Development of a Software Tool for Automation of Pre-Processing in Analysis using Finite Element/Volume Method

Best Practice Guide

Author: Filip Zaoral

Ostrava, November 2022

Table of Contents

Introduction

In terms of the workflow, numerical modelling can be divided into three phases: (i) pre-processing, (ii) solving of the governing equations and (iii) visualisation of the results. When using discretization methods such as Finite Element Method (FEM) or Finite Volume Method (FVM), pre-processing consists of preparation of geometry, its spatial discretization and application of boundary conditions. In most cases of engineering practice, we are dealing with complicated geometry for which it is impossible to create a computational mesh by trivial methods, and more comprehensive methods must be used. This text discusses the possibility of using the open-source tool Gmsh for mesh creation in a robust way, even for very complex geometries.

In the case of using open-source tools or software, separate independent libraries often must be used to do certain operations. For example, geometry preparation, finite element mesh generation, solving of the governing equations and results visualization are commonly performed using different libraries. In many cases different file formats are used for data exchange among those tools or software.

When dealing with complex geometries, designers and engineers are equipped with Computer-Aided Design (CAD) or more generally Computer Aided Engineering (CAE) software. For efficient and convenient preparation of the computational model, it is desirable that when transferring the model geometry via one of the commonly used exchange formats (. step, .iges, .stl, ...) to the finite element mesh generator, the information necessary for the definition of groups of finite elements and nodes are also transferred. During creation of geometry, it is much more efficient to assign a specific "flag" to those geometrical entities (volumes, areas), which will later be used for application of boundary conditions. This "flag" will then be used by a software for mesh generation to create separate group of elements and nodes belonging to this particular geometrical entity.

If the CAD software allows to assign names to selected geometric entities and then transfer them along with the geometry to the library for generation of finite element mesh via one of the exchange formats, then it is possible to assign the nodes and elements, belonging to these entities, to their corresponding groups automatically, i.e., without any user intervention, and then export them along with the finite element mesh for convenient application of the boundary conditions later.

This document discusses a software library under development, which provides automated workflow described above during the preparation of the finite element model.

Its input is, in addition to a set of parameters and options for creating and exporting a finite element mesh, the geometry of the model with named entities in a particular exchange format. The output of this library is then a file containing all the entities of the finite element mesh, in particular

the groups of mesh nodes and elements corresponding to the geometric entities named by the user during geometry pre-processing.

The document is structured into five chapters. The first chapter is dedicated to a general description of the Gmsh library, which has been chosen as the finite element meshing library on which the software tool will be built due to its extensive documentation and sophisticated application programming interface (API).

In the second chapter, a review of exchange formats for geometry transfer between CAE software packages was carried out. The basic requirement was the ability of the format to transfer the mentioned information about user named geometrical entities from the CAE software and the ability of the Gmsh library to read this information. Three formats that are available in Gmsh library (STEP, IGES and STL) were investigated.

The third chapter describes in detail the structure of the library under development, its operation, installation procedure and usage. Emphasis was placed on the description of the library input files. The format, content, and purpose of the GCF configuration file and the requirements for the geometry input files, the file name format and the necessary partitioning of the model into separate files.

In the fourth chapter, selected demonstration examples were described in detail to show the most important features of the developed library. All the test examples developed by the author are part of the library and can be found in the "examples" directory located in the root directory of the library.

The final chapter is dedicated to the investigation of parallel scalability of 2D and 3D meshing algorithms of Gmsh, measured in the form of strong scalability on multiple test problems.

1 Gmsh Library

Gmsh[1] is an open-source software package for geometry pre-processing, finite element discretization and post-processing, distributed under the GNU GPL version 2 or later (General Public License)[2]. The Gmsh library contains several tools for working with CAD geometry and post-processing of the created finite element mesh. To communicate with the library, both the Graphical User Interface (GUI) and the API via C[3], C++[4], Python[5], Julia[6] programming languages or the native language of the Gmsh library (ASCII files with the .geo extension) can be used.

The copyright holders of the Gmsh library since 1997 are Professor Christophe Geuzaine of the Department of Electrical Engineering and Computer Science at the University of Liege and Professor Jean-François Remacle of the Institute of Mechanics, Materials and Civil Engineering at the Catholic University of Leuven. Since the Gmsh library is distributed under the GNU GPL version 2 or later, in accordance with this license, distribution of the software tool under development is also only possible under the GNU GPL version 2 or later. The installation of the library itself is done by simply unpacking the compressed directory, which is available for download for Windows, Linux and MacOS operating systems as a stable and development version along with a software development kit (SDK), documentation and other information is available on the project website[7].

2 Data Exchange Formats

Of the plethora of exchange formats, those supported by the Gmsh library as model geometry input formats were explored, i.e. STEP, IGES and STL. The aim was to find out which of those three, if any, allow to write the arbitrarily chosen names of geometric entities, i.e. surfaces or volumes, in addition to the geometric information and then import them into the Gmsh library environment.

2.1 STEP Format

The name of the STEP format, characterized by the extension .step or .stp, is based on the English "STandard for Exchange of Product model data". The format itself is defined by ISO 10303 and its history dates to the mid-1980s. However, it is undoubtedly one of the most widely used formats for geometry exchange between CAE applications, and its development continues to this day. In addition to parametric geometry representation, the STEP format allows for transfer of much more information than just user defined geometrical entity names (e.g., surface finish information, dimensions, tolerances etc.)[8]. Fig. 1 shows a preview of the format, where the red box highlights the names of surfaces and curves entered by the user in the PTC Creo commercial software environment. However, all attempts to import these assigned names from the STEP format into the Gmsh library environment have been unsuccessful. Thus, it can be assumed that the Gmsh library does not yet fully support this feature in case of the STEP format.

```
#144=PLANE('62',#143);
#146=ORIENTED_EDGE('53',*,*,#145,.T.);
#148=ORIENTED_EDGE('HRANA_12',*,*,#147,.T.);
#149=ORIENTED_EDGE('49',*,*,#129,.F.);
#151=ORIENTED_EDGE('KOLEJNICE',*,*,#150,.T.);
#152=EDGE_LOOP('62',(#146,#148,#149,#151));
#153=FACE_OUTER_BOUND('62',#152,.F.);
⋮
#173=AXIS2_PLACEMENT_3D('',#170,#171,#172);
#174=PLANE('POTATO',#173);
#175=ORIENTED_EDGE('POTATO',*,*,#160,.F.);
#177=ORIENTED_EDGE('POTATO',*,*,#176,.F.);
#178=ORIENTED_EDGE('POTATO',*,*,#131,.F.);
#179=ORIENTED_EDGE('HRANA_12',*,*,#147,.F.);
#180=EDGE_LOOP('POTATO',(#175,#177,#178,#179));
#181=FACE_OUTER_BOUND('POTATO',#180,.F.);
#182=ADVANCED_FACE('POTATO',(#181),#174,.T.);
#183=CARTESIAN_POINT('',(5.E-1,1.E0,0.E0));
⋮
#186=AXIS2_PLACEMENT_3D('',#183,#184,#185);
#187=PLANE('LEFT',#186);
#188=ORIENTED_EDGE('LEFT',*,*,#133,.T.);
#189=ORIENTED_EDGE('LEFT',*,*,#176,.T.);
#190=ORIENTED_EDGE('LEFT',*,*,#165,.F.);
#192=ORIENTED_EDGE('LEFT',*,*,#191,.T.);
#193=EDGE_LOOP('LEFT',(#188,#189,#190,#192));
```

Fig. 1 Preview of a file in STEP format, created in the PTC Creo commercial software environment.

2.2 IGES Format

The IGES format, or "Initial Graphics Exchange Specification", is historically even older than the STEP format and its last version 5.3 was published in 1996[9]. However, it is still a very widespread exchange format among CAE applications, which allows parametric representation of geometry. However, unlike the STEP format, it does not carry information about a volume of the solid. All geometry is thus considered as a shell. This is not a limiting shortcoming, however, since the volume can be assigned in the Gmsh library environment to any group of surfaces that together form a "watertight" enclosure of a portion of space. The IGES format also supports arbitrary names for geometric entities as shown in Fig. 2, where a preview of the format can be seen with the red-highlighted user-selected names of faces of the model geometry, created in the Altair Hypermesh commercial software environment.

```
144,71,1,0,109,0,1,113;                                                  111P0000800
406,1,9Hlopatka.3                                                        113P0000801
128,1,1,1,1,1,1,1,0,0,6.02540239924561E-015,                             115P0000802
6.02540239924561E-015,0.00200568412894,0.00200568412894,                 115P0000803
1.24392639966331E-010,1.24392639966331E-010,0.00306327522761,            115P0000804
0.00306327522761,1.,1.,1.,1.217542057,0.20634899887106,                  115P0000805
-0.00153162048789,1.217542057,0.208354683,-0.00153162048789,             115P0000806
1.217542057,0.20634899887106,0.00153165461532,1.217542057,               115P0000807
0.208354683,0.00153165461532,6.02540239924561E-015,                      115P0000808
0.00200568412894,1.24392639966331E-010,0.00306327522761,0,1,117;         115P0000809
406,1,9Hlopatka.4                                                        117P0000810
128,21,16,3,3,1,1,1,0,0,0.00032663906819,0.00032663906819,               119P0000811
⋮
144,227,1,0,249,0,1,253;                                                 251P0001622
406,1,9Hkontakt.3                                                        253P0001623
128,4,10,2,2,1,1,0,0,0,5.47591682841935,5.47591682841935,                255P0001624
```

Fig. 2 Preview of an IGES file created in the Altair Hypermesh commercial software environment.

2.3 STL Format

The STL format, whose name is derived from the English "STereoLithography", is widely used in industry for rapid prototyping or 3D printing, or in medicine for visualizing CT (Computed Tomography) scans and allows to store data in ASCII or binary form. However, this format represents geometry by a finite number of triangles, determined by their vertices and normals, arranged in groups to which names can be assigned[10]. Thus, it does not allow for parametric description of the geometry. However, named groups can be used to transfer user-named surfaces, as shown in Fig. 3, where a preview of the format in the ASCII form can be seen with the group names marked in red, each representing a group of user-named surfaces. Unfortunately, the Gmsh library does not yet allow the group names (i.e., face names) to be read from STL files in binary form. Therefore, if the STL geometry files contain user-named faces, then the files must be in the ASCII form.

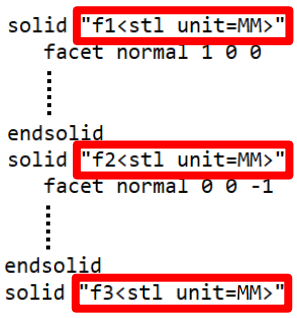

Fig. 3 Preview of an STL file created in the Ansys SpaceClaim commercial software environment.

2.4 Conclusion

From investigation of the possibilities of exchange formats (STEP, IGES, STL) used in Gmsh library, we could conclude that STEP format cannot be used to transfer information about named geometric entities to the Gmsh library environment at the moment. However, the remaining formats investigated, i.e. IGES and STL, can be used.

3 Structure, Functioning and Usage of the Library

The purpose of the developed library is to automatize the preparation of finite element computational models. In addition to the discretization of general geometry in IGES or STL format by finite elements/volumes, it also allows for automatic assignment of elements and nodes to groups, corresponding to geometric entities (surfaces and volumes), that have been assigned an arbitrary name not containing dots during pre-processing in a suitable CAD software package. The developed library is written in Python version 3.10.x.

In the root directory of the library there are:

- The "lib" directory,
- the "examples" directory with all the test problems created by the author,
- the "doc" directory containing entire documentation for the library,
- a launch file "GMTLaunch.py" to launch the tool,
- the "Gmsh.py" script to launch the Gmsh GUI conveniently,
- a configuration file with the extension .gcf (hereafter GCF), and
- the license file LICENSE.txt.

The "Lib" directory represents the library of the software tool itself. Along with the source code of the tool, it also contains the Gmsh library software development kit. The GCF file contains the user-specified settings for the finite element mesh generator. The LICENSE.txt file contains the exact wording of the GNU GPL version 2 license agreement.

The input model geometry file(s) are placed in the job working directory. The tool also stores its outputs here i.e., the resulting finite element mesh file in user-selected format and a .log (hereafter LOG) file containing informative, warning and error messages through which the tool communicates with the user. The structure of the library is illustrated in Fig. 4.

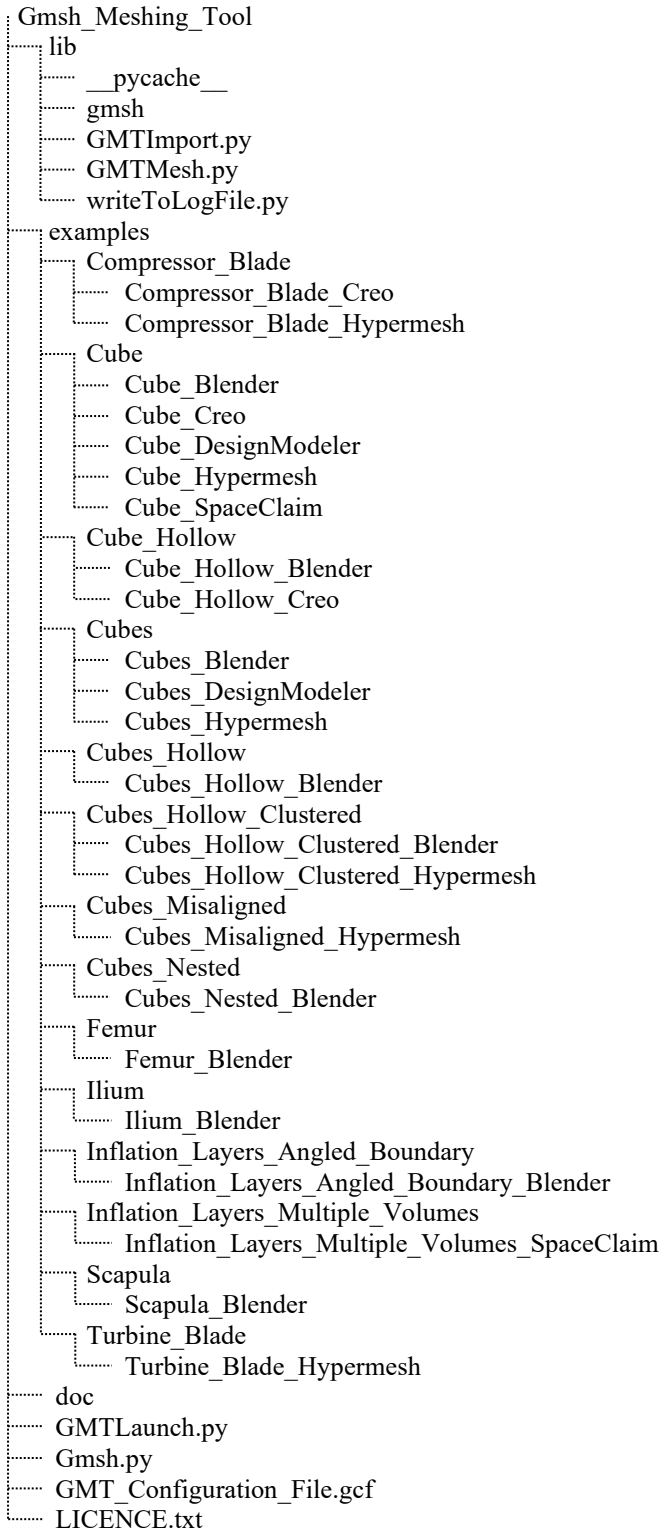

Fig. 4 Structure of the developed library.

3.1 GCF Configuration File

The developed software tool requires two different inputs for its use. The first one is the GCF configuration file. This is an ASCII file with UTF-8 encoding. It can therefore be opened and modified as a regular text file in any of a plethora of word processing applications. Each line of the

file contains one parameter. The parameter name is always followed by the equals sign "=", followed by the parameter value itself, and finally by an optional informative label after the "#" character. Tables below contain a complete list of all parameters and their descriptions. These are divided into general parameters in Tab. 1, STL topology parameters in Tab. 2, global meshing parameters in Tab. 3, local meshing parameters in Tab. 4 and parameters for generation of inflation layers in Tab. 5.

The parameter values can be one of the following data types:

- The “Integer” data type represents whole numbers,
- the “Float” data type represents decimal numbers,
- the “Bool” data type represents logical values, i.e. “True”, “False”, “Yes”, “No”, “1”, or “0”.
- The “String” data type represents strings of Unicode characters (words) and finally
- the “List” data type represents a list of values of the “String” data type, separated by a comma or semicolon.

Tab. 1 List of the general parameters of the GCF configuration file.

Parameter Name	Default Value	Data Type	Description
`WorkingDirectoryPath`	—	String	Absolute path to the working directory.
`Name`	—	String	Job name.
`InputFormat`	`1`	Integer	Input geometry file format (`1`: IGES, `2`: STL).
`GeometryTolerance`	$1 \cdot 10^{-9} \cdot d$	Float	Geometric tolerance, `d` represents the diagonal of the bounding box of the model geometry.
`MaxNumThreads`	`1`	Integer	Maximum number of CPU threads to use for meshing.
`OutputFormat`	`1`	Integer	Format of the output model mesh files (`1`: MSH, `2`: UNV, `3`: VTK).
`Binary`	`False`	Bool	Save the mesh to a binary file if possible?
`LaunchGmsh`	`False`	Bool	Run Gmsh to see the result after finishing the task?

Tab. 2 List of the STL topology parameters of the GCF configuration file.

Parameter Name	Default Value	Data Type	Description
`STLRemesh`	`False`	Bool	Parameterize and remesh the STL model geometry?
`STLFacetAngle`	`45.`	Float	[°] Minimum angle between the normals of two adjacent triangles at which their common segment is already considered an edge.
`STLCurveAngle`	`180.`	Float	[°] Minimum angle between two segments at which their common point is already considered a corner.

Tab. 3 List of the global meshing parameters of the GCF configuration file.

Parameter Name	Default Value	Data Type	Description
MeshDim	3	Integer	Number of dimensions of the resulting finite element mesh (0: 0D, 1: 1D, 2: 2D, 3: 3D).
MeshAlgorithm	6	Integer	2D meshing algorithm (1: MeshAdapt, 2: Automatic, 3: Initial mesh only, 5: Delaunay, 6: Frontal-Delaunay, 7: BAMG, 8: Frontal-Delaunay for Quads, 9: Packing of Parallelograms).
MeshAlgorithm3D	10	Integer	3D meshing algorithm (1: Delaunay, 3: Initial mesh only, 4: Frontal, 7: MMG3D, 9: R-tree, 10: HXT).
MeshSizeFromCurvature	6	Integer	Target number of segments per 2π radians of curvature.
MeshSizeExtendFromBoundary	True	Bool	Extend computation of mesh element sizes from the boundaries into the interior?
MeshSizeMax	$2 \cdot 10^{-2} \cdot d$	Float	Maximum global size of the finite elements.
MeshSizeMin	0	Float	Minimum global size of the finite elements.
MinimumCircleNodes	0	Integer	Minimum number of nodes per circle or ellipse.
MinimumCurveNodes	0	Integer	Minimum number of nodes per curve other than a line, circle, or ellipse.
ElementOrder	1	Integer	Order of the finite elements (1: linear, 2: quadratic, 3: cubic).
SecondOrderIncomplete	True	Bool	Create incomplete 2nd order elements (8-node quads, 20-node hexas, etc.,)?
SecondOrderLinear	False	Bool	Create mid-side nodes of higher-order elements and nodes resulting from "subdivision" algorithms by simple linear interpolation?
Optimize	True	Bool	Optimize the mesh to improve the quality of tetrahedral finite elements?
OptimizeNetgen	False	Bool	Optimize the mesh to improve the quality of tetrahedral finite elements using the Netgen library?
HighOrderOptimize	4	Integer	Algorithm for optimization of higher order element mesh (0: none, 1: optimization, 2: elastic and optimization, 3: elastic, 4: fast curving).

Tab. 4 List of local meshing parameters of the GCF configuration file.

Parameter Name	Default Value	Data Type	Description
`LocalMeshSurfaces`		String	List of surfaces on which to enforce the local element size.
`LocalMeshVolumes`		String	List of volumes on which to enforce the local element size.
`LocalMeshSize`		Float	Target local size of the finite elements on the selected surfaces/volumes.
`LocalMeshGrowthRate`		Float	Rate of change of size of the neighboring elements near the selected surfaces/volumes.

The foursome of local meshing parameters, see Tab. 4, can in the configuration file be declared repeatedly, i.e. separately for each group of geometric entities (surfaces and/or volumes).

Tab. 5 List of parameters of the GCF configuration file for the generation of the inflation layers.

Parameter Name	Default Value	Data Type	Description
`InflationLayersSurfaces`		String	List of surfaces on which to generate inflation layers.
`InflationLayers`		Integer	Number of inflation layers to generate on the selected surfaces.
`InflationLayersMethod`		Integer	Method of specification of inflation layer thickness (1: Total thickness, 2: First layer thickness, 3: Total aspect ratio, 4: First layer aspect ratio).
`InflationLayersThickness`		Float	Inflation layer thickness parameter according to specified method.
`InflationLayersGrowthRate`		Float	Rate of change of thickness of two neighboring inflation layers belonging to the selected surface.

All surfaces on which the inflation layers are to be created must currently have identical values of parameters, as this functionality is still under development. These are in particular the `InflationLayers`, `InflationLayersThickness`, `InflationLayersMethod` and `InflationLayersGrowthRate` parameters, see Table 5.

Length units of parameters `GeometryTolerance`, `MeshSizeMax`, `MeshSizeMin`, `LocalMeshSize` and `InflationLayersThickness` (when `InflationLayersMethod` = `1` or `2`) depends on geometry format. In case of IGS format it is millimetres. In case of STL format length units equal units in which the model geometry was exported from the geometry pre-processor.

3.2 Requirements for the Geometry Files

The second required input are the model geometry files. Files in IGES (with extension .igs or .iges) or STL (with extension .stl) format are accepted. However, depending on the format used, certain restrictions apply that impose requirements on the arrangement of the geometry files and their names.

Each job of the tool consists of a one or more bodies, volumetric or planar. All bodies that are adjacent to each other form a so-called island, see Fig. 5 a). Each island then consists of one or more bodies and those islands are not connected to each other. If the model consists of several disconnected islands, see Fig. 5 b), it is then necessary to submit a job for each island separately. However, each a job is submitted, a separate process is assigned to it and the solution of the entire problem is thus parallelized at the level of said islands. Each body of each island must then be assigned a separate geometry file or files in the same format (IGES or STL).

a) Model geometry consisting of a single island made of two adjacent volumes.

b) Model geometry consisting of two islands, each made of single volume

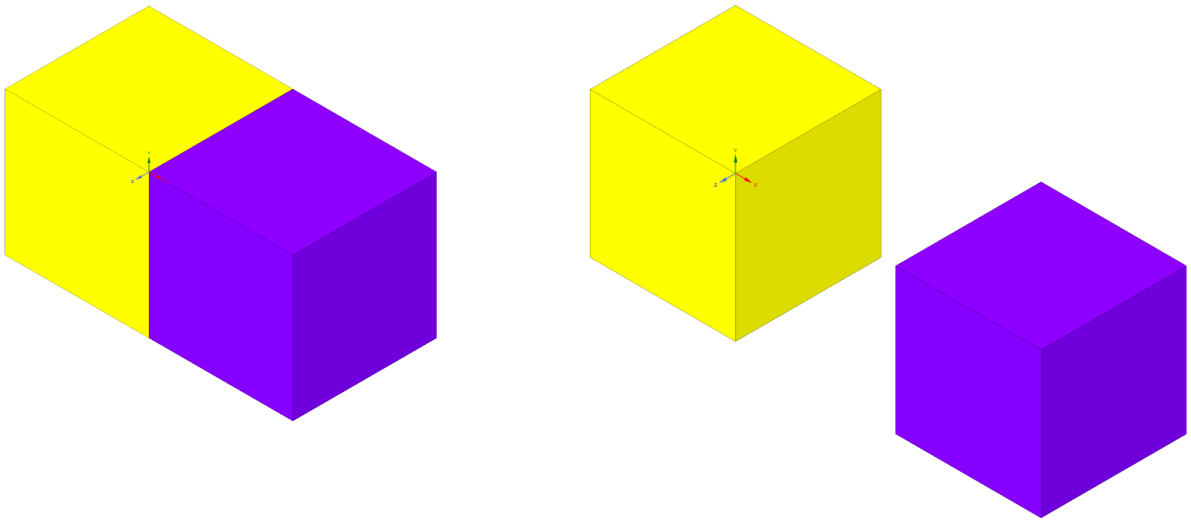

Fig. 5 Model geometry consisting of a single island versus multiple islands.

3.2.1 Geometry in the IGES Format

In the case of the IGES format, the complete name of each file consists of the model name common to all model files, the name of the body to which the particular file belongs, and the file extension, always separated by a period, see option a) in Fig. 6 below. A shortened name, consisting only of the model name and the suffix, is also acceptable, see option b). All bodies in the model will then be assigned the model name, followed by a sequence number, as their name.

It is also possible to split the geometry into IGS files already at the level of shells, see option c) in Fig. 6 below. Each shell consists of a number of faces that are adjacent to each other. In this case, the names of the files corresponding to the shells separating two bodies must then specifically follow the format shown in variant d) in Fig. 6. The file name in this case contains the names of the two bodies separated by the shell. In any case, however, it must be ensured that the faces in each file are adjacent to each other (analogous to islands of bodies, no disconnected faces allowed). For models in IGES format, all length dimensions are always expressed in millimetres.

a) `<model name>.<body name>.igs`

b) `<model name>.igs`

c) `<model name>.<body name>.<shell name>.igs`

d) `<model name>.<body 1 name>.<body 2 name >.<shell name>.igs`

Fig. 6 Possible variants of the names of the IGS geometry files, brackets <> are not written.

3.2.2 Geometry in the STL Format

Analogous naming conventions for geometry files apply to the STL format. When the model consists of only one body, variants a) and b) in Fig. 7 below can be used. However, beware that if the STL geometry files contain user-named surfaces, then it is necessary that these files are in ASCII form. This is because the Gmsh library does not yet support reading surface names from binary files.

Otherwise, when the model is composed of multiple bodies, it is necessary to split the geometry into STL files already at the level of shells, see option c) in Fig. 7 below. Each shell consists of a number of faces that are adjacent to each other. The names of the files corresponding to the shells separating two bodies must then specifically follow the format shown in variant d) in Fig. 7. The file name in this case contains the names of the two bodies separated by the shell. In this case, the geometry files may be in both ASCII or binary form. In the case of models in STL format, all length dimensions are always expressed in the units in which the model was exported from the geometry pre-processor.

a) `<model name>.<volume name>.stl`

b) `<model name>.stl`

c) `<model name>.<volume name>.<shell name>.stl`

d) `<model name>.<volume 1 name>.< volume 2 name >.<shell name>.stl`

Fig. 7 Possible variants of the names of the STL geometry files, brackets <> are not written.

3.3 Installation and Usage of the Library

To install the tool, one must first have Python installed. Compatibility is guaranteed for Python versions 3.10.x, for other versions, full functionality is not guaranteed.

After that, it is required to install the NumPy library. In Windows operating system, this can be done easily with the `pip install numpy` command in the command line (run as administrator).

In the case of the Linux operating system, use the `pip3 install numpy` command in the command line. Next step is to unzip the compressed directory containing the tool into the installation directory of one's choice.

All that remains is to set up an environment variable pointing to the installation directory of the tool. In Windows, the standard way to do this is to right click on "This Computer", then select "Properties" from the popup menu, next click on "Advanced System Settings", then on "Advanced" tab, click on "Environment Variables", under "System Variables" click on "New...", then in the "Variable Name:" field, type `GMTPATH` (uppercase) and in the "Variable Value:" field, input the absolute path to the installation directory of the tool.

In Linux, first use the `nano ~/.bashrc` command in the command line to open (or create and edit if none exists) the .bashrc file. Next add the following line `export GMTPATH=<"absolute path to the installation directory of the tool">`, note that brackets <> are not written and the quotation marks should be used if the path contains spaces.

The tool can be used in two ways, according to the first way one must first create a working directory and place the model geometry files in the appropriate format in it. Next, set the required parameter values in the GCF configuration file located in a root directory of the library. Now all that is left to do is to run the file "GMTLaunch.py" which is also located in the root directory.

The second way is to copy the filled GCF file and the "GMTLaunch.py" file to the working directory with the model geometry and rename those two files there with the model name. In this case, it is not necessary to set the `WorkingDirectoryPath` and `Name` parameters in the GCF file, as these are filled in automatically when the launch file is executed from the working directory.

When the job is complete, the tool writes the resulting finite element mesh to a file in the chosen format and saves it in the job's working directory along with the LOG file. If a fatal problem occurred while solving the job, then the tool saves only the LOG file with an explanation of the cause of the problem.

4 Demonstration Examples

This chapter provides examples demonstrating the most important features of the developed library. The examples show the discretization of geometry by tetrahedral finite elements, both parametric geometries in IGES input format and non-parametric geometries in STL format.

In the geometry pre-processor (CAD software package), the selected faces of the model geometry were assigned arbitrary names and the model was then exported to an exchange format (IGES or STL). Not every CAE software package, whether open source or commercial, allows the user to assign the names of choice to the geometry faces and to further write this information to the geometry file in the exchange format. In the case of the IGES exchange format, the following commercial software packages were used to prepare the geometry of the demonstration examples in the manner described above:

- PTC Creo CAD software,
- FEM pre-processor Altair Hypermesh.

In the case of the STL format, the following were used:

- Ansys SpaceClaim geometry pre-processor,
- 3D graphics software Blender.

The former is commercial software, and the latter is open source. Both options then allow the STL file to be written in both ASCII and binary form and allow the model to be saved either in a single file or each continuous shell to be saved in a separate STL file see Fig. 8.

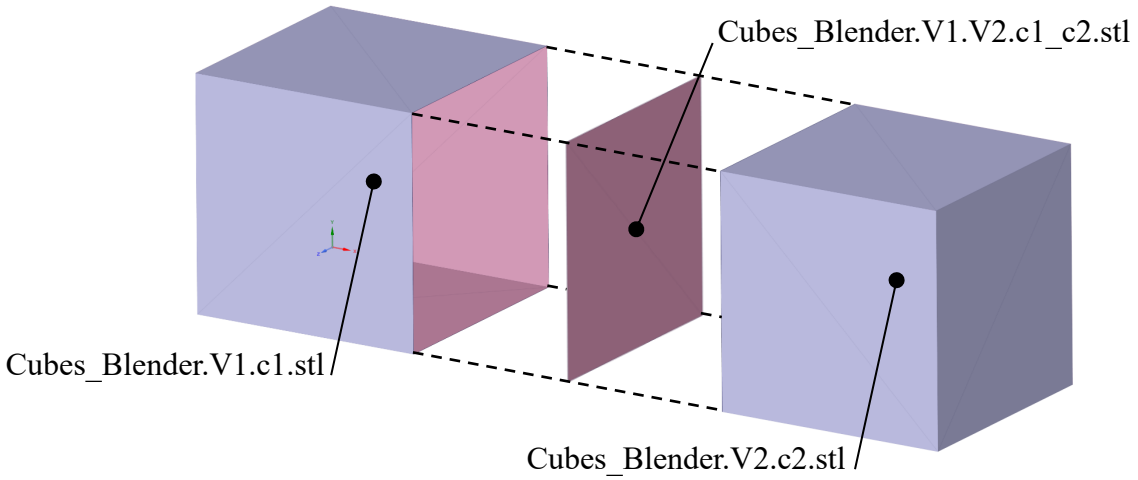

Fig. 8 Division of the model geometry into separate STL files.

In the first example, the usability of the above-mentioned software packages to prepare IGES or STL geometry with user-named faces will be demonstrated on four variants of a simple cube. The

second example uses two different cases to demonstrate partitioning of the geometry into adjacent volumes with a conjoint finite element mesh, which is needed whenever multiple different materials need to be used in the model. The examples that follow demonstrate library functions on more complicated geometry. The example of the femur is used to demonstrate the splitting of the general STL geometry into separate named sections and remeshing of the original model topology. The example of a jet engine turbine blade demonstrates the discretization of a complex IGES geometry consisting of multiple adjacent volumes. The next example demonstrates the discretization of a complicated geometry, including the mesh refining of the original STL topology, but now on the example of a pelvic bone, consisting of a total of three adjacent volumes. The final example demonstrates generation of inflation layers in domain consisting of multiple volumes.

Both output (mesh file in selected format and LOG files) and input (GCF file and geometry files in IGES or STL format) files of all demonstration examples can be found in the "examples" directory.

4.1 Single Volume

The first demonstration example is a cube discretized by linear tetrahedral finite elements. The example contains two cases, each with two variants. The total of 4 variants demonstrates capability of the library to accept geometry from 4 different CAE software environments. In the first case, the geometry of a cube with an edge length of 1 m was imported into the tool in IGES format. In the first variant, the geometry was prepared (and the names assigned to the selected faces) in the PTC Creo commercial CAD software environment, see Fig. 9 a). In the second variant, the geometry was prepared in the environment of the commercial FEA pre-processor Altair Hypermesh, see Fig. 9 b).

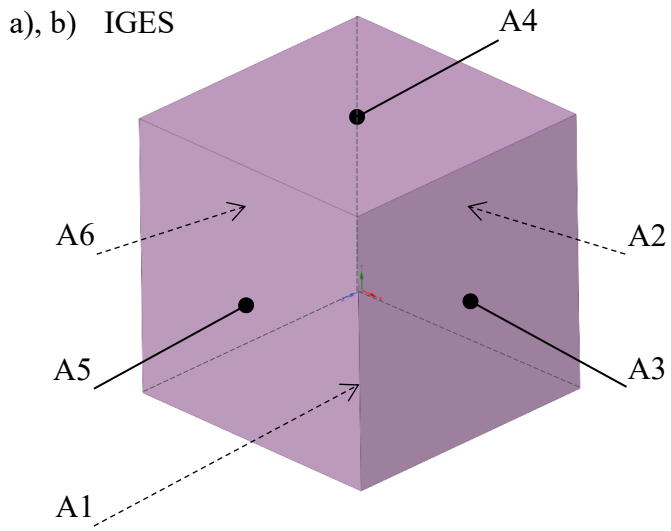

Fig. 9 Geometry of the cube model with named faces for the variants in the IGES format.

In the second case, the geometry of the cube model with an edge length of 2 m was imported in STL format. In the first variant, the model geometry was prepared in the Ansys SpaceClaim software environment as a single ASCII file with named groups (shells), see Fig. 10 c). In the second variant, the geometry was prepared in the Blender software environment and stored in the form of three continuous shells in three separate files, see Fig. 10 d):

- "Cube_Blender.V1.A1.stl",
- "Cube_Blender.V1.A2.stl" and
- "Cube_Blender.V1.A3.stl".

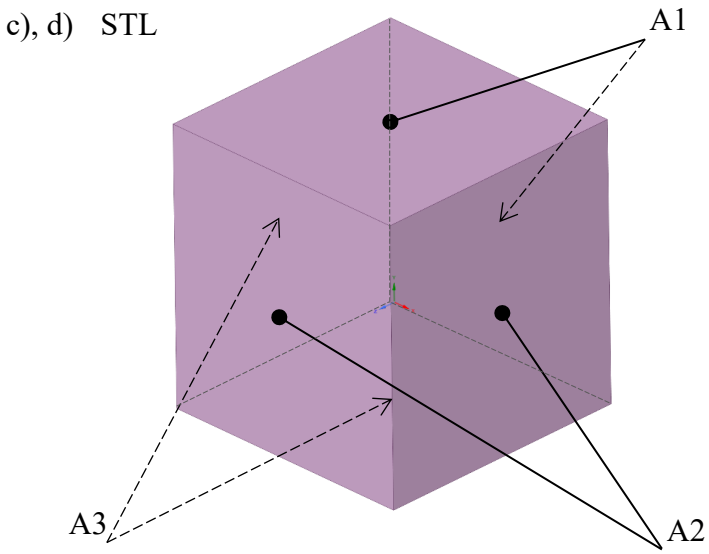

Fig. 10 Geometry of the cube model with named faces for variants in the STL format.

The input parameter values written to the GCF file in each variant are listed in Tab. 6 below. The values of the other parameters were considered as default.

Tab. 6 User-entered parameter values in the GCF file for different variants of the lone cube example.

Parameter Name	Value, Var. a)	Value, Var. b)	Value, Var. c)	Value, Var. d)
InputFormat	1	1	2	2
STLRemesh	False	False	False	True
MeshSizeMax	200. [mm]	200. [mm]	0.3 [m]	0.3 [m]
LaunchGmsh	True	True	True	True

The resulting finite element mesh was always written to the mesh file in the selected format and saved to the job's working directory along with the LOG file. The resulting finite element discretization for all four variants are shown in Fig. 11 below. The individual named surfaces are distinguished by different colours. The resulting finite element mesh of variants a) and b) came out virtually identical. Between variants c) and d), the resulting finite element mesh differs substantially,

since in variant c), unlike in variant d), there was no remeshing of the original STL cube topology used (`STLRemesh = False`).

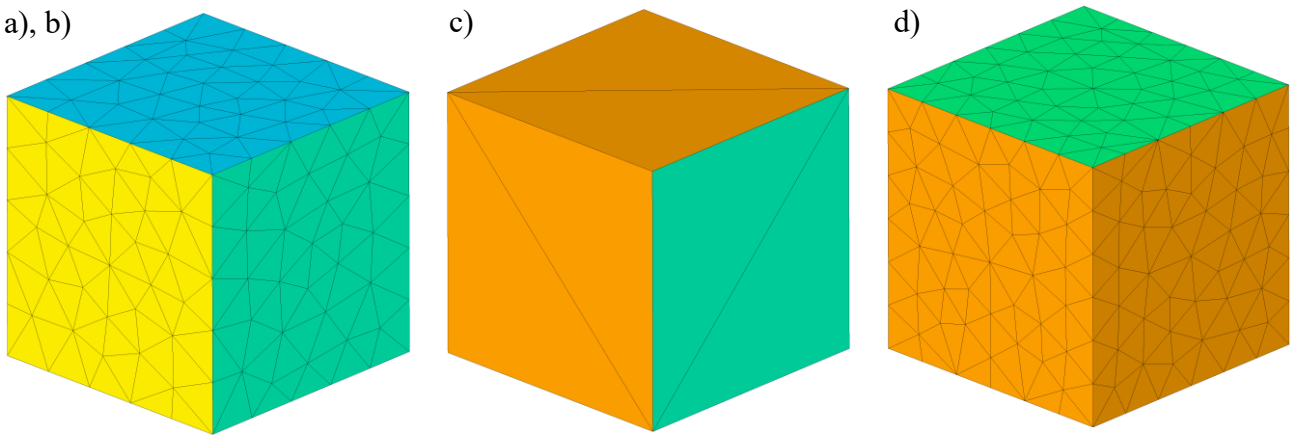

Fig. 11 The resulting finite element mesh for each variant of the cube model.

In Fig. 12 below, a preview of the resulting mesh file in .msh format (MSH) can be seen for all four variants of the example with the colour-highlighted names of the finite element groups corresponding to the user-named surfaces (red) and volumes (blue). As can be seen, the originally selected surface names have been changed in variant b). It is because the Hypermesh software automatically increments the surface names entered by the user. The surface names have been also modified in variant c). This time the SpaceClaim software is to blame, as it also automatically writes information about the length unit used into the group names inside the STL files. The STL format alone does not contain this information. A detailed description of the MSH file structure can be found in Appendix A.

a) PTC Creo

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
7
2 1 "A1"
2 2 "A2"
2 3 "A3"
2 4 "A4"
2 5 "A5"
2 6 "A6"
3 7 "V1"
$EndPhysicalNames
```

b) Altair Hypermesh

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
7
2 1 "A1.1"
2 2 "A2.1"
2 3 "A3.1"
2 4 "A4.1"
2 5 "A5.1"
2 6 "A6.1"
3 7 "V1"
$EndPhysicalNames
```

c) ANSYS SpaceClaim

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
4
2 1 ""A1<stl unit=MM>""
2 2 ""A2<stl unit=MM>""
2 3 ""A3<stl unit=MM>""
3 4 "V1"
$EndPhysicalNames
```

d) Blender

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
4
2 1 "A1"
2 2 "A2"
2 3 "A3"
3 4 "V1"
$EndPhysicalNames
```

Fig. 12 Previews of the MSH files with user-named surfaces and volumes for each variant of the lone cube the example.

4.2 Two Adjacent Volumes with One Common Face

The geometry of the second demonstration example has been divided into two cubes, as each of these cubes is made of a different material. However, the cubes still share one of their six faces. The discretization was done with linear tetrahedral elements. The example again contains two cases to demonstrate differences in workflow between different formats of geometry when working with multiple volumes.

In the first case, the geometry was prepared in Altair Hypermesh software and imported in the IGES format into the tool environment. The geometry of the model consists of two separate cubes, each with an edge length of 1 m, and thus each one belongs to a separate file. The faces of each cube were again assigned arbitrary names, see Fig. 13.

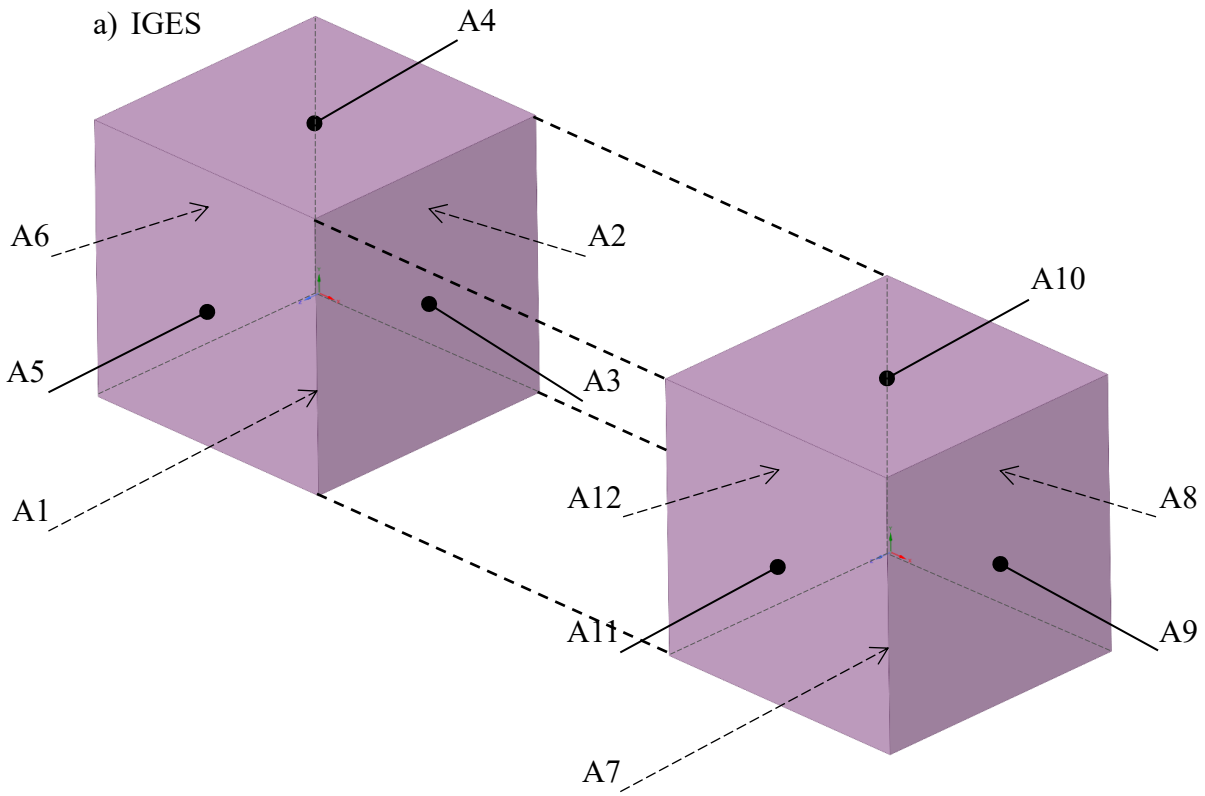

Fig. 13 Geometry of the model of two cubes with the user-provided surface names, IGES format.

In the second case, the geometry was prepared in Blender software and transferred to the tool environment in STL format. In accordance with the requirements for STL files in the case of multiple adjacent volumes, as described in Section 3.2, the model geometry consists of a total of three shells, each with an edge length of 2 m, where one of these shells represents a bulkhead separating the volumes of the individual cubes and each one of these shells belongs to a separate file, see Fig. 14.

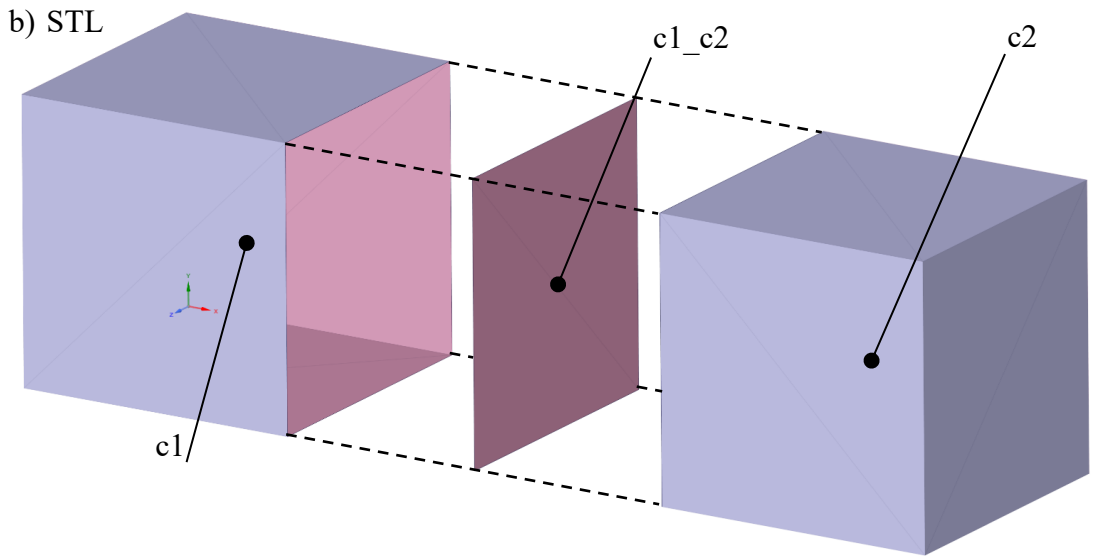

Fig. 14 Geometry of the model of two cubes with the user-provided surface names, STL format.

The input parameter values used in each variant that were written to the GCF file are listed in Tab. 7 bellow. The values of other parameters were considered as default.

Tab. 7 User-entered parameter values in the GCF file for the two cubes example.

Parameter Name	Value, case a)	Value, Case b)
InputFormat	1	2
STLRemesh	False	True
MeshSizeMax	0.2 [mm]	0.5 [m]
LaunchGmsh	True	True
LocalMeshSurfaces	V1.p6.1	
LocalMeshVolumes		V2
LocalMeshSize	0.04	0.1
LocalMeshGrowthRate	1.5	2.0

The resulting finite element mesh was written to the MSH file and saved to the job's working directory along with the LOG file. The resulting finite element discretization for both cases are depicted in Fig. 15 below. The individual named shells are distinguished by colour. Note that the finite element mesh was refined in one of the surfaces in variant a) and one of the volumes in variant b), since respective local meshing parameters were provided.

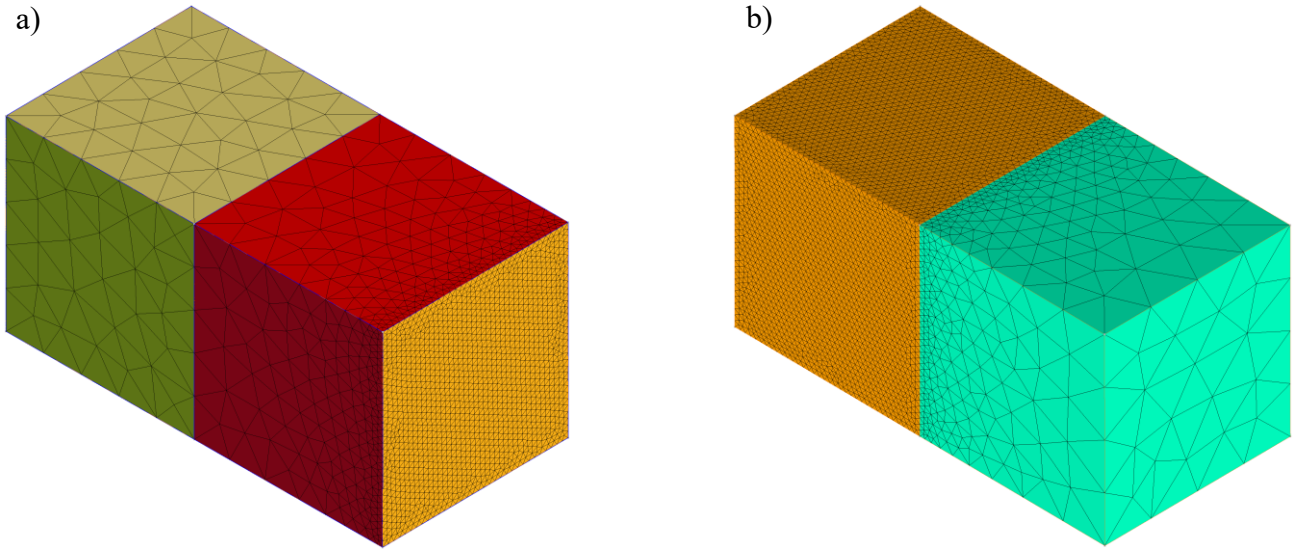

Fig. 15 The resulting finite element mesh for each case of the two cubes example.

In Fig. 16 below, a preview of the resulting MSH file for both cases can be seen, with the colour-highlighted names of the finite element groups, corresponding to the user-named surfaces (red) and volumes (blue). As can be seen, the originally chosen surface names have been changed in option b). It is because the Hypermesh software automatically increments the surface names and adds the name of the parent volume. A detailed description of the MSH file structure can be found in Appendix A.

a) IGES

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
13
2 1 "V1.A1.1"
2 2 "V1.A2.1"
2 3 "V2.16.1"
2 4 "V1.A4.1"
2 5 "V1.A5.1"
2 6 "V1.A6.1"
2 7 "V2.A7.1"
2 8 "V2.A8.1"
2 9 "V2.A9.1"
2 10 "V2.A10.1"
2 11 "V2.A11.1"
3 12 "V1"
3 13 "V2"
$EndPhysicalNames
```

b) STL

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
5
2 1 "A1"
2 2 "A1_A2"
2 3 "A2"
3 4 "V1"
3 5 "V2"
$EndPhysicalNames
```

Fig. 16 Previews of the MSH files with user-named surfaces and volumes for both cases of the two cubes the example.

4.3 Femur Bone

The third demonstration example is a human thigh bone, whose geometric model is a product of a CT scan and was obtained by the author as a freely available model from the GrabCAD 3D model

library[11]. The model geometry was further modified in the Blender software environment, where it was divided into four separate sections corresponding to the basic anatomical parts of the femur. The geometry was therefore prepared as four separate STL files, see Fig. 17, namely:

- "Femur_Blender.V1.Caput_Femoris.stl",
- "Femur_Blender.V1.Collum_Femoris.stl",
- "Femur_Blender.V1.Condyli_Femoris.stl" and
- "Femur_Blender.V1.Corpus_Femoris.stl".

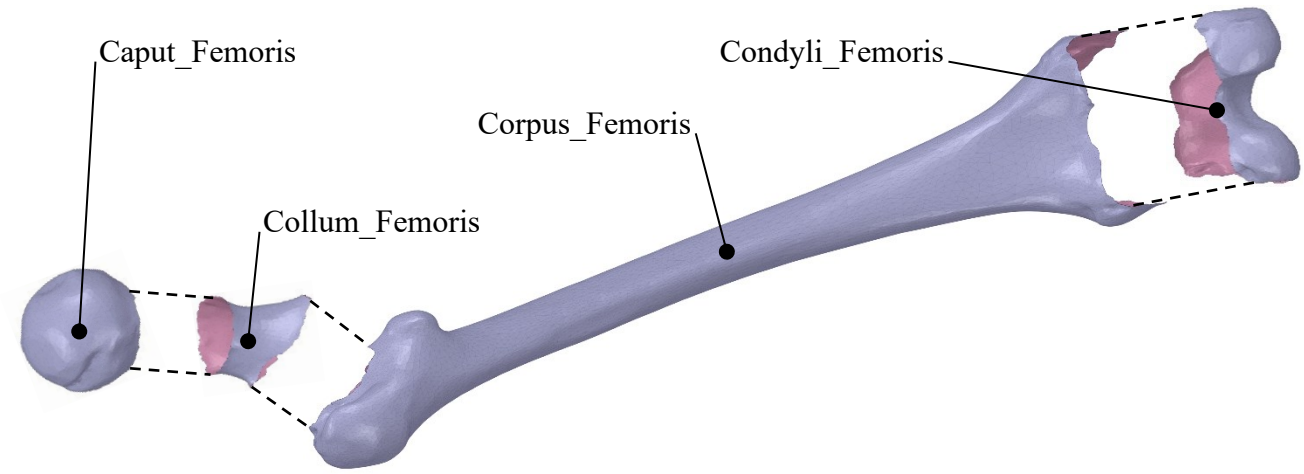

Fig. 17 Femur model geometry with the user-provided shell names.

Subsequently, the path to the working directory and the user-provided values of the configuration parameters were written into the GCF file, see Tab. 8. The values of other parameters were considered as default.

Tab. 8 User-entered parameter values in the GCF file for the Femur example.

Parameter Name	Value
InputFormat	2
LaunchGmsh	True
STLRemesh	True
STLCurveAngle	150[°]
MeshAlgorithm	5
MeshSizeFromCurvature	12
MeshSizeMax	5.E-3 [m]
MeshSizeMin	2.E-3 [m]
Optimize	False
OptimizeNetgen	True

The resulting finite element mesh was written to the MSH file and saved to the job's working directory along with the LOG file. The resulting finite element discretization is depicted in Fig. 18 below. The individual shells are distinguished by colour.

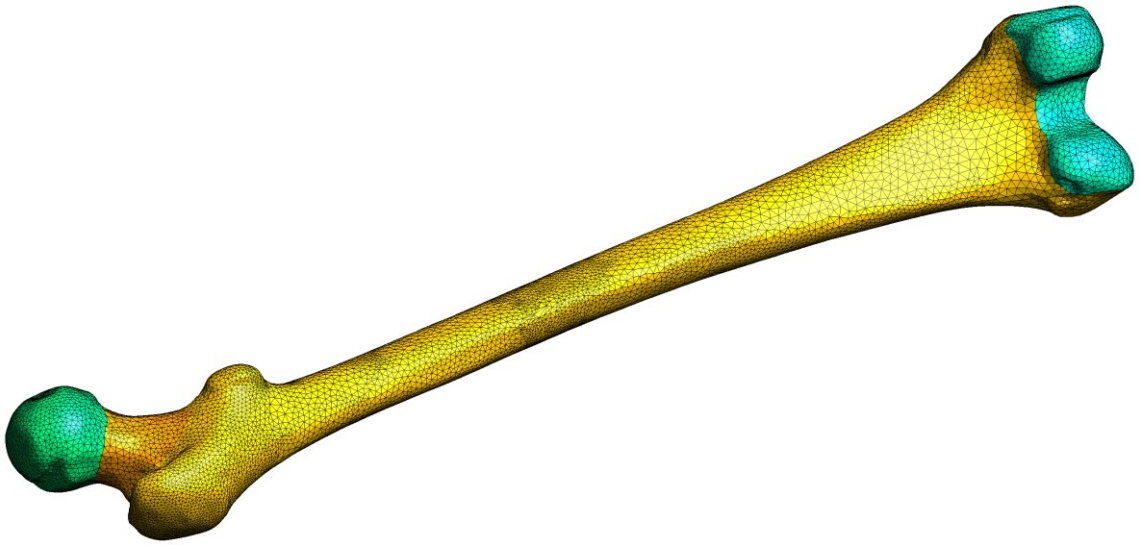

Fig. 18 The resulting finite element mesh of the femur model.

In Fig. 19 below, a preview of the resulting MSH file can be seen with the color-highlighted names of the finite element groups corresponding to the user-named surfaces (red) and volume (blue). A detailed description of the MSH file structure can be found in Appendix A.

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
5
2 1 "Caput_Femoris"
2 2 "Collum_Femoris"
2 3 "Condyli_Femoris"
2 4 "Corpus_Femoris"
3 5 "V1"
$EndPhysicalNames
```

Fig. 19 A preview of the MSH file with user-named surfaces and volume for the femur bone example.

4.4 Turbine Blade of a Jet Engine

The fourth demonstration example is a turbine blade of a jet engine, the geometric model of which is a part of a jet engine assembly, the model of which was donated to the author by Dr. Renat Badykov from the Department of Aircraft Engine Construction and Design at the National Research University of Samara in Russia. The model was pre-processed in the Altair Hypermesh, where it was divided

into separate volumes corresponding to the basic parts of the turbine blade and then exported in IGES format in the form of four files corresponding to the four volumes, see Fig. 20.

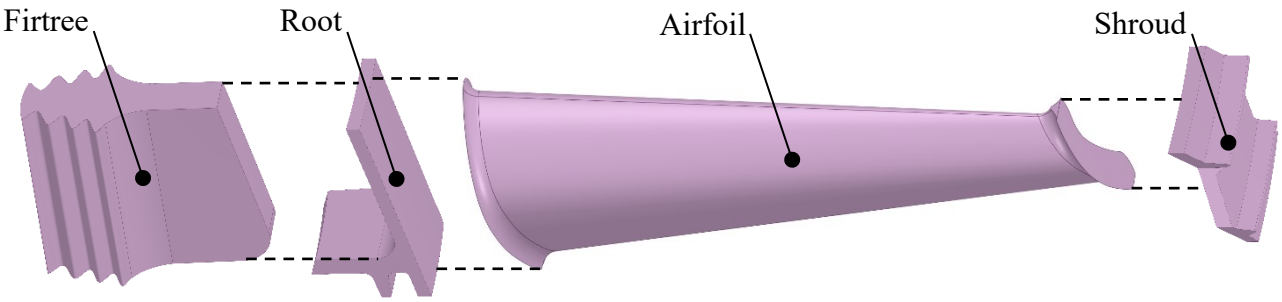

Fig. 20 Geometry of the turbine blade model with the user-provided volume names.

As usual, the absolute path to the working directory and the selected values of the configuration parameters were then written to the GCF file, see Tab. 9. The values of other parameters were considered as default.

Tab. 9 User-entered parameter values in the GCF file for the turbine blade example.

Parameter Name	Value
LaunchGmsh	True
MeshSizeFromCurvature	12
MeshSizeMax	1. [mm]
MeshSizeMin	0.1 [mm]
MinimumCirclePoints	12
MinimumCurvePoints	2
ElementOrder	2

The resulting finite element mesh was written to the MSH file and saved to the job's working directory along with the LOG file. The resulting finite element discretization is depicted in Fig. 21 below. The individual volumes are distinguished by colour.

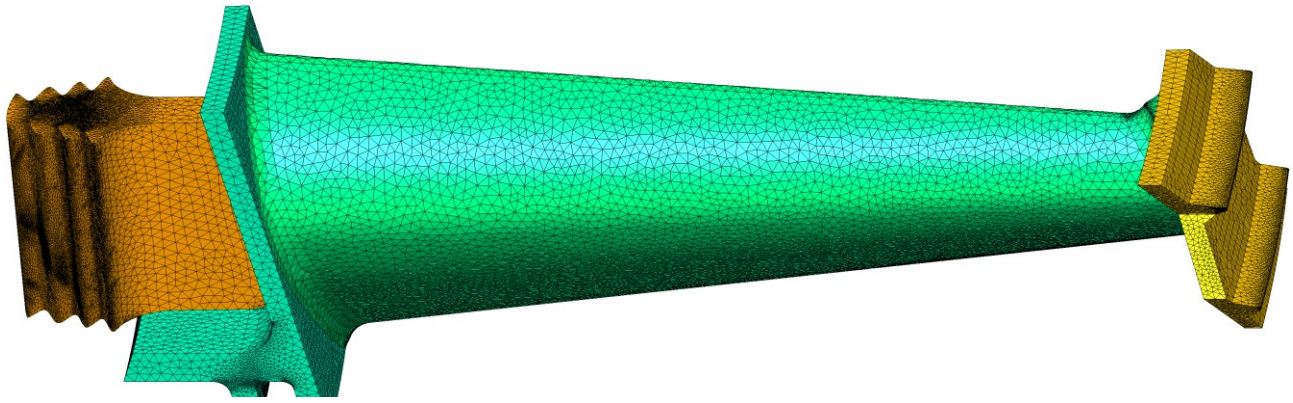

Fig. 21 The resulting finite element mesh of the turbine blade model.

A preview of the resulting MSH file can then be seen in Fig. 22 below, with the names of the finite element groups corresponding to the user-named volumes highlighted in red. A detailed description of the MSH file structure can be found in Appendix A.

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
4
3 1 "Airfoil"
3 2 "Firtree"
3 3 "Root"
3 4 "Shroud"
$EndPhysicalNames
```

Fig. 22 Preview of the MSH file with the user-named volumes for the turbine blade example.

4.5 Ilium Bone

The last demonstration example is a human pelvic bone whose geometric model is a product of a CT scan and was obtained by the author as a freely available model from the GrabCAD 3D model library. The model geometry was further modified in the Blender software environment, where it was divided into three separate volumes corresponding to the three separate bones that actually make up the pelvic bone. For these volumes, the author further modelled surfaces serving as bulkheads that separate the resulting volumes. The resulting geometry was then imported into the tool environment in the form of seven separate STL files, see Fig. 23, namely:

- "Ilium_Blender.Ilium.A1.stl",
- "Ilium_Blender.Ilium.Ischium.A1_A3.stl",
- "Ilium_Blender.Ilium.Pubis.A1_A2.stl",
- "Ilium_Blender.Ischium.A3.stl",
- "Ilium_Blender.Pubis.A2.stl",
- "Ilium_Blender.Pubis.Ischium.A2_A3_1.stl" and
- "Ilium_Blender.Pubis.Ischium.A2_A3_2.stl".

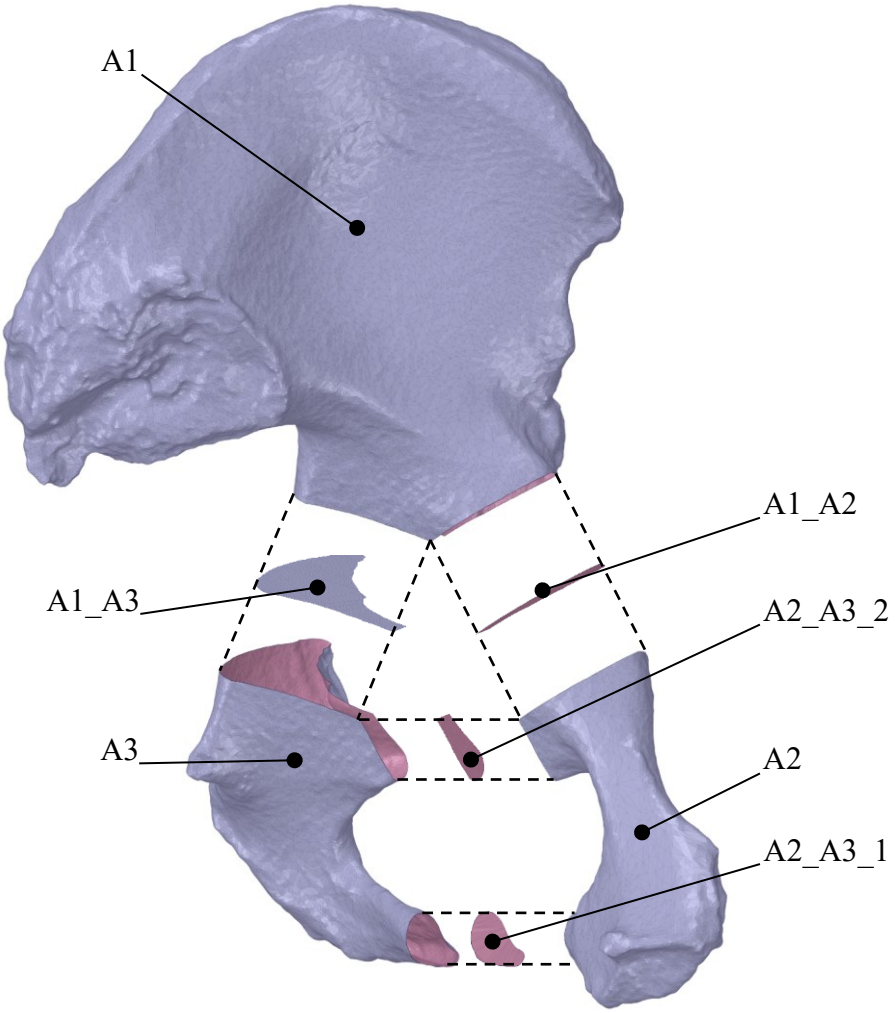

Fig. 23 Geometry of the ilium bone model with the user-provided shell names.

Subsequently, the absolute path to the working directory and user-selected values of the configuration parameters were written into the GCF file, see Tab. 10. The values of other parameters were considered as default.

Tab. 10 User-entered parameter values in the GCF file for the ilium bone example.

Parameter Name	Value
`InputFormat`	`2`
`MaxNumThreads`	`3`
`LaunchGmsh`	`True`
`STLRemesh`	`True`
`STLFacetAngle`	`60. [°]`
`MeshSizeMax`	`2.E-3 [m]`
`MeshSizeMin`	`4.E-4 [m]`

The resulting finite element mesh was written to the MSH file and saved to the job's working directory along with the LOG file. The resulting finite element discretization is depicted in Fig. 24 below. The individual user-named shells are distinguished by colour.

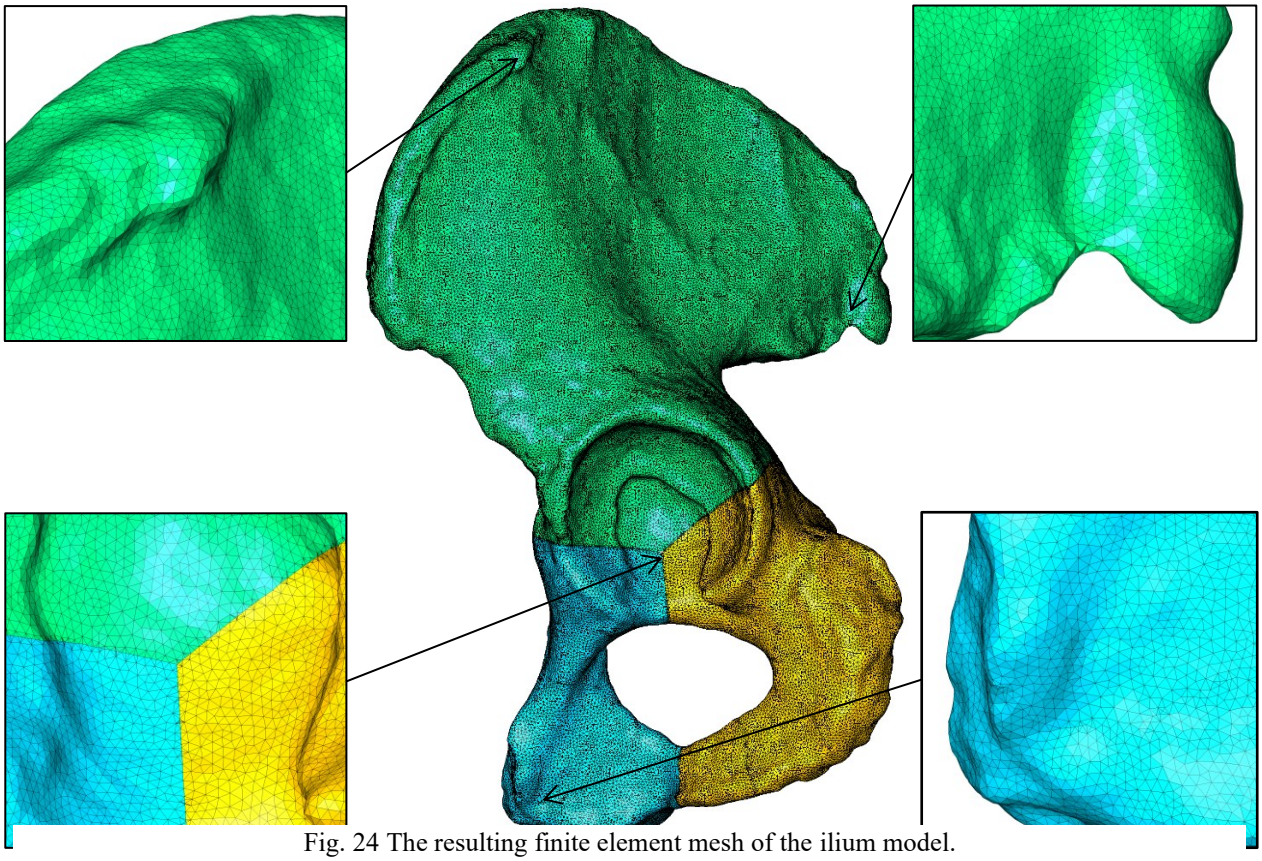

Fig. 24 The resulting finite element mesh of the ilium model.

In Fig. 25 below can be seen a preview of the resulting MSH file with the colour-highlighted names of the finite element groups corresponding to the user-named surfaces (blue) and volumes (red). A detailed description of the MSH file structure can be found in Appendix A.

```
$MeshFormat
4.1 0 8
$EndMeshFormat
$PhysicalNames
10
2 1 "A1"
2 2 "A1_A3"
2 3 "A1_A2"
2 4 "A3"
2 5 "A2"
2 6 "A2_A3_1"
2 7 "A2_A3_2"
3 8 "Ilium"
3 9 "Ischium"
3 10 "Pubis"
$EndPhysicalNames
```

Fig. 25 Preview of the MSH file with the user-named surfaces and volumes for the pelvic bone example.

4.6 Inflation Layers in Domain Made of Multiple Volumes

The last example demonstrates the generation of inflation layers on geometry composed of two fitting volumes, see Fig. 26. The surfaces on which the inflation layers are to be generated are chosen so that the boundary between the two volumes passes through the intended inflation layers. The model geometry was imported into the environment of the tool as two separate IGES files.

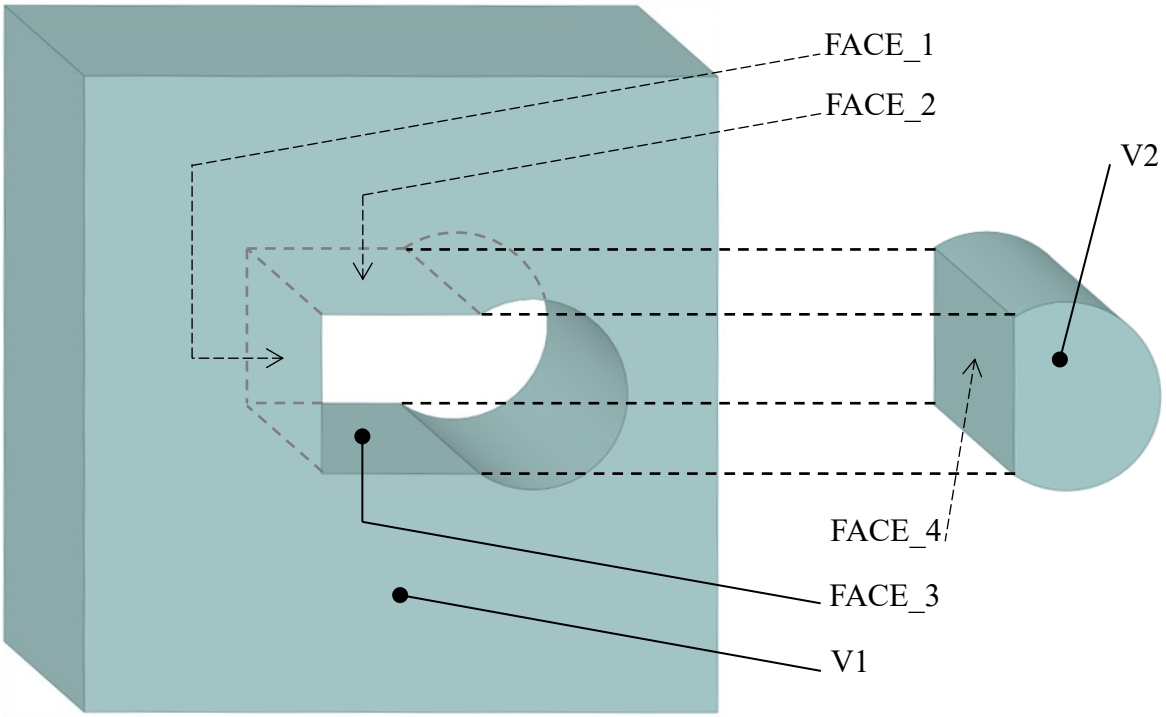

Fig. 26 Geometry of the model for generation of inflation layers with names of selected faces and volumes.

Subsequently, the absolute path to the working directory and the user-selected values of the configuration parameters were written into the GCF file, see Tab. 11. The values of the other parameters were considered as default.

Tab. 11 User-entered parameter values in the GCF file for the ilium bone example.

Parameter Name	Value
LaunchGmsh	True
MeshSizeMax	50. [mm]
MinimumCircleNodes	20
InflationLayersSurfaces	FACE_1,FACE_2,FACE_3,FACE_4
InflationLayers	10
InflationLayersMethod	1
InflationLayersThickness	35.
InflationLayersGrowthRate	1.25

The resulting finite element mesh was written to the MSH file and saved to the job's working directory along with the LOG file. The resulting finite element discretization is depicted in the Fig. 27 below. The individual named shells are distinguished by color.

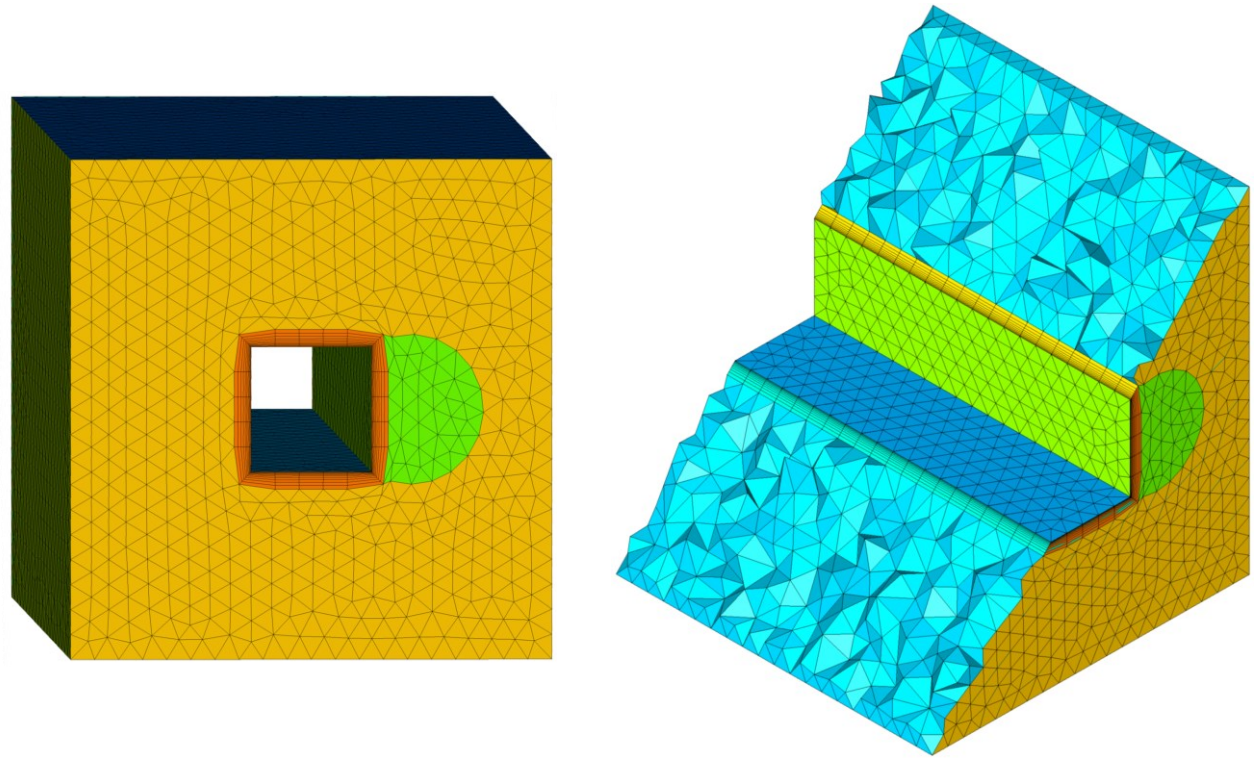

Fig. 27 The resulting finite element mesh of the model for generation of inflation layers.

5 Scalability of the Gmsh library

If the Gmsh library is compiled with OpenMP support, then the key steps in creating a finite-element mesh can be performed in parallel using multiple threads of the computer processor. The 1D and 2D meshing is then parallelized using a coarse approach, where multiple curves (1D meshing) or surfaces (2D meshing) are meshed in parallel independently of each other depending on the number of available processor threads. 3D meshing is then parallelized when using the HXT meshing algorithm (parameter `MeshAlgorithm3D = 10`) by using the so-called fine approach, where the meshing procedure on a specific volume of geometry is performed in parallel using multiple processor threads[12].

The Gmsh library was compiled with OpenMP support in the Linux operating system environment on the Karolina supercomputer of the IT4Innovations National Supercomputing Centre. The scalability of 2D meshing was then tested on one of the supercomputer nodes, having 2 processors, each with 64 cores and 128 threads, and with a clock frequency of 2.6 GHz each. The geometry of the test problem consisted of 128 identical surfaces, see Fig. 28.

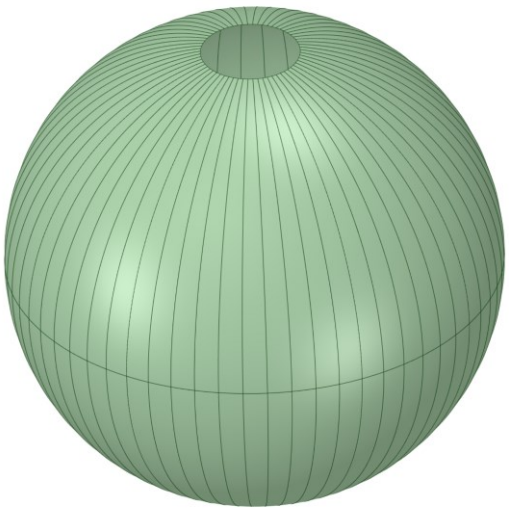

Fig. 28 Geometry for scalability test of 2D meshing.

On this test problem, a 2D mesh was then generated in the Gmsh library using the "Frontal-Delaunay" algorithm for different numbers of threads and the strong scalability of the Gmsh library was then measured in the range of 1 to 128 threads. The resulting strong scalability plot can be seen in Fig. 29.

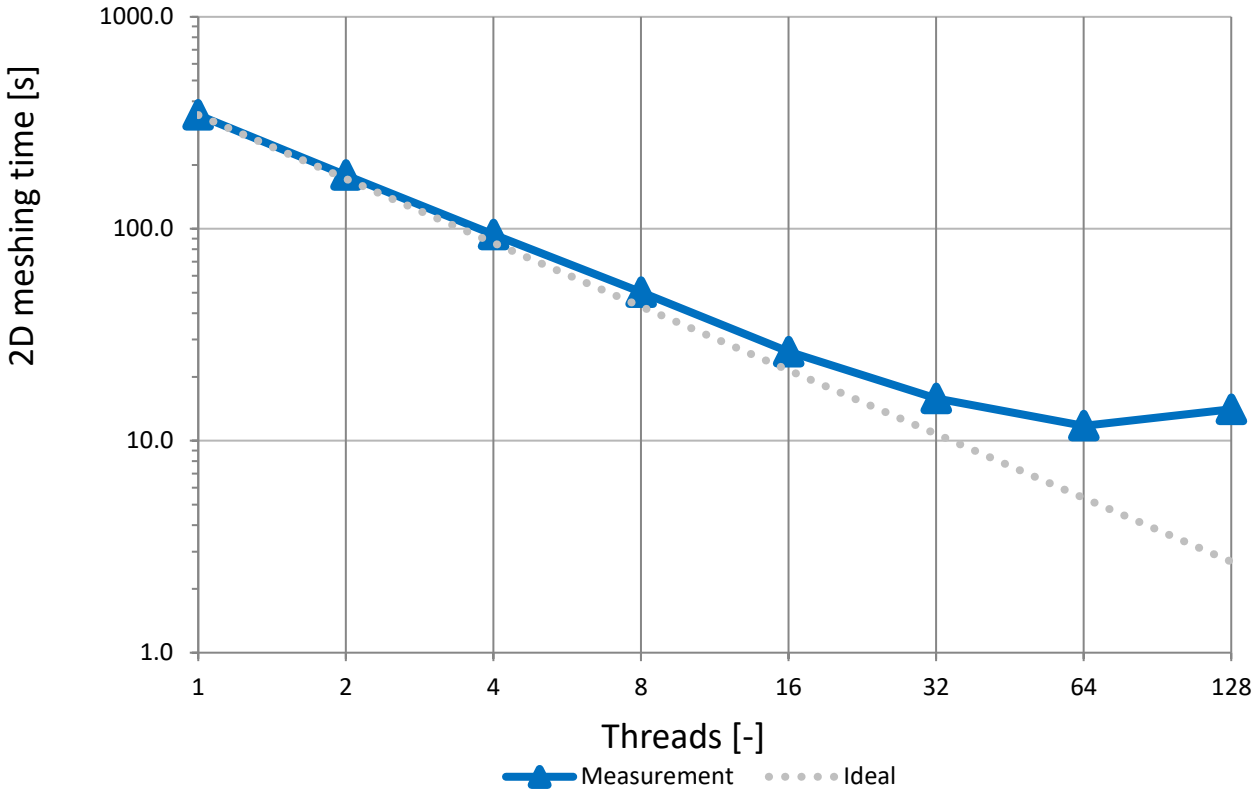

Fig. 29 Strong scalability plot of 2D meshing in the Gmsh library.

The values from the plot in Fig. 29, along with the number of nodes and finite elements, are listed in Tab. 12. The number of nodes and finite elements always vary slightly when running the same job repeatedly.

Tab. 12 Data of the strong scalability test of 2D meshing in the Gmsh Library.

Threads [-]	Nodes [-]	Finite elements [-]	Time [s]
1	3 734 989	7 470 234	344,1
2	3 734 988	7 470 233	179,1
4	3 735 027	7 470 310	94,0
8	3 735 005	7 470 267	50,2
16	3 735 003	7 470 262	26,3
32	3 735 010	7 470 276	15,8
64	3 734 935	7 470 126	11,8
128	3 734 967	7 470 191	14,1

The scalability of the 3D meshing algorithm “HXT” was also tested on the same hardware on two separate benchmarks. The geometry of the first benchmark was a simple cube, see Fig. 30 a). The geometry of the second benchmark was a car wheel rim, see Fig. 30 b), whose model was obtained by the author as a freely available model from the GrabCAD 3D model library.

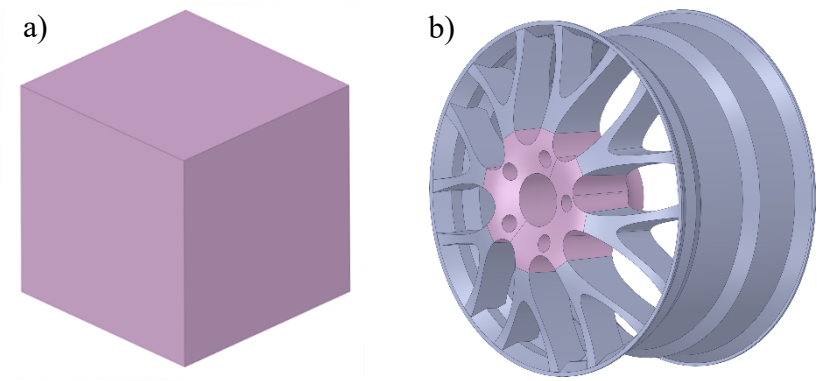

Fig. 30 Geometry of the first and second benchmark for the scalability test of 3D mesher “HXT”.

On these two benchmarks, a 3D mesh for different number of threads was then generated in the Gmsh library using the "HXT" algorithm and the strong scalability was measured in the range of 1 to 128 threads. The resulting strong scalability plots for the first benchmark (cube) and second benchmark (car wheel rim) can be seen in Fig. 31 and Fig. 32, respectively.

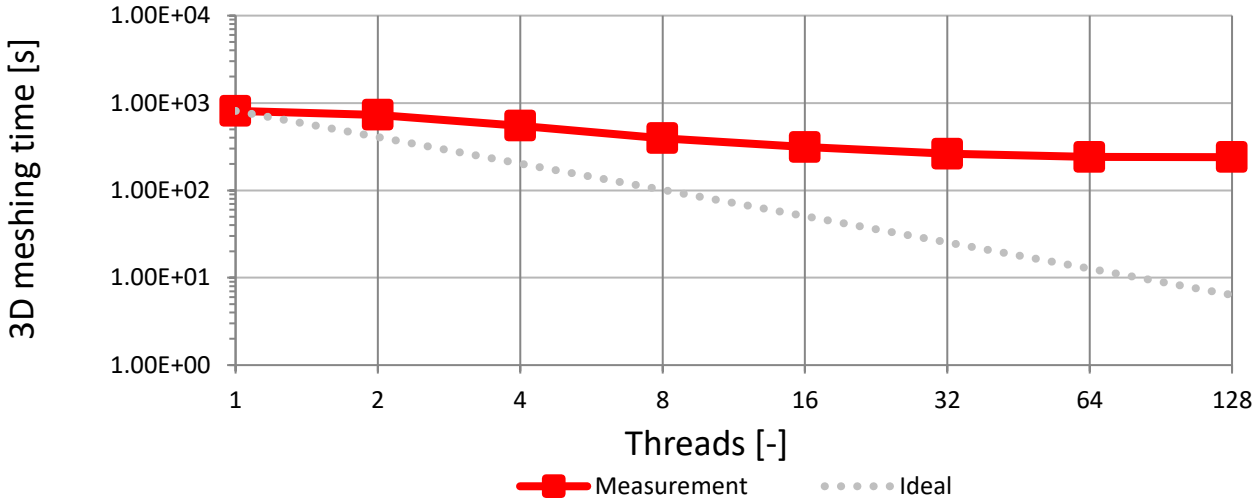

Fig. 31 Strong scalability plot of the 3D mesher “HXT” for the case of the first benchmark.

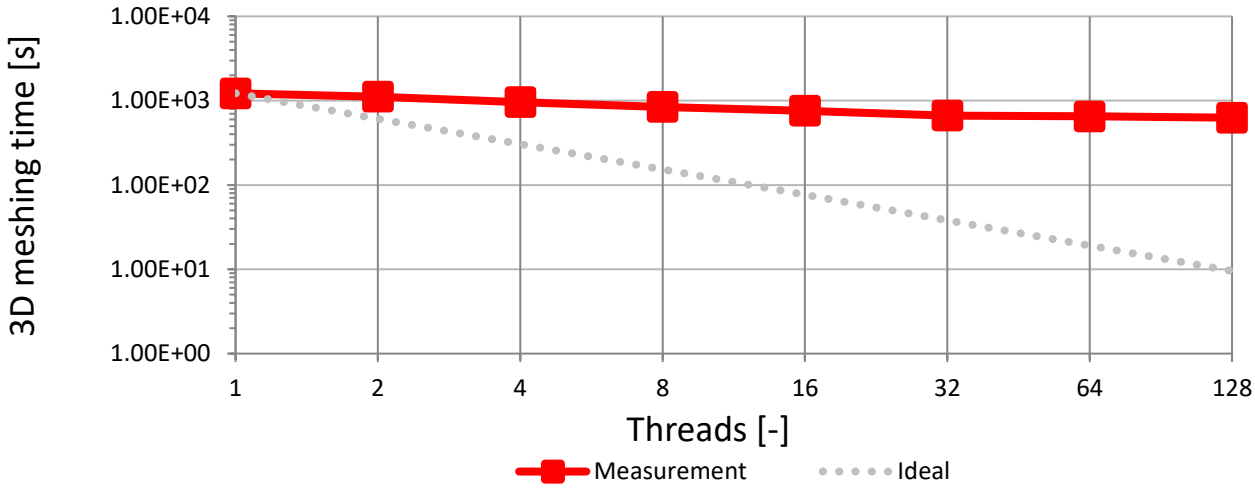

Fig. 32 Strong scalability plot of the 3D mesher “HXT” for the case of the second benchmark.

The values from the plots on Fig. 31 and Fig. 32 are listed along with the number of nodes and finite elements in Tab. 13, and Tab. 14, respectively.

Tab. 13 Data of the strong scalability test of the 3D mesher "HXT" for the case of the first benchmark.

Threads [-]	Nodes [-]	Finite elements [-]	Time [s]
1	63 937 485	399 810 852	812,0
2	63 928 263	399 753 549	729,8
4	63 928 172	399 750 288	548,6
8	63 925 537	399 738 046	394,7
16	63 926 665	399 737 308	312,9
32	63 919 854	399 700 342	261,7
64	63 923 615	399 716 852	240,8
128	63 922 421	399 706 484	239,8

Tab. 14 Data of the strong scalability test of the 3D mesher "HXT" for the case of the second benchmark.

Threads [-]	Nodes [-]	Finite elements [-]	Time [s]
1	32 007 110	390 665 482	1222,0
2	31 996 969	390 539 615	1115,5
4	31 997 547	390 552 500	960,4
8	31 999 064	390 542 436	841,9
16	31 995 520	390 502 668	758,6
32	31 998 373	390 538 860	666,3
64	31 999 604	390 545 847	652,4
128	31 998 020	390 532 424	628,3

References

[1] Geuzaine Ch., Remacle J.-F.: Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, International journal for Numerical Methods in Engineering, John Wiley & Sons, Ltd., vol. 79(11), 2009, p. 1309 to 1331, DOI: 10.1002/nme.2579, available from : https://gmsh.info/doc/preprints/gmsh_paper_preprint.pdf.

[2] Free Software Foundation: GNU General Public Licence Version 2 (GNU GPLv2), 1991, available from: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

[3] Kernighan B. W., Ritchie D. M.: The C programming language, Second Edition, Prentice-Hall, New Jersey, United States of America, 1988, ISBN: 9780131101630.

[4] Stroustrup B.: The C++ Programming Language, Fourth Edition, Addison Wesley, Reading, United States of America, 2013, ISBN: 0-321-56384-0.

[5] G. van Rossum: Python tutorial. Centrum voor Wiskunde en Informatica (CWI), Amsterdam, Netherlands, 2018, ISBN: 9781583483756, available from: https://ir.cwi.nl/.

[6] Bezanson J., Edelman A., Karpinski S., Shah V. B.: Julia: A fresh approach to numerical computing, SIAM Review, vol. 59(1), 2017, p. 65 to 98, DOI: 10.1137/141000671.

[7] Geuzaine C., Remacle J.-F.: Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, 2021, available from: https://gmsh.info/.

[8] ISO 10303-1:1994: Industrial automation systems and integration — Product data representation and exchange — Part 1: Overview and fundamental principles, International Organization for Standardization, Geneva, Switzerland, 1994.

[9] IGES/PDES Organization: Initial Graphics Exchange Specification 5.3. U.S. Product Data Association, North Charleston, United States of America, 1996, available from : https://web.archive.org/web/20120821190122/http://www.uspro.org/documents/IGES5-3_forDownload.pdf.

[10] Burns M.: Automated Fabrication — Improving Productivity in Manufacturing, Prentice-Hall, New Jersey, United States of America, 1993, ISBN: 978-0-13-119462-5.

[11] STRATASYS Ltd.: GrabCAD, 2021, available from : https://grabcad.com/.

[12] Marot C.: HXT Tet Mesher, 2020, available from: https://git.immc.ucl.ac.be/hextreme/tet-mesher-bench.

Appendix A Structure of the MSH File