
# GiD Mesh Library: Ocree Tetrahedra 1.1.1 Mesher Module

# Introduction

This is the documentation of the module of the GiDMeshLibrary **GiDML_OctreeTetrahedraMesher**. 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_OctreeTetrahedraMesher** module is an unstructured volume mesh generator. It's an octree-based meshed which ensures geometrical features and model topology preservation. It gets a triangle mesh defining the contours of the volumes of the model as an input, and returns a tetrahedra mesh of them which represents the topology of the input data, but is not constrained to those triangles. This allows to use optimized meshes for defining the shape of the contours (typically visualization meshes), and decouple the probability of success of the mesher from the quality of triangles of the contours.

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

Its main characteristics are:

- **Robustness**: the mesher has been designed based on its robustness. The algorithm can generate the tetrahedra volume mesh independently from the quality of the input contours of the volumes and the meshing parameters.
- **Speed**: it is a very fast mesher (it can generate more than 10 millions of tetrahedra per minute), also taking advantage on a parallel implementation following shared memory paradigm.
- Mesh of **non-watertight geometries**: this aspect, together with the robustness of the mesher, reduces drastically the time needed to generate a calculation mesh considering the almost no need of CAD cleaning operations in the original model.
- Suitable for **body-fitted and embedded** meshes: it can generate body-fitted meshes and embedded ones. In that case, a field of the distances from each node of the final mesh to the contours of the volumes is returned together with the mesh. The embedded approach is extremely robust.
- Mesh the **whole model at a time**: the input of the mesher are the triangles meshes of the contours of the volumes involved in the model. If more than one volume share some contour, the resulting mesh will represent that topology properly.

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

This document focus on how to use the module, rather than how the module works internally (which can be consulted in the thesis).

## Mesh size information

The GiDML_OctreeTetrahedraMesher is designed to be able to generate a mesh with the less parameters as possible, including the option not to provide with any mesh size. However, it is well known that any simulation has its own requirements in terms of mesh size, refinement or sizes transitions.

The module offers different ways to define the required mesh size:

- General mesh size: a general mesh size can be set which will be applied to the whole model as the maximum one. It is assigned using the parameter "GIDML_OCTREEMESHER_GENERAL_MESH_SIZE" (see **Input parameters**).
- Mesh size entities: a specific mesh size can be assigned to any mesh entity from the input data (being it an entity defining the model, or an entity only to provide with mesh information). It can be assigned to nodes, edges, faces or elements. See the corresponding Attributes defined in Input data for each kind of entity.
- Sizes for the volumes: a specific mesh size can be assigned to the mesh of each volume of the domain. For this purpose, the parameters vector "GIDML_OCTREEMESHER_VOLUMES_SIZES" is used (see **Input parameters**).

In case that more than specific mesh size has been assigned to the same region in space, the mesher is always getting the smaller one.

Note that this module generates the mesh from an octree structure, which is homogeneous. This implies that is hard to obtain elements with edges larger than the minimum size of the bounding box of the model.

Apart from defining specific mesh sizes in given regions of the domain, it is also interesting to control somehow the sizes transitions between areas with smaller sizes to areas with larger ones. For this purpose the parameter "GIDML_OCTREEMESHER_SIZE_TRANSITION_FACTOR" is used (see **Input parameters**).

## Forced nodes and edges

One can define specific positions in space to have a node in the output mesh (forced node). For this purpose, the module attribute "GIDML_OCTREEMESHER_NODES_FORCED" is used in the input nodes (see Module attributes for the input nodes). Note that this forced nodes can be part of the boundary of a volume, or inner to it.

It is also possible to define specific line paths (defined with line elements) to have a connected path of edges in the final mesh too (forced edges). It is done introducing edges in the input data (see Input edges). Note that this forced edges can be part of the boundary of a volume, or inner to it.

This is essential if the program linking with the GiDML module wants to have a relationship between curves of the model and edges of the output mesh.

## Preserve geometrical features

One of the important aspects of a mesh generator is to preserve the geometrical features of the input model, specifically the sharp edges (ridges) and corners.

If we know them a priori, we can always set them as forced nodes or edges in the input data (see Forced nodes and edges), but there is also de possibility to indicate the maximum angle below which a dihedral angle between adjacent faces in the input data is considered a ridge. Actually, it is not done defining the angle, but the cosinus of it. Is is done with the input parameter "GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES".

## Topology preservation

The GiDML_OctreeTetrahedraMesher module preserves the topology of the model to be meshed. This means:

- There are a set of connected tetrahedra (and only one set) for each one of the volumes from the input data.
- The triangle skin mesh of the tetrahedra of each volume is a manifold watertight set of triangles.

If we consider the typical '**constrained/no constrained**' condition used in other meshers, we can say that this mesher is partially constrained. This means that the skin of each volume meshes generated is not matching with the triangles defining the contours of the volumes in the input data. However, if the input data is properly set, we can obtain a set of triangles in the output mesh (skin of volume mesh) corresponding to a set of triangles from the input data.

This is useful, for instance, if we want to get the triangles of the final mesh comming from a given surface from the input data. To do so, the contour edges of the set of triangles from the input data must be set as forced

edges (see Forced nodes and edges), and then the generated mesh will have a set of triangles enclosed inside them, corresponding to the input ones. This is explained in more detail in the Section 3.1 of the PhD thesis "Robust volume mesh generation for non-watertight geometries", this mesher is based on.

As a main conclusion concerning the topology preservation, it is ensured that if the input data is comming from specific points, curves or surfaces, the corresponding nodes, line elements and triangles can be obtained in the generated mesh, so as there can be a one to one relationship with the input data and the output mesh.

## Working with non-watertight geometries

One of the main aspects of the GiDML_OctreeTetrahedraMesher module is that it can generate volume meshes from non-watertight definitions of the contours of the volumes of the model. We consider non-watertight definitions the ones which may have (in the definition of the volumes' contours):

- gaps
- faces overlappings

It is mainly achieved due to the advanced Ray Casting technique developed to solve the Point in Polygon (PiP) problem. That is, to detect is a specific point in space is inside or outside a volume, defined by its contours. The used Ray Casting technique, as well as some examples of its application is explained in more detail in the Section 4.3 of the PhD thesis "Robust volume mesh generation for non-watertight geometries", this mesher is based on.

At user level, one have to consider that if working with non-watertight geometries may be less robust than working with watertight ones. Of course, the size of the gaps or the distance between overlapping entities must be considerable smalles than the mesh desired size in that region, otherwise, the mesher will not be cappable to 'enclose' the volume mesh.

One main aspect for ensuring the success in the mesh generation of non-watertight geometries is to introduce a tolerance in the input data, corresponding to a higher bound of the approximate size of the gaps or distance between overlapping entities. This is done using the "GIDML_OCTREEMESHER_TOLERANCE_FOR_GAPS" parameter (see Input parameters).

# Module functions

All the functions provided by the module are described hereafter, and are declared in the header file gidml_octree_tetrahedra_mesher.h.

## Module and version information

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

### GiDML_OctreeTetrahedraMesher_GetModuleName

**Declaration:**
const char *GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher module is returned in a *const char** format. It is the value defined in the gidml_octree_tetrahedra_mesher.h file, under the *#define GiDML_OCTREE_TETRAHEDRA_MESHER_MODULE_NAME*.

### GiDML_OctreeTetrahedraMesher_GetModuleVersion

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

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

### GiDML_OctreeTetrahedraMesher_GetModuleFormatVersion

**Declaration:**
const char *GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher format is returned in a *const char** format. It is the value defined in the gidml_octree_tetrahedra_mesher.h file, under the *#define GiDML_OCTREE_TETRAHEDRA_MESHER_FORMAT_VERSION*.

### GiDML_OctreeTetrahedraMesher_GetVersion_Number

**Declaration:**
double GiDML_OctreeTetrahedraMesher_GetVersion_Number(const char* version);
**Definition:**
This function provides with a number (in double format) corresponding to the version provided by the GiDML_OctreeTetrahedraMesher_GetModuleFormatVersion and GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher_GetMinimumVersionToSaveInput

**Declaration:**
const char* GIDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher_GetMinimumVersionToSaveOutput

**Declaration:**
const char* GIDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher_CheckModuleAndVersion

**Declaration:**
int GiDML_OctreeTetrahedraMesher_CheckModuleAndVersion(const GiDMLIO_Handle hdl_gio);
**Definition:**
This function checks whether the module the data in the handle is for is the same as GiDMLOctreeTetrahedraMesher module, and if its format version is the same as the one in the module.This function is usefull 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 funcion.

**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_OctreeTetrahedraMesher_TransformGiDMLInputFromOlderVersion

**Declaration:**
int GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher module. This function is usefull 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_OctreeTetrahedraMesher_TransformGiDMLInputToOlderVersion

**Declaration:**
int GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher module. This function is usefull 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_OctreeTetrahedraMesher_TransformGiDMLOutputFromOlderVersion

**Declaration:**
int GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher module. This function is usefull 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_OctreeTetrahedraMesher_TransformGiDMLOutputToOlderVersion

**Declaration:**
int GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher 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_OctreeTetrahedraMesher

This is the function calling to the mesh generator itself.

**Declaration:**
int GiDML_OctreeTetrahedraMesher(const GiDMLInput_Handle hdl_gin, GiDMLOutput_Handle hdl_gout);
**Definition:**
This function is the unstructured tetrahedra mesher itself.
**Parameters:**
The function recieves 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_OctreeTetrahedraMesher_GetErrorString function (see GiDML_OctreeTetrahedraMesher_GetErrorStrinc ). If all the meshing process has finalized succesfully, it returns 0.

## GiDML_OctreeTetrahedraMesher_CheckConsistency

**Declaration:**
int GiDML_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher function (see GiDML_OctreeTetrahedraMesher). For instance: if there are no faces in the input datat defining the contours of the volumes, the module could not generate the mesh.
**Parameters:**
The function recieves the input data handle hdl_gin which is the candidate to be used for generating the mesh using the GiDML_OctreeTetrahedraMesher function.

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

## GiDML_OctreeTetrahedraMesher_GetErrorString

**Declaration:**
const char* GiDML_OctreeTetrahedraMesher_GetErrorString(const int error_id);
**Definition:**
This function provides with the meaningful information corresponding to an error_id returned by the functions GiDML_OctreeTetrahedraMesher, GiDML_OctreeTetrahedraMesher_CheckConsistency.
**Parameters:**
The function recieves 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 octree based mesher" |
| 2 | "There is no GiDMLInput handle" |
| 3 | "There are no nodes inside GiDMLInput handle" |
| 4 | "There are no faces inside GiDMLInput handle" |
| 5 | "Error in octree based mesher processing the input data" |
| 6 | "Error refining octree with user sizes" |
| 7 | "Error refining octree considering forced entities" |
| 8 | "Error coloring octree nodes with raycasting technique" |
| 9 | "Error refining octree for preserving topology" |
| 10 | "Error coloring tetrahedra" |
| 11 | "Error filling GiDMLOutput structure" |
| 12 | "Error preserving geometrical features" |
| 13 | "Error fitting surfaces to mesh" |
| 14 | "Error collapsing small elements after generating mesh" |
| 15 | "Error in make-up and smoothing operations after generating mesh" |
| 16 | "Error computing distances and coloring nodes in embedded mesh" |
| 17 | "Unknown error in octree based mesher" |
| 18 | "There are some forced points inner to a volume which are onto its interface" |

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

## GiDML_OctreeTetrahedraMesher_DeleteGiDMLOutputContent

**Declaration:**
int GiDML_OctreeTetrahedraMesher_DeleteGiDMLOutputContent(GiDOutput_Handle hdl_gout);
**Definition:**
This function deletes the content of the GiDMLOutput structure corresponding to the GiDMLOutput_Handle created by the GiDML_OctreeTetrahedraMesher module. As the module has created these data and filled the GiDoutput 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

This section describes the input data for the GiDML_OctreeTetrahedraMesher module.

## Spatial dimension

The GiDML_OctreeTetrahedraMesher 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_OctreeTetrahedraMesher 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
- the ones belonging to some mesh size entity
- the ones belonging to some forced edge

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 back

In the connectivities information of edges, faces and 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).

The GiDML_OctreeTetrahedraMesher module is not considering any **user attribute** for nodes from the input data. If they exists, the module is simply doing nothing with them, but it works normally.

### Module attributes for the input nodes

The module attributes for the input nodes for the GiDML_OctreeTetrahedraMesher 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_OCTREEMESHER_NODES_FORCED** | GIDML_TYPE_INTEGER | This attribute indicates if the node is a forced node in the final mesh. | If the value is 0, the node is not a forced node, otherwise, it is, so it will have a node in the same position in the output mesh. Note that in this case, the output data will have the attribute GIDML_OCTREEMESHER_NODES_FORCED_FROM_INPUT in its nodes. |
| **GIDML_OCTREEMESHER_NODES_SIZES** | GIDML_TYPE_DOUBLE | This attribute correponds to the mesh size assigned to the node. | If the value is 0.0, the size is not taken into account. |

Note that these attributes are not mandatory for the mesh generator. By default, if no attribute is set:

- it is considered that there are no forced points
- no specific size assigned to any node

## Input edges

All the edges involved in the input data must be introduced in the edges connectivities of the GiDMLInput handle, following the standard API functions of GiDML_IO module (see GiDML IO module). The GiDML_OctreeTetrahedraMesher works only with **Line elements of 2 nodes.**

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

- the forced edges to be preserved by the mesher
- the edges used to provide with mesh size information

The connectivities of the edges 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).

### Module attributes for the input edges

The module attributes for the input edges for the GiDML_OctreeTetrahedraMesher module are listed hereafter.

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

| Name | Value type | Description | Possible values |
|---|---|---|---|
| **GIDML_OCTREEMESHER_EDGES_COLORS** | GIDML_TYPE_INTEGER | This attribute indicates if the edge is part of a boundary of a volume (in this case, they must belong to some triangle contour of a volume), inner to a volume, or only to give mesh size information (not a forced edge for the final mesh). | If the value is 0 (value by default), the edge is a forced edge part of a boundary. Values 'i' greater than 0 indicate it is a forced edge inner to the volume 'i'. Value equal to -1 indicates the edge is only to give mesh size information. By default, all the edges are part of the boundary (0). |
| **GIDML_OCTREEMESHER_EDGES_SIZES** | GIDML_TYPE_DOUBLE | This attribute correponds to the mesh size assigned to the edges. | If the value is 0.0, the size is not taken into account. |

Note that these attributes are not mandatory for the mesh generator. By default, if no attribute is set:

- all the edges are in the boundaries of the volumes, and must belong to some triangle from the boundary
- no specific size assigned to any edge.

### User attributes for input edges

The GiDML_OctreeTetrahedraMesher module considers the user attributes that the input edges may have. They can have any name and value type, and there can be any number of them.

In case the forced edges have some user attribute, the attribute is transmitted to the nodes and edges of the output mesh which are onto the forced edges. This may be useful in cases where it is required to know the input entities some output entity comes from; for instance, the line elements in the output mesh corresponding to some line of the input geometry.

Note that, for using the API functions, all these user attributes are of entity type *GIDML_EDGE* and type of attribute *GIDML_USER_ATTRIBUTE*.

## 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_OctreeTetrahedraMesher works only with **3 nodes Triangle** faces in the input data.

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

- the triangles defining the contours of the volumes of the model to be meshed
- the triangles used to provide with mesh size information

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).

### Module attributes for the input faces

The module attributes for the input faces for the GiDML_OctreeTetrahedraMesher 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_OCTREEMESHER_FACES_INTERFACES_1** and **GIDML_OCTREEMESHER_FACES_INTERFACES_2** | GIDML_TYPE_INTEGER | These two attributes indicate the two ids of the volumes the face is interfacing. Exterior is considered as volume '0'. The ids of the volumes must be correlative, and beginning by 1. | Faces interfacing two times the same volume are considered as faces inner to the volume. Faces interfacing two times the volume '-1' indicate that they are only for giving mesh size information: they are not part of the domain. |
| **GIDML_OCTREEMESHER_FACES_EMBEDDED** | GIDML_TYPE_INTEGER | This attribute indicates if the faces correspond to embedded or body-fitted boundaries. Embedded boundaries are only considered for computing distances to them from the final mesh nodes. | If the value is 0 (default value), the face is part of body-fitted boundary, otherwise it is embedded. |
| **GIDML_OCTREEMESHER_FACES_SIZES** | GIDML_TYPE_ | This attribute correponds to the mesh size assigned to the faces. | If the value is 0.0, the size is not taken into account. |

| | | | |
|---|---|---|---|
| | DOUBLE | | |

Note that these attributes are not mandatory for the mesh generator. By default, if no attribute is set:

- all the faces are considered as in the boundary of one unique volume, so they interface the volume and the exterior of the domain.
- no specific size assigned to any face.

### User attributes for input faces

The GiDML_OctreeTetrahedraMesher module considers the user attributes that the input faces may have. They can have any name and value type, and there can be any number of them.

In case the input faces have some user attribute, the attribute is transmitted to the nodes, edges and faces of the output mesh which lay onto them. This may be useful in cases where it is required to know the input entities some output entity comes from; for instance, the triangles of the output mesh boundary of a volume corresponding to some set of faces (surface) of the input geometry.

Note that, for using the API functions, all these user attributes are of entity type *GIDML_FACE* and type of attribute *GIDML_USER_ATTRIBUTE*.

## Input elements

The GiDML_OctreeTetrahedraMesher only get input volume elements to provide mesh size information for the final mesh. 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).

### Module attributes for input elements

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

| Name | Value type | Description | Possible values |
|---|---|---|---|
| **GIDML_OCTREEMESHER_ELEMENTS_SIZES** | GIDML_TYPE_DOUBLE | This attribute correponds to the mesh size assigned to the elements. | If the value is 0.0, 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_OctreeTetrahedraMesher module.

## Input parameters

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

#### Parameters

The input scalar parameters for the GiDML_OctreeTetrahedraMesher 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_OCTREEMESHER_GENERAL_MESH_SIZE** | This is the general mesh size, which will be applied to the entities with no size assigned. | If the value is 0.0 (default value), the size is not taken into account. |
| **GIDML_OCTREEMESHER_QUADRATIC_TYPE** | This parameter indicates if the resultant mesh should be quadratic or not. The module only considers one type of quadratic allowed: nodes in center of edges (3 nodes line elements, 6 nodes triangles and 10 nodes tetrahedra). | If the value is 0 (default value) the resulting mesh is linear, and if it is 1 is quadratic. |
| **GIDML_OCTREEMESHER_SIZE_TRANSITION_FACTOR** | This is a double between 0 and 1 representing the sizes transition factor. The lower is the value, the more space the mesh will need to increase its size from the regions with small elements to the ones with large ones. It is analogous to the one used in the GiD meshing preferences. | The value must be between 0.0 and 1.0 (both included). 0.0 is taken if the value is lower than 0.0, and 1.0 if it is higher than 1.0.<br>Default value is 1.0. |
| **GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES** | This parameter represents the minimum cosinus of a dihedral angle over which an edge is considered as a sharp edge to be preserved by the mesher. | Its value is between -1 and 1. By default its value is -10, indicating that no edge must be considered as sharp. |
| **GIDML_OCTREEMESHER_INPUT_IS_WATERTIGHT** | This parameter indicates if the contours of the volumes of the model are watertight. If so, the mesh generator can perform some operations in a more efficient way.<br>Note that watertight means that there are no holes nor overlapping entities defining the contours of the volumes, but they can be uncoherently oriented (it is not needed to be all of them oriented towards the inner or outer part of the volume). | If the value is 0, the geometry is not watertight, and if it is 1, it is so.<br>The value equal to -1 (the default value) means that the mesher will check itself if the boundaries are watertight or not before beginning the meshing process. |
| **GIDML_OCTREEMESHER_TOLERANCE_FOR_GAPS** | This is the minimum distance to be taken into account for non-watertight input geometries (gaps and overlappings in contour domains). Gaps smaller than this value, or overlaps closer than it are skipped when generating the mesh. | It must be a positive value. If 0.0 is set (default value), a value equal to '*s*/1000' is taken, where *s* is the mimimum between *GIDML_OCTREEMESHER_GENERAL_MESH_SIZE* and the diagonal of the bounding box of the input geometry. |
| **GIDML_OCTREEMESHER_MAX_RELATIVE** | This is the maximum relative chordal error (chordal error divided by the element size) allowed for the mesh generation. | It is a positive value. By default is -1, which means that this parameter is not taken into account. |

| | | |
|---|---|---|
| **_CHORDAL_ERROR** | | |
| **GIDML_OCTREEMESHER_MAX_CHORDAL_ERROR** | This is the maximum chordal error allowed for the mesh generation. | It is a positive value. By default is -1, which means that this parameter is not taken into account. |
| **GIDML_OCTREEMESHER_ONLY_GET_INNER_TETRAHEDRA** | This parameter is used to get only the inner tetras of the volumes. If this parameter is set, the mesher is not performing the surface fitting nor the geometrical features preservation. This parameter is specially useful for connecting this volume mesher with other meshers in the boundary regions. Note that the resulting mesh (if this parameter is set) is not body-fitted, and all the elements are completely inside the volumes. | The value of the parameter is the number of layers of elements not to be considered from the interface to the inner part of the volumes.<br>By default it is 0, so the regular mesh (with elements within the contour) is obtained. |
| **GIDML_OCTREEMESHER_DELETE_OUTER_EMBEDDED_ELEMENTS** | This variable only affects the embedded meshes. If this variable is set, the elements with all its nodes out of the domain won't be written in the output mesh. It is usefull in case where those elements are not processed by the solver, so memory can be saved. | If its value is 1, the outer elements are not reuturned by the mesher.<br>Its default value is 0, which is the case where all the elements are returned in the output mesh. |
| **GIDML_OCTREEMESHER_PRIORIZE_OUTER_PART_WHEN_COLORING** | This parameter only take sense for non-watertight geometries. If this parameter is set, in the ray casting process for the nodes coloring, the nodes before the first intersection and after the last intersection of the coloring rays are set as outer automatically. It may be usefull when complex non-watertight geometries are defining the domain, and the outer skin of it has no gaps (or really small ones). | By default this value is 0, so the coloring is done in the standard way. |
| | | |

### Parameters vectors

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

| Name | Value type | Description | Possible values |
|---|---|---|---|
| **GIDML_** | GI | These values are the mesh size for each of the volumes of the model. The size in 'ivol' position | If the size is 0.0, it is not considered. If the size value is negative, it means that the |

| | | | |
|---|---|---|---|
| **OCTREEMESHER_VOLUMES_SIZES** | DML_TYPE_DOUBLE | corresponds to the 'ivol+1' identifier of the volume used when defining the volumes interfaced by the faces (remember that the 0 identifier is reserved for outer part of the domain). The dimension of the vector should be the number of volumes in the model. | mesh of that volume is not needed for the output (it won't be returned). It has to be considered that, although the final mesh of those volumes is not required, it may affect the sizes of the other volume meshes. |

# Output data

The **GiDMLOutput** structure returned by the GiDML_OctreeTetrahedraMesher function has basically the resulting mesh generated by the module. It is a tetrahedra volume mesh, triangles (contours of the volumes tetrahedra meshes). Line elements meshes may also be present, if there were forced edges in the input data, or geometrical features to be captured.

## 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_OctreeTetrahedraMesher works always in 3D (3 coordinates per node).
If the input data has some **user attribute** for edges or faces or edges, the nodes in the output data may also have them, if they are onto the corresponding input entities.

### Module attributes for the output nodes

The module attributes for the output nodes for the GiDML_OctreeTetrahedraMesher 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_OCTREEMESHER_NODES_FORCED_FROM_INPUT** | GIDML_TYPE_INTEGER | This attribute only take sense if there are forced nodes in the input data. In this case, it indicates the value of the corresponding GIDML_OCTREEMESHER_NODES_FORCED attribute of the input node it comes from. | If the value is 0, the output node is not comming from any input forced node.<br>In case it is greater that 0, it is the value of the GIDML_OCTREEMESHER_NODES_FORCED attribute of the input node it comes from. |
| **GIDML_OCTREEMESHER_NODES_DISTANCES** | GIDML_TYP | This attribute only take sense if there are some embedded boundary in the input data (indicated with the | The values are equal or greater than 0. |

| | | | |
|---|---|---|---|
| | E_DOUBLE | face attribute *GIDML_OCTREEMESHER_FACES_EMBEDDED*). The attribute indicates the minimum distance from each node to an embedded boundary. | |

## Output edges

All the line elements involved in the output data are provided in the edges' connectivities of the GiDMLOutput handle. These are the edges coming from forced edges (in the input data), or result of geometrical features preserving.

The output edges are linear line elements (2 nodes) or quadratic ones (3 nodes) depending on the *GIDML_OCTREEMESHER_QUADRATIC_TYPE* parameter from the input data.

The GiDML_OctreeTetrahedraMesher module is not returning any module attribute for edges from the output data.

If the input data has some **user attribute** for edges or faces, the edges in the output data may also have them, if they are onto the corresponding input entities.

## 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) or quadratic ones (6 nodes) depending on the *GIDML_OCTREEMESHER_QUADRATIC_TYPE* parameter from the input data.

The GiDML_OctreeTetrahedraMesher module is not returning any module attribute for faces from the output data.

If the input data has some **user attribute** for faces, the faces in the output data may also have them, if they are onto the input ones.

## 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 for the volumes of the model.

The output elements are linear tetrahedra (4 nodes) or quadratic ones (10 nodes) depending on the *GIDML_OCTREEMESHER_QUADRATIC_TYPE* parameter from the input data.

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

### Module attributes for the output elements

If the input data have interfaces information referred to the input faces (faces attributes *GIDML_OCTREEMESHER_FACES_INTERFACES_1* and *GIDML_OCTREEMESHER_FACES_INTERFACES_2*), the elements will have the following attribute with the id (color) of the volume the tetrahedra is into.

| Name | Value type | Description | Possible values |
|---|---|---|---|
| **GIDML_OCTREEMESHER_ELEMENTS_COLORS** | GIDML_TYPE_INTEGER | This attribute correponds to the color of the volume the tetrahedron is into. | The color is the id used in the input data indicating the volumes interfaced by each face ( *GIDML_OCTREEMESHER_FACES_INTERFACES_1 and GIDML_OCTREEMESHER_FACES_INTERFACES_2* face attributes). |

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

# Examples

Some simple code examples are presented in this section aiming to make more clear how to work with GiDML_OctreeTetrahedraMesher module.

The examples are written in c++ code.

## 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 hardcoded 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_octree_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
not 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_OctreeTetrahedraMesher_GetModuleName();
  const char *module_format_version =
GiDML_OctreeTetrahedraMesher_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_OCTREEMESHER_GENERAL_MESH_SIZE", general_size);

  //indicate a minimum dihedral angle to be preserved, to get the sharp
edges of the cube in the final mesh.
  const double min_cos_for_sharp_edges = -0.5; // this value (cosinus
of -120 degrees) ensures that the dihedral angles of 90 degrees
                                                 // between the cube
faces are preserved.
  GiDML_IO_SetParameter(hdl_input,"
GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES",
min_cos_for_sharp_edges);


  //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_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher_DeleteGiDMLOutputContent(hdl_output);

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

  return 0;
}
```

### Forced edges

Note that in this example, we have ensured to preserve the sharp edges of the cube in the final mesh adding a parameter defining the minimum cosinus of the dihedral angle to be preserved (GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES).

For this case (where we know a priori the edges we want to be preserved, we could also define the sharp edges of the cube as forced edges. These are the 12 edges of the cube. If so, we should add to the input data the following (and then it would not be needed to set the dihedral angle parameter):

C++ code of the example

```
  const int forced_edges_conectivities[24]=
{0,1,1,2,2,3,3,0,0,4,1,5,2,6,3,7,4,5,5,6,6,7,7,4};

  //Edges
  const int number_of_edgses = 12;
  const ElemType edges_element_type = GID_LINE_ELEMENT;
  const int nnode_edges = 2;
  GiDML_IO_SetEdges(hdl_input, number_of_edgses ,
forced_edges_conectivities, edges_element_type , nnode_edges );
//forced edges data
```

## Mesh of two connected volumes

In this example, the model to be meshed consists in two cubes sharing one of its faces. The edges contour of the common face to both cubes will be marked as forced edges in the input data.

In this example it can be seen how to include in the input data the volumes interfaced by each face of the input data.

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_octree_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[36]=
{0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,10,0,10,10,
                                  20,0,0,20,10,0,20,0,10,20,10,10};
  const int faces_conectivities[66]=
{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
,

1,8,2,8,9,2,1,8,5,8,10,5,8,9,10,9,11,10,2,9,6,9,11,6,5,10,6,10,11,6};

  //Create GiDMLInput Handle.
  const int n_dimension = 3; //space dimension of the data
  const char *module_name =
GiDML_OctreeTetrahedraMesher_GetModuleName();
  const char *module_format_version =
GiDML_OctreeTetrahedraMesher_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 = 12;
  GiDML_IO_SetNodesCoords(hdl_input, number_of_nodes, n_dimension,
number_of_nodes,coordinates); //nodes data

  //Faces
  const int number_of_faces = 22;
  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
  //topology information, indicating which  volumes is interfacing each
face
  const int* vol_interfaced_1 =
{1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2};
  const int* vol_interfaced_2 =
{0,0,2,2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  //Note that the triangle faces in pos 2 and 3 (beginning from 0),
which are the faces 1,2,5 and 2,5,6 are interfacing volumes 1 and 2.
  GiDML_IO_SetAttribute(hdl_input,"
GIDML_OCTREEMESHER_FACES_INTERFACES_1",vol_interfaced_1 ,GIDML_FACE,
GIDML_MODULE_ATTRIBUTE, GIDML_TYPE_INTEGER);
  GiDML_IO_SetAttribute(hdl_input,"
```

```
GIDML_OCTREEMESHER_FACES_INTERFACES_2",vol_interfaced_2 ,GIDML_FACE,
GIDML_MODULE_ATTRIBUTE, GIDML_TYPE_INTEGER);


  //Edges
  const int forced_edges_conectivities[8]={1,2,2,6,6,5,5,1};//these are
the edges contour of the common face to both cubes
  const int number_of_edgses = 4;
  const ElemType edges_element_type = GID_LINE_ELEMENT;
  const int nnode_edges = 2;
  GiDML_IO_SetEdges(hdl_input, number_of_edgses ,
forced_edges_conectivities, edges_element_type , nnode_edges );
//forced edges data


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

  //indicate a minimum dihedral angle to be preserved, to get the sharp
edges of the cube in the final mesh.
  const double min_cos_for_sharp_edges = -0.5; // this value (cosinus
of -120 degrees) ensures that the dihedral angles of 90 degrees
                                                 // between the cube
faces are preserved.
  GiDML_IO_SetParameter(hdl_input,"
GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES",
min_cos_for_sharp_edges);


  //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_OctreeTetrahedraMesher_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_OctreeTetrahedraMesher_DeleteGiDMLOutputContent(hdl_output);

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

  return 0;
}
```

### Different sizes in each volume

In the case that a different mesh size for each volume is required (for instance, sizes 0.8 and 3), the following parameters vector should be included in the input data:

```
  const int number_of_volumes = 2;
  const double* volumes_sizes = {0.8,3.0}
  GiDML_IO_SetParameterVector(hdl_input,
"GIDML_OCTREEMESHER_VOLUMES_SIZES", volumes_sizes ,
number_of_volumes);
```

### Entities only to give mesh size information

In this example we will include in the input data some mesh entity only to provide with mesh size information, not being part of the boundary of the mesh.

Specifically, we will include one edge from the mid point of a cube to the mid point of the other, with a mesh size equal to 0.2.

Note that for including this information, we need to add two new nodes and one new edge. The faces definitions remains the same. Only the parts changing from the previous example are included in the following code block.

```
  const double coordinates[42]=
{0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,10,0,10,10,
                                20,0,0,20,10,0,20,0,10,20,10,10,
                                5,5,5,15,5,5};

   //Nodes
  const int number_of_nodes = 14;
```

```
  GiDML_IO_SetNodesCoords(hdl_input, number_of_nodes, n_dimension,
number_of_nodes,coordinates); //nodes data

//Edges
  const int forced_edges_conectivities[8]={1,2,2,6,6,5,5,1,12,13};
//these are the edges contour of the common face to both cubes
  const int number_of_edgses = 5;
  const ElemType edges_element_type = GID_LINE_ELEMENT;
  const int nnode_edges = 2;
  GiDML_IO_SetEdges(hdl_input, number_of_edgses ,
forced_edges_conectivities, edges_element_type , nnode_edges );
//forced edges data

  const int* edges_colors = {0,0,0,0,-1}; //Note that the first four
edges are marked as '0', as they are forced ones part of the boundary
of the model.
                                          //The '-1' value indicates
that the edge is only to provide with size information.
  GiDML_IO_SetAttribute(hdl_input,"GIDML_OCTREEMESHER_EDGES_COLORS",
edges_colors ,GIDML_EDGE, GIDML_MODULE_ATTRIBUTE, GIDML_TYPE_INTEGER);

  const int* edges_sizes = {0.,0.,0.,0.,0.2};
  GiDML_IO_SetAttribute(hdl_input,"GIDML_OCTREEMESHER_EDGES_SIZES",
edges_sizes ,GIDML_EDGE, GIDML_MODULE_ATTRIBUTE, GIDML_TYPE_DOUBLE);
```

### User attribute to identify a patch of triangles

In this example, we will see how to obtain the patch of triangles in the output mesh corresponding to some set of faces of the input data. It may be useful if we have some data assigned to some part of the contours of the domain in our program code, and we want to set these data to the corresponding mesh entities of the final mesh too.

In this example we will identify the triangles in the output mesh which are onto the common face between both cubes.

For this, we should add a user attribute in the input data, and get the corresponding one from the output data:

```
  //Attribute to be added to the Faces in the input data
  const int* identifier_of_faces_input_data =
{0,0,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
  GiDML_IO_SetAttribute(hdl_input,"MY_PROGRAM_FACE_IDENTIFIER",
identifier_of_faces_input_data ,GIDML_FACE, GIDML_USER_ATTRIBUTE,
GIDML_TYPE_INTEGER);
  //Note that the name of the attribute can be whatever. The
GiDML_OctreeTetrahedraMesher module does not know anything about it. It
just transmit it to the output data.

  //in this attribute from the output data we can get the triangles in
the output mesh corresponding to the shared face between both cubes.
The ones with value equal to '1'.
```

```
  int* identifier_of_faces_output_mesh  = (int*)GiDML_IO_GetAttribute
(hdl_output,"MY_PROGRAM_FACE_IDENTIFIER",GIDML_FACE);
```

## Embedded mesh

In this example we are generating a mesh of a cube of 10 units of length size, with an embedded contour inside (representing a hole), which is a concentric cube of 4 units of length. The contour mesh of both cubes is harcoded in this example.
The following code is providing to the mesher (as input) the contour mesh of the volume to be meshed body-fitted and the one defining the embedded contour (made of 3 node-triangles). A specific mesh size of 0.5 is provided in the faces defining the embedded volume.

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_octree_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[48]=
{0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,10,0,10,10,

3,3,3,7,3,3,7,7,3,3,7,3,3,3,7,7,3,7,7,7,7,3,7,7};
  const int faces_conectivities[72]=
{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
,

8,9,11,9,11,10,9,10,13,10,13,14,8,12,11,11,12,15,12,13,15,13,14,15,8,9,1
2,9,12,13,11,10,15,15,10,14};
  //note that the triangle faces defining the contour of the volume are
not 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_OctreeTetrahedraMesher_GetModuleName();
  const char *module_format_version =
GiDML_OctreeTetrahedraMesher_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 = 16;
  GiDML_IO_SetNodesCoords(hdl_input, number_of_nodes, n_dimension,
```

```
number_of_nodes,coordinates); //nodes data

  //Faces
  const int number_of_faces = 24;
  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

  const int* embedded_faces =
{0,0,0,10,0,0,10,10,0,0,10,0,0,0,10,10,0,10,10,10,10,0,10,10,

3,3,3,7,3,3,7,7,3,3,7,3,3,3,7,7,3,7,7,7,7,3,7,7};
  GiDML_IO_SetAttribute(hdl_input,"GIDML_OCTREEMESHER_FACES_EMBEDDED",
embedded_faces,GIDML_FACE, GIDML_MODULE_ATTRIBUTE, GiDML_TYPE_INTEGER);

  const double* faces_mesh_sizes = {0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,
0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,0.,

0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,
0.5,0.5,0.5,0.5,0.5,0.5};
  GiDML_IO_SetAttribute(hdl_input,"GIDML_OCTREEMESHER_FACES_SIZES",
faces_mesh_sizes ,GIDML_FACE, GIDML_MODULE_ATTRIBUTE,
GiDML_TYPE_DOUBLE);

  //Parameters
  //indicate a minimum dihedral angle to be preserved, to get the sharp
edges of the cube in the final mesh.
  const double min_cos_for_sharp_edges = -0.5; // this value (cosinus
of -120 degrees) ensures that the dihedral angles of 90 degrees
                                                // between the cube
faces are preserved.
  GiDML_IO_SetParameter(hdl_input,"
GIDML_OCTREEMESHER_MIN_COSINUS_FOR_SHARP_EDGES",
min_cos_for_sharp_edges);


  //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_OctreeTetrahedraMesher_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);

    //Nodes distances to embedded contour
    double* distances = (double*)GiDML_IO_GetAttribute(hdl_output ,"
GIDML_OCTREEMESHER_NODES_DISTANCES_FOR_EMBEDDED",GIDML_NODE);

    int *tetras_connectivity = NULL;
    int fail=GiDML_IO_GetElements( hdl_output,tetras_connectivity);
  }

  //Delete data from GiDMLOutput
  GiDML_OctreeTetrahedraMesher_DeleteGiDMLOutputContent(hdl_output);

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

  return 0;
}
```

Note that one could set the parameter GIDML_OCTREEMESHER_DELETE_OUTER_EMBEDDED_ELEMENTS to avoid receiving in the output data the elements with all its nodes in the outer part of the domain (in this case, inside the hole defined by the embedded contour).

In this case, the following lines should be added when filling the input data:

```
const int delete_outer_elems_in_embedded=1;
GiDML_IO_SetParameter(hdl_input,
"GIDML_OCTREEMESHER_DELETE_OUTER_EMBEDDED_ELEMENTS",
delete_outer_elems_in_embedded);
```

## 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_OctreeTetrahedraMesher module inside your software.