GiD Mesh Library: StructuredVolumeMesher 1.0.0 Module

GiDML_StructuredVolumeMesher module

GiDML_StructuredVolumeMesher module

Welcome to your documentation space!

- We've added some suggestions and placeholders. Everything is customizable.
- Get started with page templates:
  - Template - Product roadmap
  - Template - Error documentation
  - Check out Get the most out of your documentation space for more tips.

Search



Featured pages

Feature important pages with Content by label

Content by Label is used to display lists of pages, blog posts, or attachments that have particular labels. It's great for curating content you want to show on your overview page.

For example, you could display a list of all pages that have the label 'feature-shipped' and include the word 'Blueprint', or to list any pages with the label 'meeting-notes' that youve been mentioned in.

To add the Content by Label element:

1. When editing type /
2. Find Content by label in the dropdown
3. Select **Insert**

To edit the Content by Label element:

1. Select the placeholder. The floating toolbar appears.
2. Select **Edit**
3. Modify the parameters. Your changes are saved as you go.
4. Resume editing the page, and the panel closes.

Recently updated

Display a list of recently changed pages

To add the Recently updated element:

1. When editing type /
2. Find Recently updated in the dropdown
3. Select **Insert**

To edit the Recently updated element:

1. Select the placeholder. The floating toolbar appears.
2. Select **Edit**
3. Modify the parameters. Your changes are saved as you go.
4. Resume editing the page, and the panel closes.

Popular labels

Display a list of labels used often

Display a list of the most popular labels used throughout your Confluence site or within a space. A popular label is a label that has been added to many pages.

To add the Popular labels element:

1. When editing type /
2. Find Popular labels in the dropdown
3. Select **Insert**

To edit the Popular labels element:

1. Select the placeholder. The floating toolbar appears.
2. Select **Edit**
3. Modify the parameters. Your changes are saved as you go.
4. Resume editing the page, and the panel closes.

Introduction

This is the documentation of the module of the GiDMeshLibrary **GiDML_StructuredVolumeMesher**. 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

The **GiDML_StructuredVolumeMesher** module is a structured volume mesh generator. It gets quadrilateral mesh defining the contours of a volume sorted in a specific way and returns an hexahedra mesh of it.

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

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

Module functions

Module and version information

GiDML_StructuredVolumeMesher_GetModuleName

Declaration:
const char *GiDML_StructuredVolumeMesher_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_StructuredVolumeMesher module is returned in a *const char** format. It is the value defined in the gidml_structured_volume_mesher.h file, under the *#define GiDML_STRUCTURED_VOLUME_MESHER_MODULE_NAME*.

GiDML_StructuredVolumeMesher_GetModuleVersion

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

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

GiDML_StructuredVolumeMesher_GetModuleFormatVersion

Declaration:
const char *GiDML_StructuredVolumeMesher_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_StructuredVolumeMesher format is returned in a *const char** format. It is the value defined in the gidml_structured_volume_mesher.h file, under the *#define GiDML_STRUCTURED_VOLUME_MESHER_FORMAT_VERSION*.

GiDML_StructuredVolumeMesher_GetVersion_Number

Declaration:
double GiDML_StructuredVolumeMesher_GetVersion_Number(const char* version);
Definition:
This function provides with a number (in double format) corresponding to the version provided by the GiDML_StructuredVolumeMesher_GetModuleFormatVersion and GiDML_StructuredVolumeMesher_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_StructuredVolumeMesher_GetMinimumVersionToSaveInput

Declaration:
const char* GIDML_StructuredVolumeMesher_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_StructuredVolumeMesher_GetMinimumVersionToSaveOutput

Declaration:
const char* GIDML_StructuredVolumeMesher_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_StructuredVolumeMesher

This is the function calling to the mesh generator itself.

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

Returned value:
This function returns an error_id (in integer format) that can be processed by GiDML_StructuredVolumeMesher_GetErrorString function (see https://gidsimulation.atlassian.net/wiki/spaces/GM1/pages/2885681231). If all the meshing process has finalized successfully, it returns 0.

GiDML_StructuredVolumeMesher_CheckConsistency

Declaration:
int GiDML_StructuredVolumeMesher_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_StructuredVolumeMesher function (see https://gidsimulation.atlassian.net/wiki/spaces/GM1/pages/2885943321). For instance: if there are no faces in the input data defining the contours of the volumes, the module could not generate the mesh.
Parameters:
The function receives the input data handle hdl_gin which is the candidate to be used for generating the mesh using the GiDML_StructuredVolumeMesher function.

Returned value:
This function returns an error_id (in integer format) that can be processed by GiDML_StructuredVolumeMesher_GetErrorString function (see https://gidsimulation.atlassian.net/wiki/spaces/GM1/pages/2885681231). If all the data inside the GiDMLInput structure are coherent, it returns 0.

GiDML_StructuredVolumeMesher_GetErrorString

Declaration:
const char* GiDML_StructuredVolumeMesher_GetErrorString(const int error_id);
Definition:
This function provides with the meaningful information corresponding to an error_id returned by the functions GiDML_StructuredVolumeMesher, GiDML_StructuredVolumeMesher_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 structured volume mesher"
2	"There is no GiDMLInput handle"
3	"There are no nodes inside GiDMLInput handle"
4	"There are no faces inside GiDMLInput handle"
5	"Unknown error in structured volume mesher"
6	"User stop"
7	"Check orientation of volume"

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

GiDML_StructuredVolumeMesher_DeleteGiDMLOutputContent

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

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

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

Input data

Spatial dimension

The GiDML_StructuredVolumeMesher 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_StructuredVolumeMesher works always in 3D (3 coordinates per node).

Note that all the nodes involved in the input data are the ones belonging to the quadrilateral mesh of the contours of the volume.

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

The GiDML_StructuredVolumeMesher 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.

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_StructuredVolumeMesher works only with Quadrilateral faces in the input data. They can be linear, or quadratic (considering two types of quadratic: one node in middle of edges, and one node in middle of faces and center of element).

Note that all the faces defining the contours of the volume to be meshed must be entered, and these faces must represent a water-tight definition of the contours of the volume.

Considering the volume to be meshed has 6 recognizible sets of elements (surfaces), which are related two-by-two opposite in order to reach the structured mesh, the quadrilateral faces must be included in the faces vector in the following way:

- each set of faces (the faces of each surface) must be oriented coherently: that is, all of them towards the inner part or the outer part of the volume
- the faces of each set of faces must be toghether in the vector, one set after the other

The connectivities of the quadrilateral 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).

The example Structured hexahedra mesh of a cube can be checked in order to make clear how to include the contour faces of the volume in the input data.

Input parameters

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

Parameters

The input scalar parameters for the GiDML_StructuredVolumeMesher 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_VOLUME_MESH_SMOOTHING_METHOD	This is the quadratic type of mesh to be generated. Note that it should be the same quadratic type as the faces (quadrilateral) of the input data.	If the value is 0 (default value) which corresponds to linear elements. Value 1 corresponds to quadratic elements (with a node in center of edges). Value 2 corresponds to quadratic2 elements (with a node in center of edges and node in center of faces and element).
GIDML_STRUCTUREDMESHER_SURFACES_ORIENTATIONS	This is a vector of 6 components (one for each surface), indicating the orientation of each surface.	Value 0 indicates the surface is oriented towards the inner part of the volume. Value 1 indicates the surface is oriented towards the outer part of the volume.
GIDML_STRUCTUREDMESHER_NUMBER_OF_ELEMENTS_BY_SURFACE	This is a vector of 6 components (one for each surface), indicating the number of elements for each surface.	Values indicates the number of elements for each surface.
GIDML_STRUCTUREDMESHER_NUMBER_OF_LINES_ELEMENTS	This is a vector of 24 components (6 surfaces x 4 sides each surface) indicating the number of line elements present in each line of each surface. The first 4 components correspond to the lines of the first surface, the second 4 components for the second surface, and so on. The lines of each surface must be sorted coherently with the orientation of the surface (considering the right-hand law, closed loop of lines pointing towards the inner or the outer part of the volume.	Values are the number of line elements for each line of each surface.
GIDML_STRUCTUREDMESHER_LINES_NODES	This is a vector of 48 components (6 surfaces x 4 sides each surface x 2 nodes for each side). It vector includes two nodes id's for each line of each surface. It is not needed for these nodes to be sorted anyhow. The first 8 components correspond to each pair of nodes of the 4 lines of the first surface, and so on.	The values in this vector are the nodes id's of two nodes for each line.

The example Structured hexahedra mesh of a cube can be checked in order to make clear how to include the contour faces of the volume in the input data.

Output data

Output nodes

All the nodes involved in the output data are provided in the list of nodes coordinates of the GiDMLOutput handle. These are the nodes from the hexahedra generated by the mesher. Note that the first nodes appearing in this list are the ones from the input data which are constrained: the ones of the input boundaries of the volume.

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

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

Output elements

All the volume elements involved in the output data are provided in the elements' connectivities of the GiDMLOutput handle. These are the hexahedra generated by the mesher, of the quadratic type indicated in the input data. Note that the number of nodes of the hexahedra can be obtainedfollowing the standard API functions of GiDML_IO module (see GiDML IO module).

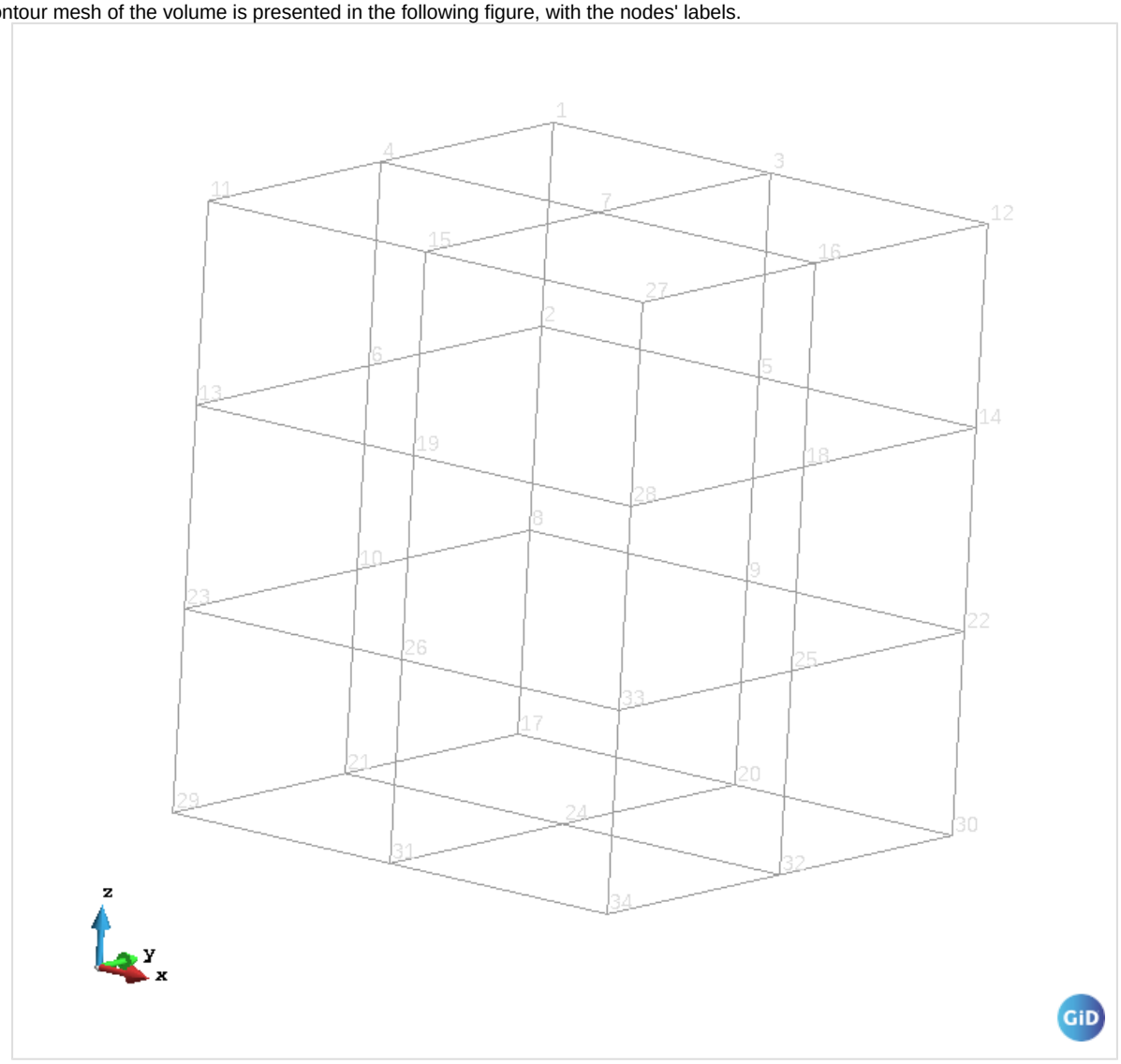
The GiDML_StructuredVolumeMesher module is not returning any **user attribute** in the output elements.

Example

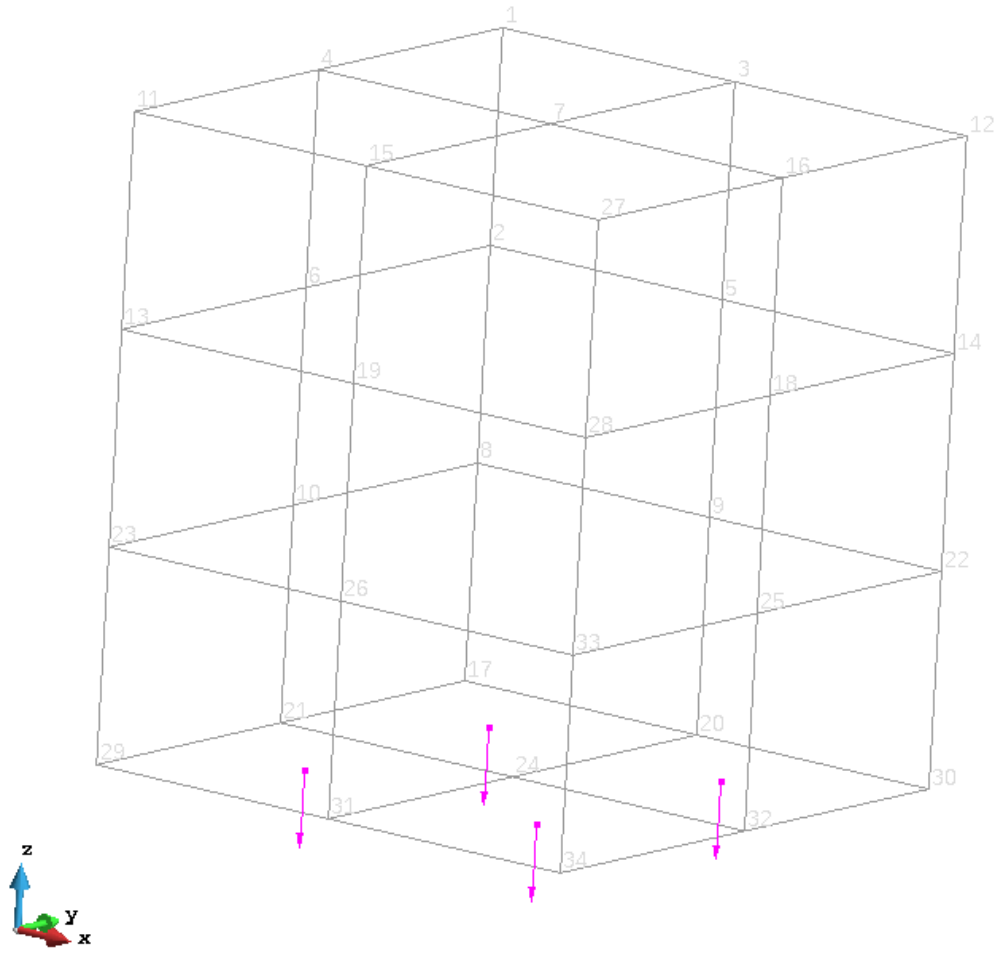
Structured hexahedra mesh of a cube

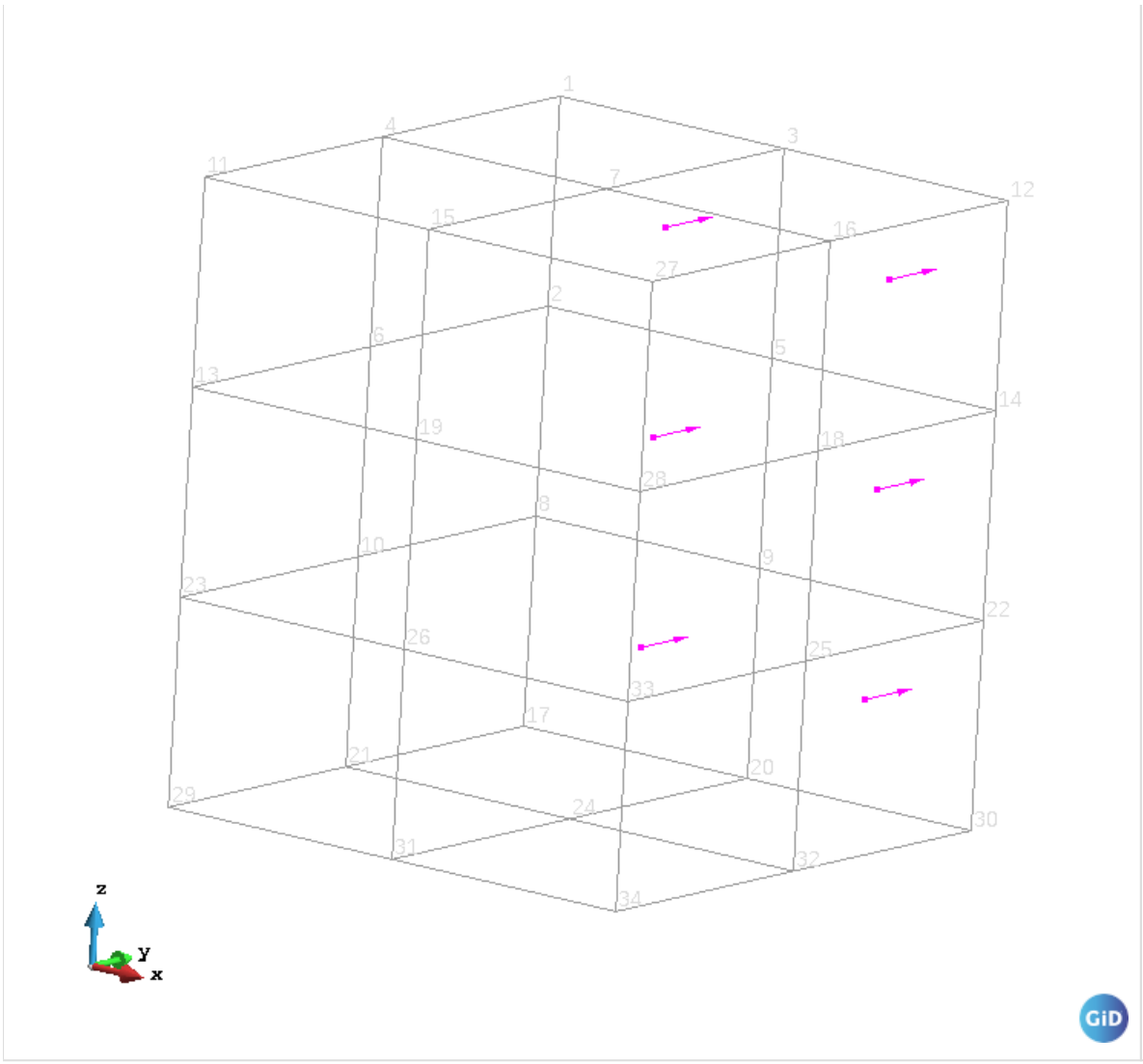
In this example we are generating an hexahedra structured mesh of a cube. The contour mesh of the cube is hardcoded in this example.

The contour mesh of the volume is presented in the following figure, with the nodes' labels.



In the following figures the orientation of each one of the faces of the cube is depicted:


1
4
3
11
7
12
15
16
27
2
6
5
13
14
19
18
28
8
10
9
23
22
26
25
33
17
21
20
29
24
30
31
32
34
z
y
x
z
y
GiD


1
4
3
11
7
12
15
16
27
2
6
5
13
14
19
18
28
8
10
9
23
22
26
25
33
17
21
20
29
24
30
31
32
34
z
y
x
z
y
GiD

C++ code of the example

The following code is providing to the mesher (as input) the contour mesh of the volume to be meshed (made of 4 node quadrilateral).

C++ code of the example

```
#include "gidml_io.h"
#include "gidml_structured_volume_mesher.h"
#define NAME_OF_MY_PROGRAM "MY_PROGRAM"
//identifier of who is creating the input data
int main(int argc,char** argv) {
  //in this example a cube which contour is defined by 32 faces (4 node-quadrilateral) is used
  //There are 34 nodes, so the coordinates vector has 102 (34*3) components.
  double coordinates[102]=
  {0,2,3, 0,2,2, 1,2,3, 0,1,3, 1,2,2, 0,1,2, 1,1,3, 0,2,1, 1,2,1, 0,1,1,
  0,0,3, 2,2,3, 0,0,2, 2,2,2, 1,0,3, 2,1,3, 0,2,0, 2,1,2, 1,0,2, 1,2,0,
  0,1,0, 2,2,1, 0,0,1, 1,1,0, 2,1,1, 1,0,1, 2,0,3, 2,0,2, 0,0,0, 2,2,0,
  1,0,0, 2,1,0, 2,0,1, 2,0,0};
  int faces_conectivities[128]=
  {20,23,30,28, 16,19,23,20, 19,29,31,23, 23,31,33,30,
  0,2,6,3, 2,11,15,6, 15,26,14,6, 3,6,14,10,
  0,1,5,3, 1,7,9,5, 7,16,20,9, 3,5,12,10, 5,9,22,12, 9,20,28,22,
  0,2,4,1, 2,11,13,4, 1,4,7,8, 4,13,21,8, 8,21,29,19, 8,19,16,7,
  15,11,13,17, 26,15,17,27, 17,13,21,24, 27,17,24,32, 21,29,31,24, 24,31,33,32,
  14,10,12,18, 226,14,18,27, 27,18,25,32, 18,12,22,25, 25,22,28,30, 32,25,30,33};
  //note that the quadrilateral faces defining the contour of the volume are oriented corresponding to
the orientation of the cube faces.

  //Create GiDMLInput Handle.
  const int n_dimension=3; //space dimension of the data
  const char* module_name=GiDML_StructuredVolumeMesher_GetModuleName();
  const char* module_format_version=GiDML_StructuredVolumeMesher_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=34;
  GiDML_IO_SetNodesCoords(hdl_input,number_of_nodes,coordinates); //nodes data

  //Faces
  const int number_of_faces=32;
  const ElemType faces_element_type=GID_QUADRILATERAL_ELEMENT;
  const int nnode_faces=4;
  GiDML_IO_SetFaces(hdl_input,number_of_faces,faces_conectivities,faces_element_type,nnode_faces);
//faces data

  //Parameters
  const double quadratic_type=0.;
  GiDML_IO_SetParameter(hdl_input,"GIDML_STRUCTUREDMESHER_QUADRATIC_TYPE",quadratic_type);

  const int nsurfaces=6;
  int original_versos_array[nsurfaces]={1,0,1,1,0,1};
  GiDML_IO_SetParameterVector(hdl_input,"GIDML_STRUCTUREDMESHER_SURFACES_ORIENTATIONS",
original_versos_array,6);

  int nelems_by_surface[nsurfaces]={4,4,6,6,6,6};
  GiDML_IO_SetParameterVector(hdl_input,"GIDML_STRUCTUREDMESHER_NUMBER_OF_ELEMENTS_BY_SURFACE",
nelems_by_surface,6);

  const int ndim_linenodes=48; //([6][4])*2 //first and last node of every line
  int lines_nodes_array[ndim_linenodes]=
  {28,18, 18,29, 29,33, 33,28,
  0,11, 11,26, 26,10, 10,0,
  28,10, 10,0, 0,16, 16,28,
  0,11, 11,29, 29,18, 18,0,
  26,11, 11,29, 29,33, 33,26,
  26,10, 10,28, 28,33, 33,26};
  GiDML_IO_SetParameterVector(hdl_input,"GIDML_STRUCTUREDMESHER_LINES_NODES",lines_nodes_array,
ndim_linenodes);

  const int ndim_lineelements=24;//[6][4]
  int number_of_lines_elements[ndim_lineelements]=
  {2,2,2,2,
  2,2,2,2,
  3,2,3,2,
  2,3,2,3,
  3,2,3,2,
  2,3,2,3
  };
  GiDML_IO_SetParameterVector(hdl_input,"GIDML_STRUCTUREDMESHER_NUMBER_OF_LINES_ELEMENTS",
number_of_lines_elements,ndim_lineelements);

  //End Parameters
```

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

    //GiDOutput Handle.
    GiDMLOutput_Handle hdl_output=GiDML_IO_NewGiDMLOutputHandle();

    //Call the mesher.
    if(error_returned == 0){
      error_returned=GiDML_StructuredVolumeMesher(hdl_input,hdl_output);
    }

    if(error_returned){
      const char* error_message=GiDML_StructuredVolumeMesher_GetErrorString(error_returned);
      //somehow show this error message...
    } else {
      //Get the hexahedra mesh
      const int number_of_nodes_in_hexahedra_mesh=GiDML_IO_GetNumberOfNodes(hdl_output);
      const int number_of_hexas=GiDML_IO_GetNumberOfElements(hdl_output);
      double* coords=NULL;
      GiDML_IO_GetNodesCoords(hdl_output,coords);
      int* hexas_connectivity=NULL;
      int fail=GiDML_IO_GetElements(hdl_output,hexas_connectivity);
    }

    //to write in a file the meshes of input and output that can be imported in GiD to see them
    //GiDML_IO_WriteGiDMLInput("input.gidml",hdl_input);
    //GiDML_IO_WriteGiDMLOutput("output.gidml",hdl_output);

    //Delete data from GiDOutput
    GiDML_StructuredVolumeMesher_DeleteGiDMLOutputContent(hdl_output);

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

    return 0;
  }
```

Terms of use of the module

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

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