangle elements (3 nodes).

Module attributes for the output faces

The module attributes for the output faces for the GiDML_Image2Mesh module are listed hereafter.

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

Name	Value type	Description	Possible values
GIDML_IMG2MESH_FACES_COLORS	GIDML_TYPE_INTEGER	This attribute indicates the color of each face (triangle).	The values of the colors corresponds to an id of the interface between the volumes interfaced. This id is an integer greater than the maximum volume color defined in the nput data inside the GIDML_IMG2MESH_INPUT_NODES_COLORS parameters vector.

Output edges

In case the mesh must be periodic, the contact elements (linear elements between periodic nodes) connecting the nodes of opposite faces are provided as edges in this structure. They are line elements of 2 nodes.The first node of each element is always the one with less coordinate in the corresponding direction.

Module attributes for output edges

The GiDML_Image2Mesh module is providing one module attribute for the output edges, indicating the relationship between periodic nodes and the axis the linked nodes are aligned with.

Name	Value type	Description	Possible values
GIDML_IMG2MESH_EDGES_CONTACT_AXIS	GIDML_TYPE_INTEGER	This attribute indicates the axis the edge is aligned with, considering the domain cartesian aligned.	The values can be 0, 1 or 2, corresponding to x, y and z axis.

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

Module compilation

Requirements

To compile this module you'll need the GiDML IO module

Compilation

Go inside the

`gid_mesh_library/gidml_image_2_mesh`

folder and edit the

`Makefile.linux`

to substitute following line:

`include $(GID_SRC_DIR)/make_opt_flags.mk`

with

```
FOPENMP = -fopenmp
CPPFLAGS_OPT = -O3
```

And then, in the command line:

`make -f Makefile.linux`

and you'll have the static library.

Example

A simple code example is presented in this section aiming to make more clear how to work with GiDML_Image2Mesh module.

The examples are written in c++ code.
In this example we are generating a mesh from a synthetic image of 2 colors (materials). The input data is hardcoded, so the 3D image is relatively small. It has to be considered that the 3D images definitions should come from real data (dicon images, etc...).

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_image_2_mesh.h"
#define NAME_OF_MY_PROGRAM "MY_PROGRAM" //identifier of who is creating
the input data
int main() {
  //in this example a synthetic small 3D image is used, of 5x5x5 pixels
  const double points_colors[24]={0,1,1,0,0,  0,1,1,0,0,  0,0,1,1,0,
0,0,1,1,0, 0,0,0,1,1,

0,1,1,0,0,  0,1,1,0,0,  0,0,1,1,0,  0,0,1,1,0, 0,0,0,1,1,

0,1,1,0,0,  0,1,1,0,0,  0,0,1,1,0,  0,0,1,1,0, 0,0,0,1,1,

0,1,1,0,0,  0,1,1,0,0,  0,0,1,1,0,  0,0,1,1,0, 0,0,0,1,1,

0,1,1,0,0,  0,1,1,0,0,  0,0,1,1,0,  0,0,1,1,0, 0,0,0,1,1};

  //Create GiDMLInput Handle.
  const int n_dimension = 3; //space dimension of the data
  const char *module_name = GiDML_Image2Mesh_GetModuleName();
  const char *module_format_version =
GiDML_Image2Mesh_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

  //Set parameters
  //enter colors in GiDMLInput
  const int num_points = 125;// =5*5*5
  GiDML_IO_SetParameterVector(hdl_input,
"GIDML_IMG2MESH_INPUT_NODES_COLORS", points_colors, num_points);

  //enter num of points in GiDMLInput
  const int* num_ponits = {5,5,5};
```

```
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_NUMBER_POINTS_X",
num_ponits[0] );
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_NUMBER_POINTS_Y",
num_ponits[1] );
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_NUMBER_POINTS_Z",
num_ponits[2] );

  //enter spacings in GiDMLInput
  const int* spacing = {0.5,0.5,0.2};//note that is not needed for the
3 spacings to be the same.
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_SPACING_X", spacing
[0] );
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_SPACING_Y", spacing
[1]);
  GiDML_IO_SetParameter(hdl_input, "GIDML_IMG2MESH_SPACING_Z", spacing
[2]);

  //Enter origin in GiDMLInput
  const double* origin = {0.,0.,0.}
  GiDML_IO_SetParameter(hdl_input,
"GIDML_IMG2MESH_ORIGIN_COORDINATE_X", origin[0]);
  GiDML_IO_SetParameter(hdl_input,
"GIDML_IMG2MESH_ORIGIN_COORDINATE_Y", origin[1]);
  GiDML_IO_SetParameter(hdl_input,
"GIDML_IMG2MESH_ORIGIN_COORDINATE_Z", origin[2]);

  //periodicity
  const int must_be_periodic=1;
  GiDML_IO_SetParameter(hdl_input,"GIDML_IMG2MESH_PERIODIC_MESH",
must_be_periodic);

  //interface triangles in the final mesh
  const int return_interface_triangles = 0; //by setting this parameter
to 1, the triangles can be obtained in the output data
  GiDML_IO_SetParameter(hdl_input,
"GIDML_IMG2MESH_RETURN_INTERFACE_TRIANGLES",
return_interface_triangles);

  //minimum element size
  const double min_elem_size = 0.1;
  GiDML_IO_SetParameter(hdl_input,
"GIDML_IMG2MESH_MINIMUM_ELEMENT_SIZE", min_elem_size);

  //Check the that the Input format is correct.
  int error_returned = GiDML_Image2Mesh_CheckConsistency(hdl_input);

  //GiDMLOutput Handle.
  GiDMLOutput_Handle hdl_output = GiDML_IO_NewGiDMLOutputHandle();
  if ( error_returned == 0){
```

```
    //Call the mesher.
    error_returned = GiDML_Image2Mesh(hdl_input,hdl_output);
  }

  if ( error_returned ){
    const char *error_message = GiDML_Image2Mesh_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);

    //Get edges providing with the periocicity information
    const int num_of_edges = GiDML_IO_GetNumberOfEdges( hdl_output );
    int *edges_connectivity = NULL;
    fail=GiDML_IO_GetEdges( hdl_output,edges_connectivity );
  }

  //Delete data from GiDMLOutput
  GiDML_Image2Mesh_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 proprietary of this module. Any use of it involves the sign of an agreement with CIMNE.

Please, contact gidml@cimne.upc.edu if you are interested in integrating the GiDML_Image2Mesh module inside your software.