Elmer FEM

open source multiphysical simulation software

ElmerGrid Manual

Peter Råback

CSC – IT Center for Science

January 20, 2026

ElmerGrid Manual

About this document

The ElmerGrid Manual is part of the documentation of Elmer finite element software package. ElmerGrid Manual gives instruction on the usage of ElmerGrid, the mesh generation and manipulation utility of Elmer.

The present manual corresponds to Elmer software version 26.1. Latest documentations and program versions of Elmer are available (or links are provided) at `http://www.csc.fi/elmer`.

Copyright information

Contents

Chapter 1

Overview of ElmerGrid

1.1 Introduction

ElmerGrid is a simple mesh generator and mesh manipulation utility. It is an appropriate tool for generating structured meshes for simple 1D, 2D, and 3D geometries. It can also read meshes generated by other mesh generators and manipulate them. Among the possible operations are scaling, changing of element type, defining boundaries, or partitioning the mesh for parallel solution, for example.

ElmerGrid was originally a side product of research done in the area of silicon carbide crystal growth. The main goal was to write a simple mesh generator for multiphysical problems where different meshes based on the same geometry were needed for different physical problems. Since that the software has been modified to meet the needs of Elmer development. Thus the name has been changed from Quickmesh to ElmerGrid. Because ElmerGrid was never designed to be a stand-alone mesh generator it has been developed in an uncontrolled manner. The data structures are not optimal for all the added features. Therefore the software may at times cause problems.

This chapter shortly describes the main capabilities of the program.

1.2 Usage

The executable binary of the program is named `ElmerGrid`. It may be started with no arguments at all. Then the following help text is automatically displayed.

```
Starting program Elmergrid
******************* Elmergrid ************************
This program can create simple 2D structured meshes consisting of
linear, quadratic or cubic rectangles or triangles. The meshes may
also be extruded and revolved to create 3D forms. In addition many
mesh formats may be imported into Elmer software. Some options have
not been properly tested. Contact the author if you face problems.

The program has two operation modes
A) Command file mode which has the command file as the only argument
   'ElmerGrid commandfile.eg'

B) Inline mode which expects at least three input parameters
   'ElmerGrid 1 3 test'

The first parameter defines the input file format:
1)  .grd      : ElmerGrid file format
2)  .mesh.*   : Elmer input format
```

```
3)  .ep       : Elmer output format
4)  .ansys    : Ansys input format
5)  .inp      : Abaqus input format by Ideas
6)  .fil      : Abaqus output format
7)  .FDNEUT   : Gambit (Fidap) neutral file
8)  .unv      : Universal mesh file format
9)  .mphtxt   : Comsol Multiphysics mesh format
10) .dat      : Fieldview format
11) .node,.ele: Triangle 2D mesh format
12) .mesh     : Medit mesh format
13) .msh      : GID mesh format
14) .msh      : Gmsh mesh format
15) .ep.i     : Partitioned ElmerPost format
16) .2dm      : 2D triangular FVCOM format


The second parameter defines the output file format:
1)  .grd      : ElmerGrid file format
2)  .mesh.*   : ElmerSolver format (also partitioned .part format)
3)  .ep       : ElmerPost format
4)  .msh      : Gmsh mesh format
5)  .vtu      : VTK ascii XML format


The third parameter is the name of the input file.
If the file does not exist, an example with the same name is created.
The default output file name is the same with a different suffix.


There are several additional in-line parameters that are
taken into account only when applicable to the given format.
-out str             : name of the output file
-in str              : name of a secondary input file
-decimals            : number of decimals in the saved mesh (eg. 8)
-relh real           : give relative mesh density parameter for ElmerGrid
                       meshing
-triangles           : rectangles will be divided to triangles
-merge real          : merges nodes that are close to each other
-order real[3]       : reorder elements and nodes using c1*x+c2*y+c3*z
-centralize          : set the center of the mesh to origin
-scale real[3]       : scale the coordinates with vector real[3]
-translate real[3]   : translate the nodes with vector real[3]
-rotate real[3]      : rotate around the main axis with angles real[3]
-clone int[3]        : make ideantilcal copies of the mesh
-clonesize real[3]   : the size of the mesh to be cloned if larger to
                       the original
-mirror int[3]       : copy the mesh around the origin in coordinate
                       directions
-cloneinds           : when performing cloning should cloned entities
                       be given new indexes
-unite               : the meshes will be united
-unitenooverlap      : the meshes will be united without overlap in
                       entity numbering
-polar real          : map 2D mesh to a cylindrical shell with given
                       radius
-cylinder            : map 2D/3D cylindrical mesh to a cartesian mesh
-reduce int[2]       : reduce element order at material interval
```

```
                        [int1 int2]
-increase            : increase element order from linear to quadratic
-bcoffset int        : add an offset to the boundary conditions
-discont int         : make the boundary to have secondary nodes
-connect int         : make the boundary to have internal connection
                       among its elements
-removeintbcs        : remove internal boundaries if they are not needed
-removelowdim        : remove boundaries that are two ranks lower than
                       highest dim
-removeunused        : remove nodes that are not used in any element
-bulkorder           : renumber materials types from 1 so that every
                       number is used
-boundorder          : renumber boundary types from 1 so that every
                       number is used
-autoclean           : this performs the united action of the four above
-multidim            : keep lower order entities even if they are not
                       boundaries
-bulkbound int[3]    : set the intersection of materials [int1 int2]
                       to be boundary int3
-boundbound int[3]   : set the intersection of boundaries [int1 int2]
                       to be boundary int3
-bulktype int[3]     : set material types in interval [int1 int2] to
                       type int3
-boundtype int[3]    : set sidetypes in interval [int1 int2] to type int3
-layer int[2] real[2]: make a boundary layer for given boundary
-layermove int       : apply Jacobi filter int times to move the layered
                       mesh
-divlayer int[2] real[2]: make a boundary layer for given boundary
-3d / -2d / -1d      : mesh is 3, 2 or 1-dimensional (applies to
                       examples)
-isoparam            : ensure that higher order elements are convex
-nonames             : disable use of mesh.names even if it would be
                       supported by the format
-nosave              : disable saving part altogether
-nooverwrite         : if mesh already exists don't overwrite it
-vtuone              : start real node indexes in vtu file from one
-timer               : show timer information
-infofile str        : file for saving the timer and size information


Keywords are related to mesh partitioning for parallel ElmerSolver runs:
-partition int[3]    : the mesh will be partitioned in cartesian main
                       directions
-partorder real[3]   : in the 'partition' method set the direction of
                       the ordering
-parttol real        : in the 'partition' method set the tolerance for
                       ordering
-partcell int[3]     : the mesh will be partitioned in cells of fixed
                       sizes
-partcyl int[3]      : the mesh will be partitioned in cylindrical main
                       directions
-metiskway int       : mesh will be partitioned with Metis using graph
                       Kway routine
-metisrec int        : mesh will be partitioned with Metis using graph
                       Recursive routine
```

```
-metiscontig          : enforce that the metis partitions are contiguous
-metisvol             : minimize total communication volume in Metis
-metisminconn         : minimize the maximum connectivity count in Metis
-metisseed int        : random number generator seed for Metis algorithms
-metisncuts int       : number of competing partitions to generate
-partdual             : use the dual graph in partition method (when
                        available)
-halo                 : create halo for the partitioning for DG
-halobc               : create halo for the partitioning at boundaries only
-haloz / -halor       : create halo for the the special z- or
                        r-partitioning
-halogreedy           : create halo being greedy over the partition
                        interfaces
-indirect             : create indirect connections (102 elements) in
                        the partitioning
-periodic int[3]      : periodic coordinate directions for parallel
                        & conforming meshes
-partoptim            : apply aggressive optimization to node sharing
-partnobcoptim        : do not apply optimization to bc ownership sharing
-partbw               : minimize the bandwidth of partition-partion
                        couplings
-parthypre            : number the nodes continuously partitionwise
-partzbc              : partition connected BCs separately to partitions
                        in Z-direction
-partrbc              : partition connected BCs separately to partitions
                        in R-direction
-metisbc              : partition connected BCs separately to partitions
                        by Metis
-partlayers int       : extend boundary partitioning by element layers

Keywords are related to (nearly obsolete) ElmerPost format:
-partjoin int         : number of ElmerPost partitions in the data to
                        be joined
-saveinterval int[3] : the first, last and step for fusing parallel data
-nobound              : disable saving of boundary elements in ElmerPost
                        format

Thank you for using Elmergrid!
Send bug reports and feature wishes to elmeradm@csc.fi
```

As may be seen the user may choose between several input and output formats. The format is specified by an integer number (or an appropriate string) and the file name that is given may or may not have the appropriate suffix.

1.3 Input formats

The ElmerGrid code has two different operation modes. In the one all the information is read from a command file. Alternatively all necessary data may be read from from the inline arguments.

The native formats of the code are thus the command file with the suffix `.eg` and the native geometry format that has the suffix `.grd`. Note that most commands that may be given in the command file may also be given in the geometry format.

Additionally there are several other input formats that helps in linking the Elmer software with other pre- and postprocessors. The other formats may not always be used to create a complete model with boundary conditions. Also their functionality does not include all the possible commands.

1) `.grd : ElmerGrid file format`
The native ElmerGrid format is described in a separate section.

2) `.mesh.* : Elmer output format`
Input format for ElmerSolver consisting of separate header, node, coordinate and topology files.

3) `.ep : Elmer input format`
This format is the same that ElmerSolver uses in saving the results to be postprocessed with ElmerPost.

4) `.ansys : Ansys input format`
This format is the input format created by the preprocessor of Ansys using a special Ansys macro. An example is given in later chapters.

5) `.inp : Abaqus input format by Ideas`
This format is the input format created by Ideas preprocessor for Abaqus.

6) `.fil : Abaqus output format`
This format is the output format created by the solver of Abaqus.

7) `.FDNEUT : Fidap neutral file`
Also created by Gambit, the preprocessor of Fluent.

8) `.unv : Universal mesh file format`
A subset of the universal mesh file format commands may be read by ElmerGrid.

9) `.mphtxt : Comsol mesh format`
The mphtxt format created by Comsol Multiphysics.

10) `.dat : Fieldview format`
This may be used with Gridgen preprocessor.

11) `.node,.ele: Triangle 2D mesh format`
Read in meshes created by Triangle 2D Delaunay mesh generator.

12) `.mesh : Medit mesh format`
Format of Medit mesh generation program.

13) `.msh : GID mesh format`
Format of GID mesh generation program.

14) `.msh : Gmsh mesh format`
Format of Gmsh mesh generation program.

15) `.ep.i : Partitioned ElmerPost format`
Partitioned Elmer results created by parallel computing.

16) `.2dm : FVCOM mesh format`
2D triangular format for the Unstructured Grid Vinite Volume Community Ocean Model, FVCOM.

1.4 Output formats

The supported output formats are all different formats of Elmer software. Additionally, there are some undocumented formats. Thus, if you need to export some data from Elmer format you may ask for the possibilities.

1) `.grd : ElmerGrid file format`
The native format, may be used to refresh the input file if it becomes corrupted or outdated.

2) `.mesh.*` : `ElmerSolver file format`
Format that may be used in Elmer calculations. In a single-processor version the input is divided to four different files with suffixes, `mesh.boundary`, `mesh.elements`, `mesh.header` and `mesh.nodes`.

The command-line parameters `-part` and `-metis` activate the partitioned input format for the ElmerSolver. The ending is `.part.*` and the files are saved to subdirectory with suffix `.part.n`, where `n` is the running number of the partition.

3) `.ep` : `ElmerPost file format`
Format for Elmer postprocessing. This is particularly handy for tuning the mesh before it is used in the solution. Then the only variable that is saved is the node index referred to by field variable `Number`.

4) `.msh` : `Gmsh file format`
Format for Gmsh mesh generator. This may be useful if one wants to use Gmsh to view entity numbers or mesh quality. Also one could use Gmsh as a tool to export meshes from Elmer project to a more commonly used format for interoperability with other software.

5) `.vtu` : `XML based unstructured VTK format`
This format may be read by software utilizing the VTK library such as Paraview. The format could be useful for studying the mesh prior to actual solution.

1.5   ElmerGrid command-line arguments

ElmerGrid has a lot of simple functionalities that may be activated with the right command-line arguments. Most of them may operate on all meshes, some are limited to 2D meshes, for example.

In the command-line operation mode there are only three mandatory arguments: The input format, the output format, and the name of the input file. For example, to read an ElmerGrid file named `angle.grd`, and to output the ElmerSolver files to a folder named `angle`:

```
elmergrid 1 2 angle
```

All other arguments are optional.

`-out` `String`
Gives an alternative name for the output file or directory. The default name is the same as the name of the input file attached with an appropriate suffix.

`-in` `String`
Sometimes there are may be more than one input mesh. Then the name of the secondary input file may be given after this keyword.

`-decimals` `Integer`
The number of decimals for Elmer mesh is by default set to 6. Sometimes more decimals may be needed and this command may then be used. This could be the case, for example, if the elements are small compared to the absolute values of the coordinates. The size of the ascii file is affected by the number of decimals and therefore the default is quite small.

`-relh` `Real`
Gives a relative density of the mesh. Value of one will have no effect while smaller values will result to a denser mesh. This is an easy way to increase mesh density of ElmerGrid meshing without having to edit the .grd file.

`-triangles`
The meshes consisting of rectangles may be divided into triangles. This operation may be performed on all 2D meshes, not only to the ones created using ElmerGrid format.

`-merge` `Real`
> Sometimes when creating the computational mesh the different domains are not properly joined. Then it is possible to use this option, where nodes that are close to each other are fused. The value given for the node separation should be smaller than the smallest element size. Otherwise there is a risk of ruining the mesh.
>
> For optimal computational efficiency the merging of nodes is done in two stages. In the first stage the nodes are ordered and in the second stage the nodes close to each other are checked for distance. If distance is smaller than the critical one, then the nodes are given one common coordinate and the other node is eliminated from the whole mesh.

`-order` `Real[DIM]`
> The nodes may be ordered if the original ordering is uneconomical. The three components gives the direction in which the nodes are reordered. More accurately, the nodes are ordered using the dot product $\vec{c} \cdot \vec{r}$ as a measure. The reordering of the degrees of freedom is usually performed within ElmerSolver and therefore it is usually not necessary to renumber the nodes within ElmerGrid.

`-centralize`
> Sets the center of the mesh to origin. The bounding box is used to define the size of the mesh.

`-scale` `Real[DIM]`
> This simply scales the coordinates of the mesh by multiplying with the given constants, i.e. $x := c_x x$, $y := c_y y$ and $z := c_z z$. This is useful, for example, if the mesh is in other units that is preferable.

`-translate` `Real[DIM]`
> This option translates the mesh with a given vector by simply adding the given values to the original coordinates, i.e. $\vec{r} := \vec{r} + \vec{r}_0$.

`-rotate` `Real[3]`
> The mesh may be rotated around the main axis. The rotation is done consecutively in all three angles. This feature may be handy in the debugging of some equation – in Cartesian grids the solution should be rotation invariant.

`-clone` `Integer[DIM]`
> The mesh may be copied so that the same structure is repeated over and over. The number of copies in each coordinate direction may be given independently. The total number of individual copies is their product.

`-clonesize` `Real[DIM]`
> By default the copies of the mesh are performed so that the new meshes are set directly besides the old ones. This flag may be used to define a larger size for the mesh which enables empty space being left between the meshes.

`-mirror` `Integer[DIM]`
> The mesh may be mirrored over the main axis. The arguments tell over which axis the mesh should be mirrored. If all three axis are active then the original mesh will be eight times the size of the original one. Note that the origin is assumed to be at zero in all directions.

`-unite`
> This option unites the ElmerGrid meshes. In every other format its currently not possible to create more than one mesh at a time.

`-unitenooverlap`
> As the previous one except avoids overlapping entity numbering. New numbers will be generated in order to avoid conflicts.

`-polar` `Real`
> This option maps the 2D mesh an a cylindrical shell with the radius given as a parameter.

`-cylinder`

This option assumes that the original 3D mesh was given in cylindrical coordinates (r, θ, z) and maps them to Cartesian coordinates (x, y, z).

`-reduce` `int[2]`

This feature reduces the element order of materials at the interval defined by the two parameters. It may be used to lower the element order only in parts of the geometry. For example, in fluid-structure-interaction the fluid could be modelled with linear elements while the structure might need second order elements.

`-increase`

Increase the element order in all the elements from one to two. This may be used to improve computational accuracy of existing linear meshes by making them quadratic.

`-bcoffset` `Integer`

Adds on offset to the boundary condition numbering. This may be useful for multimesh computations where it is advisable that the same boundary numbers are not defined in different meshes.

`-discont` `Integer`

Sometimes the field variables behave discontinuously over a short distance. For example, in solving the heat equation there may be a small gap between different bodies. To account for such discontinuities the mesh should have duplicate nodes over the discontinuity. This feature does exactly that and saves the boundary conditions for both sides.

`-connect` `Integer`

Makes the boundary to have internal connections between among its elements. This will enable connections in the resulting finite element matrices. The feature could be used in setting constraints in the finite element solution.

`-removelowdim`

Removes boundary elements that are two or more dimensions lower than the highest dimensional bulk elements.

`-removeunused`

Removes unused nodes from the mesh. In some mesh formats there are control nodes that may cause problems to ElmerSolver if these are not treated.

`-bulkorder`

Renumbers the body indexes so that every number is used. This feature may be handy particularly when importing meshes that have a very wild numbering of bodies.

`-boundorder`

Renumbers the boundary element types so that every number is used. This feature may be handy particularly when importing meshes that have a very wild numbering of boundary elements types.

`-autoclean`

This flag activates several of the above procedures that simplifies the mesh for solution purposes. The lower dimensional boundary elements are removed, and the body and boundary element types are renumbered.

`-bulkbound` `Integer[3]`

Sometimes the geometry does not include all the necessary boundary conditions. Then this inline parameter may be used to define a new boundary (first parameter) on the interval of the two given materials.

`-boundbound` `Integer[3]`

Sometimes the geometry does not include all the necessary boundary conditions. Then this inline parameter may be used to define a new boundary (first parameter) on the interval of two already existing boundaries. This makes it possible to define lines or even single points in 3D geometry that has only surfaces defined.

`-bulktype` `Integer[3]`
> Sometimes there is a multitude of bodies that may be united with this option. It may be easier to perform the further analysis if the number of bodies is not too large. For example, in ElmerPost and ElmerGUI a very lengthy list of bodies or boundaries is difficult to visualize.

`-boundtype` `Integer[3]`
> This option does exactly the same as the above but now for the boundary elements. Again the advantage may be that the command-file will be more economical as if the number of boundaries is reduced.

`-layer` `Integer[2] Real[2]`
> This option may be used to create boundary layers on an existing 2D mesh. The given boundary is extruded with the given number of layers. Also the total thickness of the layer and the desired ratio between first and last element is given.

`-layermove` `int`
> This option activates a filter that is used to map the node coordinates so that the boundary layers are fitted within the original geometry.

`-divlayer` `Integer[2] Real[2]`
> This option may be used to create boundary layers on an existing 2D mesh. The layer is created by splitting the triangles or quadrilaterals along the boundary to give a structured boundary layer mesh.

`-3d / -2d / -1d`
> This option is usually not needed as the files most often have the information of the space dimension. The exception is the creation of an example file. 3D is the default, thus the `-2d` should be used if one wants to create a 2D example file. Also some command-line parameter lists are by default assumed to have three values if 2D dimension is not given.

`-isoparam`
> In creating mapped meshes by ElmerGrid the higher-order nodes may sometimes be oddly displaced. This utility sets the intermediate nodes to lie on the connecting lines between the corner nodes.

`-nobound`
> Disable the saving of boundary elements. This may sometimes be useful if there are a lot of boundaries and one is just interested in the bulk elements.

`-nosave`
> Do not save anything, just do the preceding parts.

`-nooverwrite`
> If the mesh already exists don't overwrite it. This may be handy for automated processes where we don't want to redo steps.

`-timer`
> Show the elapsed time of the ElmerGrid execution.

`-infofile` `string`
> File for saving information about execution times and mesh size.

There are a number of inline parameters that are related only to the parallel Elmer computations.

`-partition` `Integer[DIM]`
> There are two methods to partition geometry for parallel computing. This method takes each coordinate direction and divides the mesh in the number of slices given in the parameter list. For example option `-part 2 3 5` would split the mesh into $2 \times 3 \times 5 = 30$ partitions. Each split is done so that the number of elements in each partition is almost equal. The method is best suited for simple geometries where more generic strategies usually do not work as well.

`-partorder`   `Real[DIM]`
> The simple partitioning is done by dividing the elements into subgroups. The division is done by first ordering the nodes. This keyword is followed by the normal vector of the desired ordering.

`-parttol`   `Real`
> Sometimes when using the geometric partitioning we use the ideal structured form of the mesh in the process. For example extruded meshes may result to partitions that could not have been extruded. The remedy is to let the elements be in the same partition if they are close enough to each other when ordered geometrically.

`-metiskway`   `Integer`
> This utility does the partitioning utilizing the Metis library, see `http://glaros.dtc.umn.edu/gkhome/views/metis`. Metis has subroutines for basic linear elements. Also higher order elements may be partitioned by first neglecting the higher order nodes.
>
> The suffix `kway` refers to the `PartGraphKway` routine of Metis. This may be used for the nodal and dual graph. The dual graph is recommended. The parameter is the desired number of partitions.
>
> Note that this option may not be active if the code has been compiled without the Metis library.

`-metisrec`   `Integer`
> This is as the previous one except the suffix `rec` refers to the `PartGraphRecursive` routine of Metis. This may be used for the nodal and dual graph. The dual graph is recommended. The parameter is the desired number of partitions.

`-metiscontig`
> This flag enforces that the Metis partitions are contiguous. This means that each partition is continuous. This may be numerically favourable and certainly needed by some algorithms.

`-metisseed`   `Integer`
> This allows the user to change the Metis seed. The purpose for this could be to study the effect on partitioning on the parallel solution. Some preconditioners are not fully parallel and their success may vary between different seed values.

`-partdual`
> The Metis algorithms that need a graph to work with by default use the nodal connectivity graph. As an alternative choice the user may enforce the use of elemental connectivity graph, i.e. the dual graph. This may be advantageous for the continuity of the partitions.

`-halo`
> This flag creates a halo for the partitioning. This must be provided, for example, in discontinuous Galerkin computations where at the boundaries the gradient information must also be computed. The halo is created for all nearest neighbours.

`-halobc`
> This flag creates a halo for the partitioning but only around the boundary conditions. No new neighbourhood connections are added to the shared nodes list among the pure halo nodes. This could be useful if halo information is needed only around the boundaries such as is the case for internal discontinuity creation in parallel.

`-haloz`
> This flag creates a halo for only in the z-direction such that the the next and previous partition in the z-direction is saved. The halo has been designed to work with rotating machines and to be used with `-partconnect` flag.

`-halor`
> As the previous one but creates a halo only in radial direction.

`-halogreedy`
> Create halo being greedy over the partition interfaces.

-indirect

Creates the indirect connections in the partitioning. These are connections in a finite element matrix that exist between two nodes but would not be apparent if only the partition in question would be studied. These should be not needed any more since ElmerSolver should be able to detect them on its own.

-periodic `Integer[3]`

If the solution of the mesh is known to be periodic ElmerSolver uses a special kind of boundary condition. Also in partitioned meshes used in parallel computation the periodic boundaries must be known as they require the flow of information between opposing sides. This option finds periodic boundaries in the given directions given by 0/1 logical values. If periodic boundary is found it is taken into account in saving and partitioning the mesh. The determination of the periodic nodes only works when there is one-to-one mapping between the nodes on the boundaries.

-partoptim

Applies aggressive optimization to node sharing. This may not pay off since the even execution of owned nodes is not that important. And this optimization does not affect the number of total nodes in each partition that is determined by the partitioning algorithm.

-partnobcoptim

Do not Apply optimization for the ownership of internal boundaries. There will be problems in parallel runs if an internal boundary with certain flux BCs does not have both parents in the same partition. Unfortunately enforcing both parents to be in the same partition may also be undesirable since it may destroy the initial good partitioning. Therefore this flag has been made optional but those that need internal BCs in parallel runs should be aware of the problem.

-partbw

Minimize the bandwidth of partition-partition couplings. This could be advantageous for the communication depending on the network connecting the nodes.

-parthypre

Number the nodes continuously partitionwise. Some linear solvers, such as Hypre, need this property. May be obsolete since this can be taken care by ElmerSolver also.

-partzbc

Partition connected boundaries separately to partitions in z-direction. Thereafter apply the main strategy (e.g. Metis) to the other elements.

-partrbc

Partition connected boundaries separately to partitions in r-direction. Thereafter apply the main strategy (e.g. Metis) to the other elements.

-metisbc

Partition connected BCs separately to partitions by Metis.

-partlayers

When using either two of the hybrid partitioning routines where boundary is partitioned first we may extend this partitioning to several layers of bulk elements using this flag.

These are keywords related to ElmerPost and are more or less obsolete.

-partjoin `Integer`

Gives the number of partitions in the data to be joined. This relates only to the output format 15 where individual .ep files are joined.

-saveinterval `Integer[3]`

In conjunction with the previous work gives the index of first and last timestep, and the timestep interval that should be used in the merging.

-nobound

Disable saving of boundary elements in ElmerPost format.

Chapter 2

ElmerGrid format

The native mesh definition format of the program is the ElmerGrid format that has the suffix `.grd`. If the user gives a name for the input file that doesn't exist, then a simple geometry file is automatically created by the program. This may then be edited by any text editor to suit the needs of the particular case.

The ElmerGrid parser is not very bright. It knows only a handful of keywords. If the default values are used the corresponding commands may be neglected. The commands usually assume a format where the keyword is followed by its arguments. If the arguments are on the same line then an equality sign "=" should be used in between. The parser is not case sensitive and thus the keywords may be written with capital or small letters, or any combination of them.

2.1 ElmerGrid keywords

Version `210903`

Gives the version of the current input file. This should not be changed unless the format is changed. The parser should be downward compatible also with the older format of the input file but the new format is preferred.

MATC `Logical`

Setting this keyword to be true activates the MATC parser. As an effect all statements starting with the $ sign will be evaluated by MATC. This enables parametrized mesh generation, for example. For more information look at the MATC manual. Note that if this does not work the version of ElmerGrid may have been compiled without MATC.

Coordinate System `String`

The choices are Cartesian 1D, Cartesian 2D, Cartesian 3D, Polar and Axisymmetric. Axisymmetric and Cartesian 2D have currently no difference. A Cartesian 3D grid is the default.

Subcell Divisions `Integer[DIM]`

The mesh is composed of subcells that are topologically squares. This keywords should be followed by the number of subcells in the main directions, n_1, n_2 and n_3. Thus the number of arguments may be 1, 2 or 3 depending on the dimension of the geometry definition.

Subcell Limits i `Real[]`

The size of the subcells are defined by $n_i + 1$ boundaries in each coordinate direction. These values define the physical dimensions of the mesh before possible manipulation. This keyword exists for all active coordinate directions.

Subcell Sizes i `Real[]`

An alternative way to give the subcell sizes is to give the size of each subcell in each coordinate direction. The subcell limits are then the cumulative sum of these sizes.

`Subcell Origin i` `Real[]`

 When the subcells are defined by the sizes the user may also define the origin in each direction. The origin may be a real number or also some of the following strings, `Left`, `Center`, `Right`.

`Material Structure` `Integer[][]`

 The subcells form a $n_1 \times n_2$ matrix in 2D. The material structure of this plane may therefore be given by a matrix consisting of integers relating to different materials. These numbers may then later be referred when creating the mesh and boundary conditions. In 1D the structure is simply a vector. In 3D the material structure refers to the structure of the xy-plane before extrusion or rotation takes place.

`Materials Interval` `Integer[2]`

 Elements are created only for those materials that lie in the interval defined by the two integers. This is on easy way to define different meshes for different physical phenomena. In 3D the interval refers to the materials that should be present in the extrusion.

`Extruded Structure`

 The keyword may be used to define the layered structure of the geometry. Otherwise the extruded structure is just the same as the material structure in the plane. The definition must include n_3 lines with three integers on each. The first two integers define the materials interval that is extruded and the third one the new material index of the extruded layer. Materials outside the interval are not extruded and therefore materials may selectively be dropped out during extrusion process. If the new material has value zero then the corresponding material layer in the extruded structure remains unaltered. This feature may, for example, be useful in MEM-applications.

 When the 2D geometry is extruded the rectangular elements 404, 408 and 409 result to elementtypes are 808, 820 and 827 (using Elmer notation). Also linear triangles may be extruded and they result to linear prisms of type 606.

`Extruded Material Mappings`

 Sometimes the previous way of defining the extruded structure is not flexible enough. Then the user may given for each extruded layer the mapping between initial and extruded material indexes. This enables any kind of extruded structures be created. There will be n_3 lines and each line should have `MAX(material)` entries.

`Extruded BC Offset` `Integer`

 When the mesh is extruded into 3D new boundaries will be created. By default the numbering of boundaries is such that each boundary index number is used and a heuristic method for the numbering defines the indexes. This may be difficult for the user since there is no way of knowing for sure what the indexes will be. This keyword enforces the new BCs on the extruded layers to have deterministic indexes. The first layer will get indexes $N_{offset} + N_{2D}$ the following will have an additional offset of N_{offset} etc.

`Extruded Helicity` `Real`

 When the geometry is extruded in z-direction the user may optionally give a constant skew to the mesh by giving the helicity of the mesh using this keyword. For example, value 360.0 means a full rotation during the extruded distance.

`Geometry Mappings`

 The grid is by default rectangular since all the subcells are also rectangular. Non-rectangular shapes may be introduced by mapping the straight lines separating the subcells to lines represented by analytical functions. The mapping may be performed in the xy-plane or out of the plane. The number of the mappings is calculated automatically and therefore the field should be ended by `End`.

 If the mode of the mapping is more than 50 then the mapping is performed out of plane while mappings in the plane is the default. Mapping out of the plane makes the 2D plates to become 3D shells.

 This keyword activates mappings as in the case of mappings in plane but now the mapping is performed out of the plane. Thus the 2D geometry becomes 3D geometry. This way some simple 3D shells may be created. If in addition, the coordinate system is set to be polar this third coordinate is treated as radius r and the second coordinate as angle ϕ.

Polar Radius   Real
> If the coordinate system is chosen to be polar then the values defining the grid are (z, θ). When saving the mesh these are mapped into cartesian coordinates. There may be an offset in the radius when using the polar coordinates. This value is then added to the value of the r-coordinate before making the coordinate transformation. This would result into simple pipes, for example. In addition there is a possibility to have the radius change as a function of position using special mappings.

Revolve Radius   Real
> There are three keywords that may be used to define how a 2D mesh is revolved around the y-axis to create 3D cylindrical meshes. Defining any of the keywords enforces revolution. Simple rotation would result to singular elements at the origin. Therefore a square block is set at the origin. By default this square has the same x-coordinate as the first subcell in x-direction. In rotation the material structure remains unaltered.

Revolve Blocks   Integer
> The keyword has a parameter 1, 2, 3 or 4 depending on how many degrees of rotation is desired. The values correspond to rotation of 90, 180, 270 or 360 degrees. If there is less than 4 blocks the program finds the sides that form the symmetry axis and creates additional boundaries from them.

Revolve Improve   Real
> This keyword controls the quality of the rotated mesh. The rotation creates elements that are not rectangular as the whole angle of 360 degrees needs to be divided between 3 elements. The default value for this keyword is one which means that the inner block is a square. The value zero results to the angle being equally divided to three 120 degree angles by the three elements.

Revolve Curve Direct   Real
> The rotated structure may have a curve. This defines the direct part of the curve.

Revolve Curve Angle   Real
> This defines the angle of the curve.

Revolve Curve Radius   Real
> This defines the radius of the curve.

Boundary Definitions
> There may be several boundaries. Each is defined by a line of four parameters. For example,
>
> `type out int double`
>
> Note that this field should be finished by `End`.
>
> Each boundary requires a `type` that refers to the number of the boundary condition. It may then later be referred to when setting up the boundary conditions. The boundaries are created in such a way that the material `int` should have elements created. The `out` may be an empty subcell. If the value of the flag `double` is 2 or more secondary nodes on the boundary will be created. This enables the use of discontinuous boundary conditions.
>
> It is also possible to create boundary conditions on a given side of a material. The number -1 refers to **down**, -2 to **right**, -3 to **top** and -4 to **left**. This means that the boundary is created on the given side of the material that is specified by the other index.
>
> In order to make the creation of the boundary condition more robust there are some special numbers that may be used. -11 refers to **smaller**, -9 to **bigger**, and -10 to **anything**. For example, if the `intmat` is 5 and the `outmat` is -9, then the boundary is created for all materials that have a larger material number than 5 and have a common boundary with material 5. This makes it possible to create several material boundaries with one definition.

Numbering   Vertical/Horizontal
> The keyword enforces the numbering of nodes and elements. The numbering affects the bandwidth of the resulting matrix equation. The numbering should rather be performed by the shorter axis.

Reference Density `Real`
> This keyword defines the local global mesh parameter h that sets the maximum size of an element. Additionally a local element density may be defined with respect to this.

Surface Elements `Integer`
> This keyword may be used to roughly determine the number of elements on the surface. This unique feature makes it possible to change the mesh density by altering just one figure in the mesh definition file. Using the constraints on the relative mesh densities the program then tries to match this given number.

Volume Elements `Integer`
> In 3D it is not as easy to obtain the desired number of elements. In this case the desired number of surface elements is altered iteratively until also the number of volume elements is close to the desired one.

Volume Nodes `Integer`
> Instead of trying to meet the desired number of elements one may also try to optimize the number of nodes using this keyword.

Element Degree `Integer`
> The mesh generator may produce linear, quadratic and cubic elements. The elements are initially always topologically rectangles and may have 4, 5, 8, 9, 12 or 16 nodes.

Element Midpoints `Logical`
> For higher order elements the user may choose whether to create nodes inside the elements. For example a second order rectangle may have 8 or 9 nodes. The default is `False`.

Triangles `Logical`
> ElmerGrid makes originally structured quadrilateral meshes. These may be divided into triangles using the shorter of the diagonals. The resulting mesh is still structured and often inferior in computations. For some purposes triangular meshes may however be favored. The default is `False`.
>
> The quadrilateral elements may have 4, 5, 8, 9 or 16 nodes. The elements with 4, 9 and 16 nodes are divided into two and the element with 5 nodes into four, correspondingly. The 8-node element cannot be divided.
>
> If the mesh is also extruded the triangles become prisms. In this case only linear triangles are applicable.

Elementtype `Integer`
> An alternative way to define the elementtype is to define the element type as in Elmer software. The allowed types are 303, 306, 310, 404, 405, 408, 409, 412 and 416. Then the three flags above are tuned accordingly.

Coordinate Ratios `Real[DIM-1]`
> By default the relative density of the coordinate directions is the same. This keyword affects the mesh density ratio between different directions. Thus the dimension is one less than the space dimensions.

Element Ratios i `Real[]`
> Defines the ratio of the first and last element in the particular subcell in direction `i`. The default value is 1. Does not compare the mesh density with other subcells. The ratio must be given for each coordinate direction separately. Value close to unity leads to linear meshing. Negative values activate symmetric density profile within the subcell.

Element Densities i `Real[]`
> This keyword refers to densities compared to other subcells in direction `i`. The default value is 1. A larger value means local mesh refinement on that particular subcell.

Minimum Element Divisions `Integer[DIM]`
> This keyword defines the minimum number of elements within each subcell for each coordinate direction. The default is 1 1 2, this is also the set of minimum values that may be used.

`Element Divisions i` `Integer`

This keywords sets the number of elements in direction `i` in the subcells absolutely. Therefore it overrides the desired number of elements and the relative element densities. By default the relative mesh density definition is used.

`Start New Mesh`

The format includes the possibility to generate multiple meshes using one input file. This keyword invokes the start of a new mesh. All properties are inherited from the previous mesh and therefore the new mesh may be defined only by few changes.

2.2 Command file keywords

Sometimes the necessary commands in using the inline parameters should create an exhaustive and noninformative list. Therefore most of the inline parameters may also be given in an command file. Here the description only deals with the command file syntax. For other information on these commands a more accurate description will be found in the previous chapter where the inline arguments were explained.

Note that these commands may also be put to the end of the ElmerGrid mesh definition file.

`Input File` `String`

The name of the input file. The default name the same as the name of the command file (if any) attached with an appropriate suffix.

`Output File` `String`

The name of the input file. The default name is the same as the name of the input file attached with an appropriate suffix.

`Input Mode` `String/Integer`

The name of the Input mode or the corresponding integer. The choices are ElmerGrid(1), ElmerSolver(2), ElmerPost(3), Ansys(4), Ideas(5), Abaqus(6), Fidap(7), Easymesh(8), FemLab(9), Fieldview(10) and Triangle(11).

`Output Mode` `String/Integer`

The name of the Input mode or the corresponding integer. The choices are ElmerGrid(1), ElmerSolver(2), ElmerPost(3).

`Decimals` `Integer`

The number of decimals in output.

`Triangles` `Logical`

Divide rectangles into triangles.

`Merge Nodes` `Real`

Merges nodes close to one another, $|\vec{r} - \vec{r}_0| < \varepsilon$.

`Unite` `Logical`

Unite different meshes, default is `False`.

`Order Nodes` `Real[DIM]`

Orders the nodes using $\vec{c} \cdot \vec{r}$ as the measure.

`Scale` `Real[DIM]`

Scale the mesh coordinates.

`Translate` `Real[DIM]`

Translate the mesh with the given vector.

`Rotate` `Real[DIM]`

Rotate the mesh around the main axis with the given degrees.

Clone `Integer[DIM]`
> Make a multiple copies of the mesh.

Clone Size `Real[DIM]`
> The size of the mesh to be cloned.

Mirror `Real[DIM]`
> Mirror the mesh around the main axis.

Polar Radius `Real`
> Gives the radius of the 2D mesh in polar coordinates and also enforces the new coordinate system.

Reduce Degree `Integer[2]`
> Reduces the element degree in the given interval.

Increase Degree `Logical`
> Increase the degree. Default is `False`.

Power Elements
> This keyword activates the p-elements. There may be several p-element definition. Is has a separate line and defines with three integers the interval on which the p-element is applied and the power of the element. The field should be ended with `End`.

Partition `Integer[DIM]`
> Activates the simple mesh partitioning. Gives the number of divisions in each coordinate direction.

Partition Order `Real[DIM]`
> The ordering used in making the simple partitioning.

Metis `Integer`
> Activates the Metis partitioning.

Periodic `Integer`
> Enforces periodic boundary conditions with the partitioning algorithms.

Material Boundary
> Define boundaries between materials. Each boundary is an a separate line and the field should be ended with `End`

Boundary Boundary
> Define boundaries between boundaries. Each boundary is an a separate line and the field should be ended with `End`

Renumber Material
> Renumbers the materials on the given interval to be of desired type. Each renumbering is an a separate line and the field should be ended with `End`

Renumber Boundary `Integer[3]`
> Renumbers the boundaries on the given interval to be of desired type. Each renumbering is an a separate line and the field should be ended with `End`

Isoparametric `Logical`
> Sets the intermediate nodes in elements to be in between the corner nodes.

No Boundary `Logical`
> Disables the saving of boundary information. Default is `False`.

Extruded Divisions `Integer`
> A two-dimensional mesh may be extruded to multiple layers. This keyword gives the number of subcell in the extruded direction n_3.

`Extruded Limits` `Real[$n_3 + 1$]`
    The dimensions of the extruded layers.

`Extruded Elements` `Integer[$n_3$]`
    The number of elements in each extruded layer.

`Extruded Ratios` `Real[$n_3$]`
    Defines the ratio of the first and last element in the particular subcell in the extruded direction.

`Extruded Structure`
    Functionality as in the `ElmerGrid` geometry format.

2.3   Mapping modes

The use of different mapping options is quite an unintuitive matter. Unfortunately it is the only way to create non-rectangular shapes in ElmerGrid. The mesh mapping is activated by the keyword `Geometry Mappings`. There are several mapping modes that define the type of mapping.

Most of the mappings operate on the lines separating different subcells. Positive mode refers to mapping of horizontal lines, and negative mode to mapping of vertical lines. The limits of the mapping define the distance at which the mapping of the line settles down. If the distance is large compared to the size of geometry then the same mapping is performed to the whole geometry.

Several mappings may be applied at the same time. The coordinates are then mapped as a linear combination of the individual mappings. Mappings may also be performed consecutively up to five levels. This is obtained by adding the figure 10 to the mode. Mappings of the same level are done at the same time while mappings of different levels are done at an increasing order of modulo 10s.

In order to perform mappings out of plane a value of 50 may be added in the mapping mode.

The parameters used in the mapping vary from case to case. Note that if a vertical line is mapped the coordinates x and y are switched.

1: piecewise linear
    Parameters are (x_0, dy_0), (x_1, dy_1), (x_2, dy_2), ... Between the points linear interpolation is used.

2: piecewise power law
    Parameters are (x_0, dy_0), b_1, (x_1, dy_1), b_2, (x_2, dy_2), ... Between the points the values for dy are interpolated from $a_i(x - x_i)^{b_i}$.

3: piecewise circle arc
    Parameters are (x_0, dy_0), r_1, (x_1, dy_1), r_2, (x_2, dy_2), ... Between the points the values for dy are calculated assuming that there is a circle with radius r_i drawn through the lines. If radius is larger than zero the origin of the circle is above the mapped line and vice versa.

4: line to circle
    Parameters are $x_0, r_0, x_1, r_1, \ldots$ Different from other models also the other coordinate is transformed. In order to get a circle with radius r_i as a result there must initially be a square with side length $2r_i$. The line to be mapped is originally one of the sides of this squares.

5: line to sinus
    Parameters are x_0, x_1, c, dy_1, where c is the number of waves between x_0 and x_1 and dy_1 is the height of the wave.

6: line to power series
    Parameters are $x_0, x_1, a_0, \ldots, a_i$ and the displacement is then $\sum_i a_i u^i$, where u is x normalized between 0 and 1.

7: line to power series variation
    Parameters are $x_0, x_1, a_0, \ldots, a_n$ and the displacement is then $\sum_i a_i p_i(u)$, where u is x normalized between 0 and 1. The polynom $p_0 = 1$, $p_1 = u$, $p_2 = u(u - 1)$, $p_3 = u(u - 1/2)(u - 1)$, $p_4 = u(u - 1/3)(u - 2/3)(u - 1), \ldots$

8: piecewise constant angle
Parameters are (x_0, ϕ_0), (x_1, ϕ_1), (x_2, ϕ_2), . . . This is used to create 3D shells only. The angle of the shell is ϕ_i if $x \in [x_i, x_{i+1}]$. May be used to create polygonal shells.

Chapter 3

ElmerGrid mesh generation examples

3.1 2D examples

Unit square: `rect.grd`

The first example is a simple unit square, as shown in Figure 3.1a. The input file is the simplest possible as the geometry consists of 1×1 subcells. The boundaries in x- and y-directions are 0 and 1. The desired number of elements is 100 and the distribution is uniform. There is one boundary created that includes all four sides of the square. They are found by the fact that they are at the 0-1 material interface where 0 is the default material outside the square. The input file is given below.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 1 1
Subcell Limits 1 = 0 1
Subcell Limits 2 = 0 1
Material Structure in 2D
  1
End
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1        0        1        1
End
Element Degree = 1
Surface Elements = 100
```

The elements do not need to be bilinear. In 2D also second and third order elements are possible. They are created by setting the number of nodes to 5, 8, 9, 12 or 16 depending on the preferred element type. Some types may be divided to triangles. The first variation of the rectangle case, shown in Figure 3.1b has 200 second order triangles that are created with the lines.

```
Element Degree = 2
Element Innernodes = True
Triangles = True
```

Within the subcells the division of elements may be controlled in both directions independently, as shown in Figure 3.1c. A variation of the unit square has different division in the x and y-directions. In x-direction the last and first elements have a size ratio of 2.0. In y-direction the division is symmetric so that the middle element is 3.0 times larger than the side elements.

```
Element Ratios 1 = 2.0
Element Ratios 2 = -3.0
```
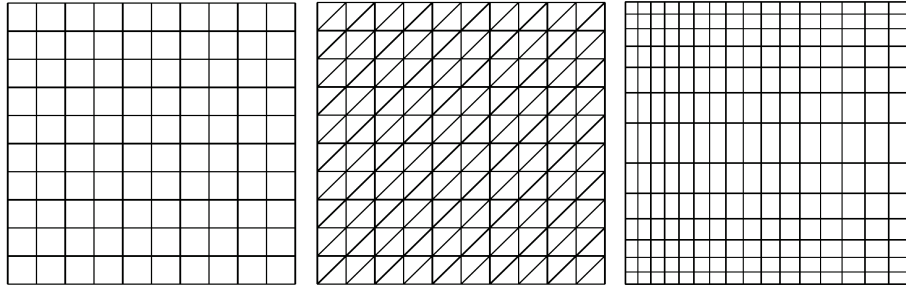

Figure 3.1: Unit square with different meshes. a) 100 bilinear rectangles b) 200 quadratic triangles c) 200 bilinear elements with nonuniform division.

Tube: `tube.grd`

The next case could be a simple 2D tube, as shown in Figure 3.2a. Now the width is 5.0 which is reflected in the subcell boundaries. Again the mesh has some nonuniform division.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 1 1
Subcell Limits 1 = 0 5
Subcell Limits 2 = 0 1
Material Structure in 2D
  1
End
Materials Interval = 1 1
Boundary Definitions
! type     out     int     double   of the boundaries
  1        -1      1       1
  2        -2      1       1
  3        -3      1       1
  4        -4      1       1
End
Numbering = Vertical
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 200
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 2
Element Ratios 2 = -2
Element Densities 1 = 1
Element Densities 2 = 2
```

The variation of the case includes two mappings. The first mapping in Figure 3.2b maps the lower horizontal boundary so that it goes linearly between four points. The second mapping in Figure 3.3b sets the upper horizontal boundary to be a sinusoidal function with three periods.

```
Geometry Mappings
! mode   line   limits(2)   Np   params(Np)
  1       0      1.0   1.0   8    0.0 0.1 1.0 -0.1 3.0 0.1 5.0 -0.1
  5       1      1.0   1.0   4    0.0 5.0 3.0 -0.3
End
```
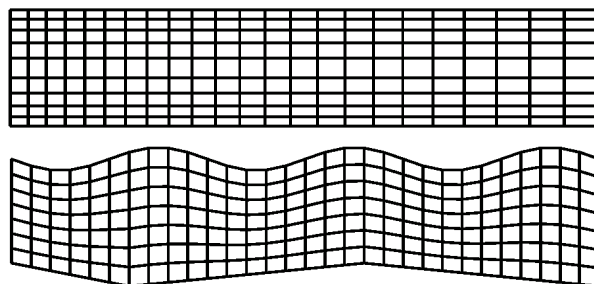

Figure 3.2: Tube with 200 bilinear rectangles. a) straight tube with two kinds of mesh refinement b) sinusoidal and linear boundary mappings

Square with holes: `holes.grd`

The subcell structure may be quite complicated. This is reflected by an example of a square with a large number of square holes, as shown in Figure 3.3a. Even at such a complicated case almost the desired amount of elements may be created quite quickly. The second case in Figure 3.3b with 81 holes and 112000 elements was created in about a second an a normal PC.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 9 9
Subcell Limits 1
  0 1 2 3 4 5 6 7 8 9
Subcell Limits 2
  0 1 2 3 4 5 6 7 8 9
Material Structure in 2D
  1   1   1   1   1   1   1   1   1
  1   2   1   2   1   2   1   2   1
  1   1   1   1   1   1   1   1   1
  1   2   1   2   1   2   1   2   1
  1   1   1   1   1   1   1   1   1
  1   2   1   2   1   2   1   2   1
  1   1   1   1   1   1   1   1   1
  1   2   1   2   1   2   1   2   1
  1   1   1   1   1   1   1   1   1
End
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1         2        1        1
  2         0        1        1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
```

```
Triangles = False
Surface Elements = 1024
```
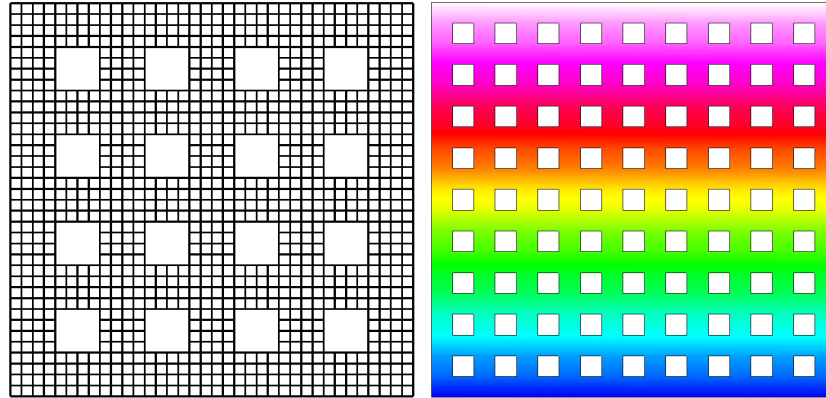

Figure 3.3: Squares with holes: a) square with 16 square holes and 1040 elements, b) square with 81 square holes and 112000 elements.

Cross: `cross.grd`

To show that multiple mappings may be used at the same time a more complicated case is given. Here two horizontal and two vertical lines are mapped, as shown in Figure 3.4a. The mesher has the nice property that the same file may be used to created multiple meshes with different settings. The other mesh in Figure 3.4b is now the external mesh and it is divided to triangles.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 3 3
Subcell Limits 1 = 0 2 3 5
Subcell Limits 2 = 0 2 3 5
Material Structure in 2D
  2    1    2
  1    1    1
  2    1    2
End
Geometry Mappings
! mode  line   limits(2)   Np   params(Np)
  1     1     2      1      8    0 -1 2 0 3 0 5 -1
  1     2     1      2      8    0 1 2 0 3 0 5 1
  -1    1     2      1      8    0 -1 2 0 3 0 5 -1
  -1    2     1      2      8    0 1 2 0 3 0 5 1
End
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double    of the boundaries
  1        0        1        1
  2        2        1        1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
```

```
Triangles = False
Surface Elements = 400
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 0.5 1 2
Element Ratios 2 = 0.5 1 2
Element Densities 1 = 1 1 1
Element Densities 2 = 1 1 1

Start New Mesh
Materials Interval = 2 2
Boundary Definitions
! type      out      int      double   of the boundaries
  1        0        2        1
  2        1        2        1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
Triangles = True
Surface Elements = 400
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 0.5 1 2
Element Ratios 2 = 0.5 1 2
Element Densities 1 = 1 1 1
Element Densities 2 = 1 1 1
```
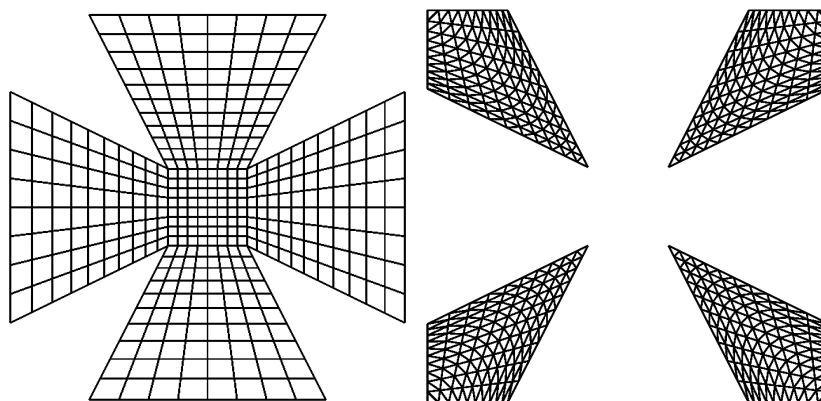

Figure 3.4: Cross with shape used to create 2 computational meshes: a) the internal mesh of 400 bilinear rectangles b) the external mesh of 800 linear triangles.

Hand-weight: `weight.grd`

First we will create a 2D mesh, as shown in Figure 3.5a. Then we will use extrusion to create a 3D mesh, as shown in Figure 3.5b.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 3 3
```

```
Subcell Limits 1 = 0 2 4 6
Subcell Limits 2 = 0 0.8 1.2 2
Material Structure in 2D
  1    0    1
  1    1    1
  1    0    1
End
Geometry Mappings
! mode  line  limits(2)  Np  params(Np)
  4     0     2      2      4   1 1 5 1
End
Materials Interval = 1 2
Boundary Definitions
! type    out      int      double   of the boundaries
  1        0        1        1
End
Numbering = Vertical
Element Degree = 1
Element Innernodes = False
Triangles = True
Surface Elements = 500
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 1 1 1
Element Ratios 2 = 1 1 1
Element Densities 1 = 1 1 1
Element Densities 2 = 1 2 1
```

The extrusion requires that there are subcells also in the z-direction. In this case only one subcell is set.

```
Coordinate System = Cartesian 3D
Subcell Divisions = 3 3 1
```

In addition all information that is given for x and y must now also be given for z. For example, the subcell boundaries in the extruded direction must be given.

```
Subcell Limits 3 = 0.0 0.5
```
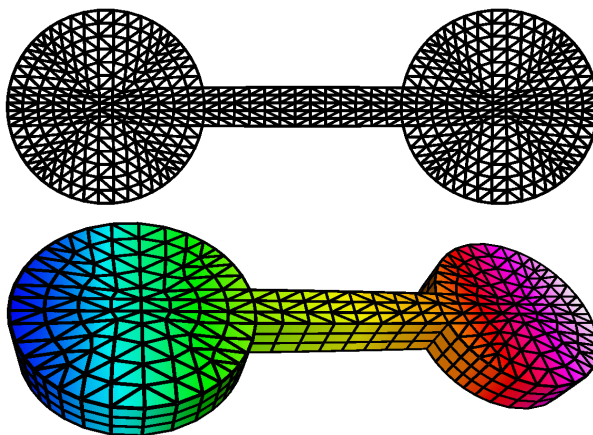

Figure 3.5: Hand-weight consisting of two spheres and a joining tube a) 2D mesh with 1000 linear triangles b) extruded 3D mesh with 1200 prisms.

3.2 3D examples

Shell of waves: `waves.grd`

The originally 2D mesh may be given a third dimension with the help of mapping. In this example two sinusoidal mappings are used. They have a amplitude with different sign and therefore there is a phase-shift of 180 degrees between the two opposing sides, as shown in Figure 3.6.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 1 1
Subcell Limits 1 = 0 100
Subcell Limits 2 = 0 100
Material Structure in 2D
  1
End
Geometry Mappings
! mode  line   limits(2)   Np   params(Np)
  55    0     50     50     4    0 100 5 10
  55    1     50     50     4    0 100 5 -10
End
Materials Interval = 1 1
Boundary Definitions
! type     out       int       double    of the boundaries
  1        0         1         1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 10000
```
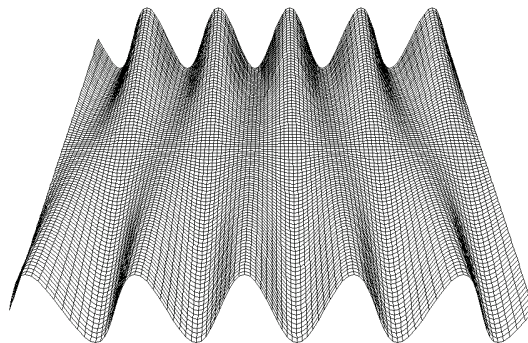

Figure 3.6: A shell with a wave pattern

Shell consisting of two adjacent cones: `cones.grd`

There are some cases where a mesh consisting of 2D elements may be used to model also 2D geometries. This is the case for thin shells, for example. ElmerGrid does offer limited functionality for creating these. Only shells with rotational symmetry are possible to be created. Then the computational mesh is done for the z and ϕ variables. The missing radius must also be given. After that $x = r \cos \phi$ and $y = r \sin \phi$. Also limited possibilities to alter the radius as a function of positions is given. The same mappings may be used as for 2D meshes in cartesian coordinates.

The examples shows a simple cylinder where a mapping is used to create two cones, as shown in Figure 3.7.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = 2D Polar
Subcell Divisions in 2D = 1 1
Subcell Limits 1 = 0 100
Subcell Limits 2 = 0 360
Material Structure in 2D
  1
End
Geometry Mappings
! mode  line  limits(2)   Np  params(Np)
  51    0     1e+06  1e+06   6   0 0 50 -25 100 0
End
Polar Radius = 30
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1        -1       1        1
  2        -3       1        1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 1000
Coordinate Ratios = 1
Element Ratios 1 = -0.25
Element Ratios 2 = 1
```
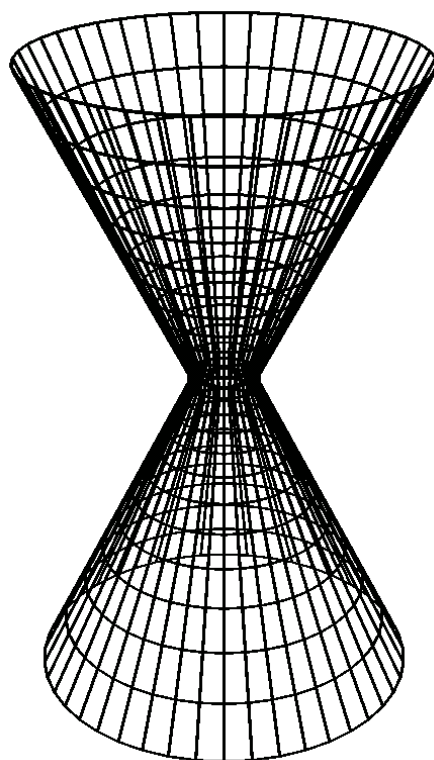
Figure 3.7: A 2D mesh mapped into two 3D cones

A rolled shell: `cones.grd`

The mappings out of plane may also be used to create a simple roll. Now the angles extends to 3600 degrees making 10 full rotations. The radius goes linearly from 1 to 101 and thus the roll has a spiraling structure, as shown in Figure 3.8.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = 2D Polar
Subcell Divisions in 2D = 1 1
Subcell Limits 1 = 0 100
Subcell Limits 2 = 0 3600
Material Structure in 2D
  1
End
Geometry Mappings
! mode  line  limits(2)  Np  params(Np)
  -51   0     1e+06  1e+06   4   0 0 3600 100
End
Polar Radius = 1
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1        -1       1        1
  2        -3       1        1
End
Numbering = Horizontal
```

```
Element Degree = 1
Element Innernodes = False
Triangles = False
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 1
Element Ratios 2 = 1
Element Divisions 1 = 10
Element Divisions 2 = 1000
```
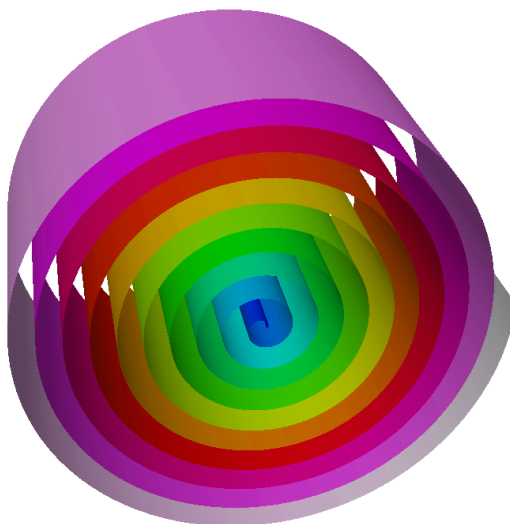

Figure 3.8: A rolled shell with a spiraling structure

A hexagonal frame: `hexframe.grd`

There is also a very special mapping for creating 3D shell structures. The mapping is used to give an angle for the shell. The angle is stepwise constant which makes is possible to create polygonal 3D shells, as shown in Figure 3.9.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 6 1
Subcell Limits 1 = 0 1 2 3 4 5 6
Subcell Limits 2 = 0 2
Material Structure in 2D
  1    1    1    1    1    1
End
Geometry Mappings
! mode  line  limits(2)  Np  params(Np)
  58    0     1e+06  1e+06   14   0 0 1 60 2 120 3 180 4 240 5 300 6 360
End
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1        -1       1        1
  2        -3       1        1
```

```
End
Numbering = Vertical
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 200
```
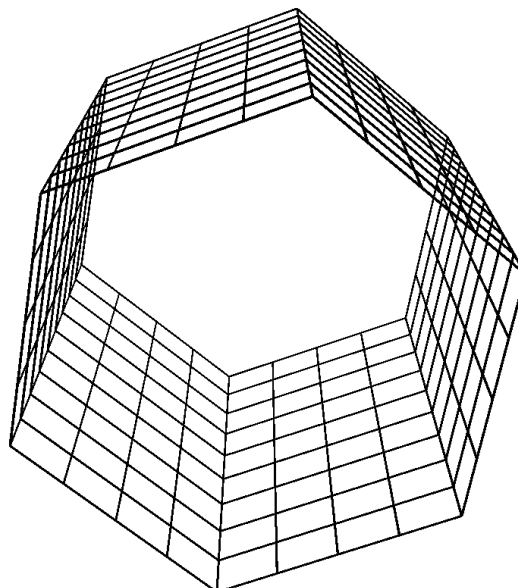

Figure 3.9: A hexagonal 3D frame made by mapping by angle

Rotated step structure: `barrel.grd`

Rotate a 2D step structure to create a 3D barrel shaped structure, as shown in Figure 3.10.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 3 3
Subcell Limits 1 = 0 1 2 3
Subcell Limits 2 = 0 1 2 3
Material Structure in 2D
  1    0    0
  1    1    0
  1    1    1
End
Revolve Blocks = 2
Revolve Radius = 2
Materials Interval = 1 1
Boundary Definitions
! type     out      int      double   of the boundaries
  1        0        1        1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
```

```
Triangles = False
Surface Elements = 100
Coordinate Ratios = 1
Minimum Element Divisions = 1 1
Element Ratios 1 = 1 1 1
Element Ratios 2 = 1 1 1
Element Densities 1 = 1 1 1
Element Densities 2 = 1 1 1
```
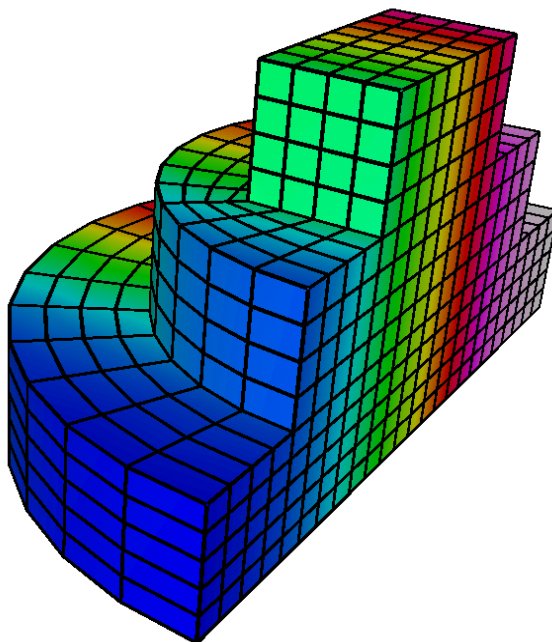

Figure 3.10: Step structure rotated 180 degrees to create 3D object with 1312 elements

Extruded structure: `blocks.grd`

The 2D structures may be extruded in the third dimension. Then the third dimension is added to the subcells boundaries and to the mesh definitions, as shown in Figure 3.11.

The material structure is always 2D. The given material structure may be extruded as is, or alternatively a selective extrusion for different materials may be performed as is done in this example. It is therefore possible to omit some materials, and to rename the extruded layer to some new material.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 3D
Subcell Divisions in 3D = 4 5 5
Subcell Limits 1 = 0 0.1 0.4 0.5 0.6
Subcell Limits 2 = 0 0.1 0.2 0.4 0.5 0.6
Subcell Limits 3 = 0 0.05 0.1 0.2 0.25 0.3
Material Structure in 2D
  1    1    1    1
  1    3    3    1
  1    2    3    1
  1    3    3    1
  1    1    1    1
```

```
End
Materials Interval = 1 3
Extruded Structure
! 1stmat    lastmat   newmat
  1         3         1
  1         1         2
  1         2         3
  1         1         4
  1         3         5
End
Boundary Definitions
! type      out       int       double   of the boundaries
  1         3         1         1
  2         2         1         1
  3         3         2         1
  4         0         1         1
End
Numbering = Horizontal
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 200
Coordinate Ratios = 1 1
Minimum Element Divisions = 2 2 2
```
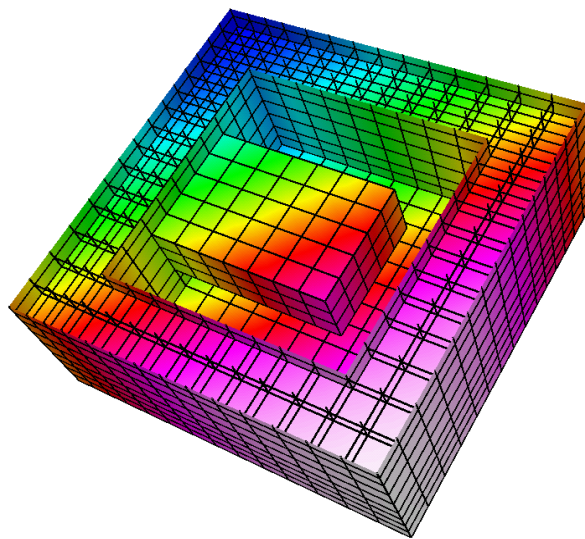

Figure 3.11: Simple 3D structure created by selective extrusion. The upper part has been removed for visualization purposes.

3.3 1D examples

The ElmerGrid program was originally written for 2D mesh generation. However, it is also possible to make even more trivial one dimensional grids. This is rarely needed as most FEM computations are at least two dimensions.

Line: `line.grd`

The example of the 1D case is naturally a line segment – there are no other kind of basic 1D structures. It consists of two sections, the first one has a increasing mesh density to the right and the second one has an even distribution of nodes, as shown in Figure 3.12.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 1D
Subcell Divisions in 1D = 3
Subcell Limits 1 = 0 1 3 4
Material Structure in 1D
  1    1    1
End
Materials Interval = 1 1
Boundary Definitions
! type      out      int
  1         -1       1         1
  2         -2       1         1
End
Element Degree = 1
Element Innernodes = False
Triangles = False
Surface Elements = 40
Element Ratios 1 = 2 1 0.5
Element Densities 1 = 1 0.5 1
```
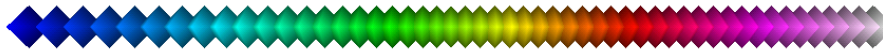

Figure 3.12: A line segment where the element nodes are marked with polygons. The mesh has 50 elements and thus 51 nodes.

Chapter 4

ElmerGrid mesh manipulation examples

ElmerGrid may often be used as a grid manipulation tool even though the computational mesh would originally have been made by some other mesh generator. There are many basic operations, such as scaling or rotation, which do not require any special attention. Here we look at some more advanced things that may be performed on a mesh.

4.1 Mesh partitioning

Nowadays the best performance from computers is obtained using parallel solution techniques. In finite element method this means that the mesh must be divided into separate domains that are solved by single processors. The domains should be chosen so that the extra work needed for the communications is minimized. This procedure is called mesh partitioning. ElmerGrid offers two different techniques for this. The first one divides the mesh using simple ordering schemes, and the second one uses the METIS library.

As an example we choose an L-shaped domain of bilinear elements. The ElmerGrid file for creating this example is given below.

```
***** ElmerGrid input file for structured grid generation *****
Version = 210903
Coordinate System = Cartesian 2D
Subcell Divisions in 2D = 2 2
Subcell Limits 1 = 0 1 2
Subcell Limits 2 = 0 1 2
Material Structure in 2D
  1    0
  1    1
End
Materials Interval = 1 1
Boundary Definitions
! type      out       int
  1         0         1         1
End
Element Degree = 1
Element Innernodes = False
Triangles = False
Plane Elements = 30000
```

The partitioning may be invoked using command line arguments, or as here using command file. Below is the command file for using the simple partitioning scheme to partition the mesh into 2×2 domains, as shown in Figure 4.1a.

```
Input File = angle
Output File = angle
Input Mode = ElmerGrid
Output Mode = ElmerPost
Partition = 2 2
```

In the start of the command file the user defines the input and output formats and filenames. The only additional command is the last line that defines how the mesh is partitioned. Another choice is to use METIS which is invoked with the line

```
Metis = 4
```

The resulting partitionings are shown in Figure 4.1. In this case the partitionings are almost equally good. In very simple cases the simple geometrical algorithm may give better results but in general cases METIS should be favored.
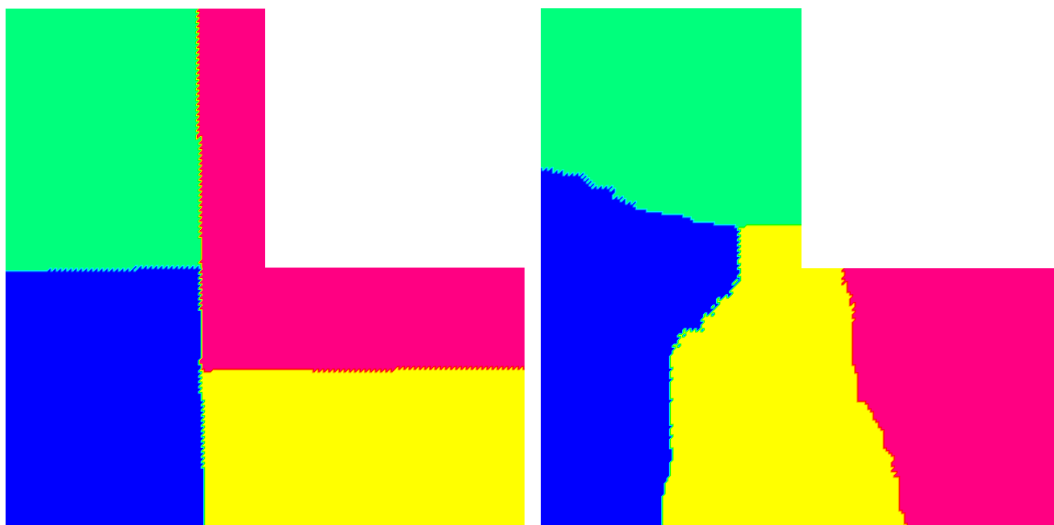
Figure 4.1: A simple mesh partitioned with a) a simple geometrical scheme and b) METIS library.

4.2   Boundary layer creation

In nature there are often sharp changes in the physical behavior which may be seen in forms of boundary layers. Particularly in the area of fluid dynamics boundary layers are often of special interest. A physical boundary layer requires a suitable computational mesh if one wants to model the physical phenomena accurately. Equally tight mesh is not usually needed elsewhere and thus an optimal solution is a proper boundary layer mesh. Ideally the mesh is such that the elements are small only in the direction of the gradient.

ElmerGrid includes a simple 2D capability for creating boundary layer meshes. There the user may choose the boundary conditions of interest and extrude them in the direction of the normal to create a structured mesh just on the boundary. The new mesh may also be fitted into the size of the original mesh using a simple Jacobian smoother which tries to maintain the geometrical shape of the mesh. Often this is not so easy and therefore the methodology may create some complications on the corner nodes, for example.

The example case chosen is the same one as for the partitioning example except the number of plane elements is reduced to 300 to make the boundary layer more visible. In the geometry there is only one boundary and therefore only one boundary layer is also created. There can however be several boundary layers at a time. In the intersections of different boundary layers one should however notice that the number of new elements should be the same. The resulting mesh is shown in Figure 4.2.

```
Input File = angle
Output File = layer
```

```
Input Mode = ElmerGrid
Output Mode = ElmerPost
Boundary Layer Move = 1000
Boundary Layer Epsilon = 1.0e-4
Boundary Layer
  1 5 0.2 5.0 0
END
```
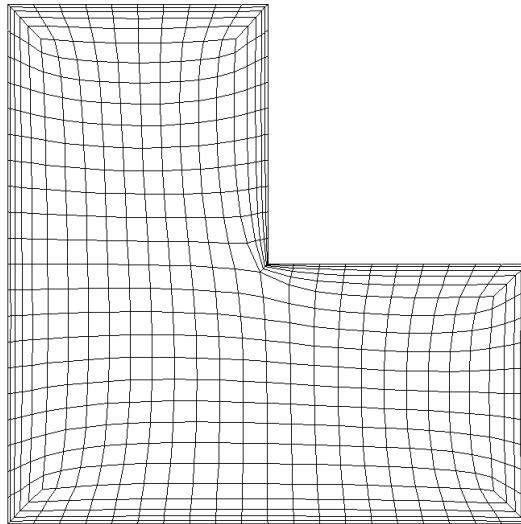

Figure 4.2: A simple boundary layer created on an angle mesh

4.3   Mesh extrusion

ElmerGrid enables that any mesh on a plane may be extruded in the third dimension by giving the specifications on how to extrude. For example the command file below may be used to extrude the mesh created in the first example in the third dimension. The result is a unit cube with 1000 trilinear elements.

```
Input File = rect
Output File = cube
Input Mode = ElmerSolver
Output Mode = ElmerSolver
Extruded Divisions = 1
Extruded Limits = 0 1
Extruded Ratios = 1.0
Extruded Elements = 10
```
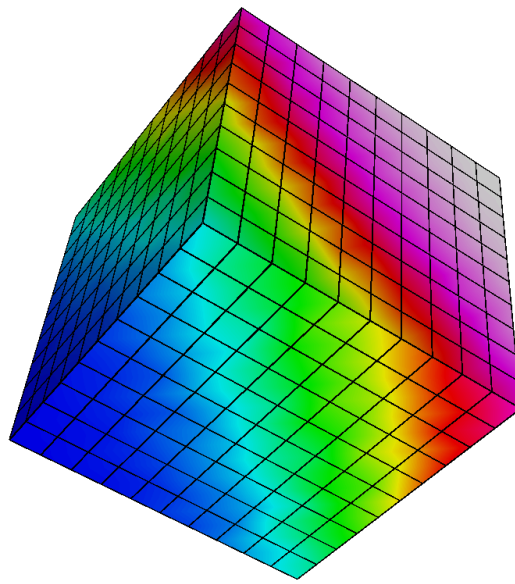
Figure 4.3: A simple cube created by the extrusion of a unit square

Chapter 5

ElmerGrid Interface Examples

ElmerGrid understands a number of different mesh formats which enables the import from meshes created by other mesh generators into the Elmer solution cycle. Supported formats includes those by Ansys (separate macro), Abaqus, Gambit (FDNET), Comsol Multiphysics, Fieldview, Triangle, Medit, Gsmh, and GiD.

5.1 Ansys mesh to Elmer

The APDL macros and the following documentation have been written by *Antti Pursula* (email: Antti.Pursula@csc.fi). He will answer any further questions related to the macros.

Two APDL (Ansys parametric design language) macros has been created which allow writing geometry data in ASCII format out of Ansys. The macros can be activated from Ansys GUI using custom added buttons in the Ansys Toolbar. The ASCII format files can further be converted in the Elmer format by using ElmerGrid.

The information written out by the macros consists of four files that are named `ExportMesh.*` (`*` = `header, node, elem`, or `boundary`). The files include all the information Elmer needs for using the mesh ie., besides the obvious node and element definitions, also information about the boundary nodes.

In Ansys, the geometry model is often created combining several elementary geometries. This means that the model usually includes boundaries, that have no physical meaning. These do not disturb the analysis neither in Ansys nor in Elmer, but they can make the definition of boundary conditions quite cumbersome in Elmer. Therefore, there are two possible macros for exporting data from Ansys: ELMER_AU and ELMER_CH. The first writes all the boundaries of the model automatically while the second asks the user to graphically pick the boundaries that have physical meaning. It is worthwhile to notice that Elmer does not necessarily need boundary information for all the physical boundaries, but only those on which a boundary condition is to be applied.

To use Ansys to Elmer interface you need to download two macro files and an Ansys start-up file and copy them into your working directory. After the files have been copied, start Ansys and the buttons ELMER_AU and ELMER_CH appear in the Toolbar. The conversion of the resulting ASCII files into Elmer format is done by the ElmerGrid program. The files as well as ElmerGrid can be downloaded from the Elmer download page.

In ElmerGrid the Ansys input format is 4 and output format can be Elmer mesh format (3) or ElmerPost format (2). So, creating Elmer meshes from Ansys input is done with the command: `ElmerGrid 4 3 ExportMesh`. There are also many command-line options available. The most important ones in this case are:

- `-o name` for saving output with the name specified.

- `-order` c_x c_y c_z for reordering nodes and elements using $c_1x + c_2y + c_3z$.

- `-scale` c_x c_y c_z for scaling the mesh using the constant to multiply the coordinate values.

- `-merge epsilon` for merging nodes that are at most distance epsilon apart.

The -merge option is important in a case where Ansys creates multiple nodes on a boundary of two elementary geometries. These nodes are often noticed only when Elmer is not able to use a mesh created from Ansys. In such a case, use -merge option with a suitable value for the node separation.
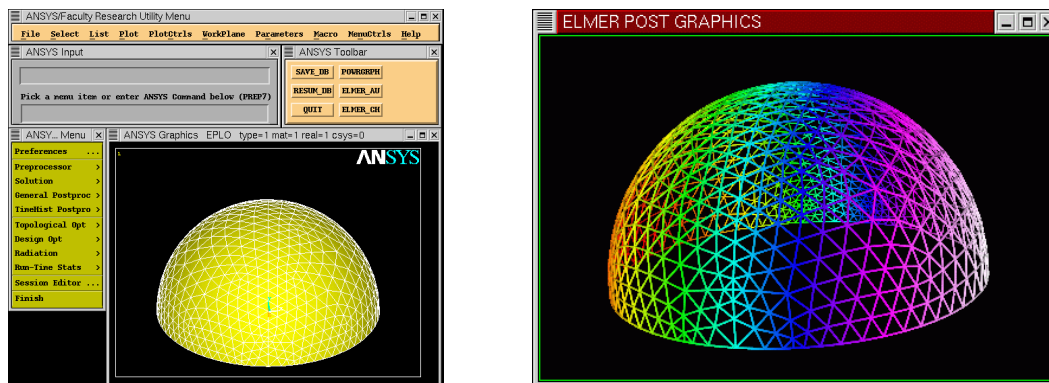

Figure 5.1: Pictures of the same mesh in a) Ansys desktop and in b) ElmerPost window.

5.2   Comsol Multiphysics to Elmer

Comsol Multiphysics is a widely used finite element tool with highly automated mesh generation. Previously the software was known as Femlab. In the older versions of Comsol Multiphysics/Femlab there wasn't any export format that would have included the boundary definitions. However, since version 3.2 of Comsol Multiphysics there is a suitable export format that may be directly used to export meshes in Elmer – the mphtxt format. The ElmerGrid program understands a sufficient subset of the commands in mphtxt-file of Comsol multiphysics.

The procedure for importing meshes to Elmer is

- In Comsol Multiphysics choose
  ```
  File -> Export -> Mesh to File
  ```

- Select the export format
  ```
  ''COMSOL Multiphysics text file'' (mphtxt)
  ```

- Give the mesh a file name, e.g. mymesh and you obtain the file mymesh.mphtxt

- Call then ElmerGrid with the following command line options to obtain E ElmerSolver mesh file format
  ```
  ElmerGrid 9 2 mymesh.mphtxt
  ```

- And with the following options to obtain ElmerPost result format
  ```
  ElmerGrid 9 3 mymesh.mphtxt
  ```
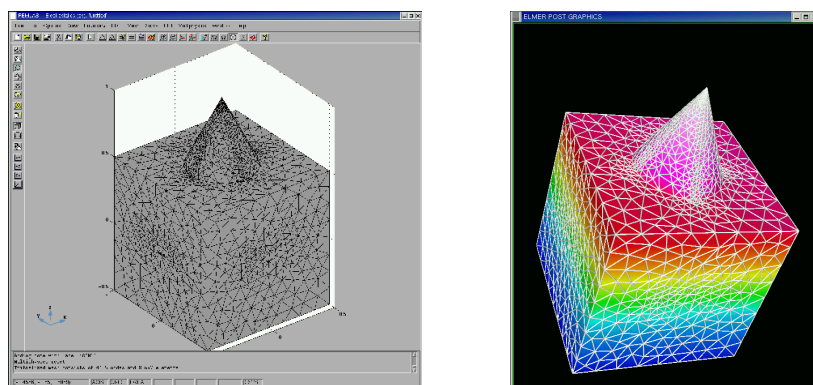
Figure 5.2: Pictures of the same mesh in a) Femlab desktop and in b) ElmerPost window.

5.3   Triangle mesh to Elmer

Triangle is a two-dimensional quality mesh generator and Delaunay triangulator. written by Jonathan Richard Shewchuk in University of California at Berkeley (email: jrs@cs.berkeley.edu).

The format written by Triangle is very similar to that one needed by Elmer. The only problem is to find the boundaries defined by the nodes known to lie on the boundary. The current implementation of the algorithm may result to problems at the corners. Also the functionality hasn't been tested for cases with several materials or several boundaries.

Assuming we have a triangle input file name `tri.poly`. This may be meshed with second order triangles having all corners above 32.0 degrees and area below 0.001 with the command

```
triangle -pq32.0 -a.0001 -o2 tri
```

This creates files `tri.i.ele` and `tri.i.node` (i=1, 2, . . .), which may be used to create the Elmer model.

The Triangle format corresponds to input mode 11 at ElmerGrid. Therefore the mesh for ElmerSolver is created with command

```
ElmerGrid 11 2 tri.1
```

which results to a computational mesh being saved at ElmerSolver format in subdirectory `tri`.
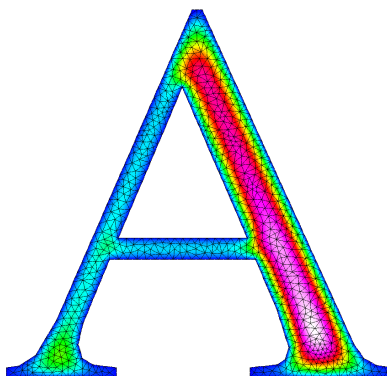
Figure 5.3: Picture of the Triangle example case in ElmerPost.

Index