

# ELP USER MANUAL

version 1.6  
July 2003

prepared by  
Jelle Muylle  
Peter Iványi

Heriot-Watt University  
Structural Engineering Computational Technology (SECT) Research Group  
Department of Mechanical and Chemical Engineering  
Edinburgh, U.K.

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
<b>2</b>	<b>ELP program interaction</b>	<b>7</b>
2.1	Overview . . . . .	7
2.2	ELP programs . . . . .	7
<b>3</b>	<b>E_Library file types</b>	<b>9</b>
3.1	The Mesh Definition File . . . . .	10
3.2	The Material File . . . . .	17
3.3	The Domain Decomposition File . . . . .	19
3.4	The Mesh Parameters File . . . . .	20
3.5	The Finite Element Error File . . . . .	22
3.6	The Finite Element Stress File . . . . .	23
3.7	The Element Geometric Definition File . . . . .	24
<b>4</b>	<b>Unstructured mesh generation: MGN</b>	<b>27</b>
4.1	Status . . . . .	27
4.2	Syntax . . . . .	27
4.3	Overview . . . . .	27
4.4	MGN batch file syntax . . . . .	28
4.4.1	Tasks . . . . .	28
4.4.2	Methods . . . . .	29
4.4.3	Delaunay options . . . . .	29
4.4.4	Control types . . . . .	29
4.4.5	Mesh options . . . . .	30
4.4.6	Filenames . . . . .	30
4.4.7	Smoothing . . . . .	31
4.4.8	Output format . . . . .	31
4.5	CTL local control sources syntax . . . . .	31

<b>5</b>	<b>Structured mesh generation: CUBIC</b>	<b>32</b>
5.1	Status . . . . .	32
5.2	Syntax . . . . .	32
5.3	File format . . . . .	32
5.4	Entity definitions . . . . .	32
5.5	Data file format . . . . .	33
5.5.1	Header section . . . . .	34
5.5.2	Point entity definition . . . . .	34
5.5.3	Surface entity definition . . . . .	35
5.5.4	Link entity definition . . . . .	37
5.5.5	Boundary condition definition section . . . . .	37
5.6	Examples for parametric definition . . . . .	39
<b>6</b>	<b>2D Finite element analysis: FEM</b>	<b>44</b>
6.1	Status . . . . .	44
6.2	Syntax . . . . .	44
6.3	Overview . . . . .	44
6.4	Program limitations . . . . .	45
<b>7</b>	<b>Error analysis: ADAPT</b>	<b>46</b>
7.1	Status . . . . .	46
7.2	Syntax . . . . .	46
7.3	Overview . . . . .	46
7.4	Program limitations . . . . .	46
<b>8</b>	<b>Viewing and printing: E_PLOT32</b>	<b>48</b>
8.1	Status . . . . .	48
8.2	Syntax . . . . .	48
8.3	Overview . . . . .	48
8.4	Help . . . . .	49
8.5	Program limitations . . . . .	49
8.6	Alternative platforms . . . . .	49
<b>9</b>	<b>Editing and modifying E_Library files: MDFTOOLS</b>	<b>50</b>
9.1	CHKMDF . . . . .	51
9.1.1	Status . . . . .	51
9.1.2	Syntax . . . . .	51
9.1.3	Overview . . . . .	51
9.2	CLEANUP . . . . .	52

9.2.1	Status . . . . .	52
9.2.2	Syntax . . . . .	52
9.2.3	Overview . . . . .	52
9.3	EMP2MPR . . . . .	53
9.3.1	Status . . . . .	53
9.3.2	Syntax . . . . .	53
9.3.3	Overview . . . . .	53
9.4	FLOOR . . . . .	54
9.4.1	Status . . . . .	54
9.4.2	Syntax . . . . .	54
9.4.3	Overview . . . . .	54
9.5	MAKEDOM . . . . .	55
9.5.1	Status . . . . .	55
9.5.2	Syntax . . . . .	55
9.5.3	Overview . . . . .	55
9.6	MAKEMAT . . . . .	56
9.6.1	Status . . . . .	56
9.6.2	Syntax . . . . .	56
9.6.3	Overview . . . . .	56
9.7	MAKEMPR . . . . .	57
9.7.1	Status . . . . .	57
9.7.2	Syntax . . . . .	57
9.7.3	Overview . . . . .	57
9.8	MDF2K . . . . .	58
9.8.1	Status . . . . .	58
9.8.2	Syntax . . . . .	58
9.8.3	Overview . . . . .	58
9.9	MDFMERGE . . . . .	59
9.9.1	Status . . . . .	59
9.9.2	Syntax . . . . .	59
9.9.3	Overview . . . . .	59
9.10	MPR2EMP . . . . .	60
9.10.1	Status . . . . .	60
9.10.2	Syntax . . . . .	60
9.10.3	Overview . . . . .	60
9.11	RENUMBER . . . . .	61
9.11.1	Status . . . . .	61

9.11.2	Syntax . . . . .	61
9.11.3	Overview . . . . .	61
9.11.4	Program limitations . . . . .	61
9.12	REV . . . . .	62
9.12.1	Status . . . . .	62
9.12.2	Syntax . . . . .	62
9.12.3	Overview . . . . .	62
9.13	SPHERE . . . . .	63
9.13.1	Status . . . . .	63
9.13.2	Syntax . . . . .	63
9.13.3	Overview . . . . .	63
9.14	VSPH . . . . .	64
9.14.1	Status . . . . .	64
9.14.2	Syntax . . . . .	64
9.14.3	Overview . . . . .	64
9.15	TRF . . . . .	65
9.15.1	Status . . . . .	65
9.15.2	Syntax . . . . .	65
9.15.3	Overview . . . . .	65
9.15.4	Syntax of transformation file . . . . .	65
9.15.5	Coordinate system . . . . .	66

# List of Figures

2.1	The ELP programs . . . . .	8
3.1	The element node orders . . . . .	10
3.2	An example with TRIANG1 elements . . . . .	11
3.3	An example with TRIANG2 elements . . . . .	12
3.4	An example domain decomposition with TRIANG1 elements . . . . .	19
3.5	An example element mesh parameter definition . . . . .	20
3.6	An example nodal mesh parameter definition . . . . .	21
3.7	An example for geometric definition . . . . .	26
5.1	Conventions for the entity definition, (a) uniform quadrilateral, (b) nonuniform quadrilateral and (c) triangular entities . . . . .	33
5.2	Surface definition (a) without and (b) with “referencing points” . . . . .	36
5.3	Referenced node in the first surface entity . . . . .	36
5.4	LINK_DIRECTION values and the corresponding direction of the one dimensional elements, arrows also show the order of the definition of the elements according to LINK_DIVISION . . . . .	37
5.5	Boundary condition generation defined by two reference points for an area, the same boundary conditions will be generated for the points inside and on the edge of the shaded area . . . . .	39
5.6	The real domain $\Omega$ (a) and the unit square domain $\overline{\Omega}$ (b) for the complex example . . . . .	40
9.1	Result of <code>rev test 10 -o 90 -y</code> . . . . .	62
9.2	Coordinate system . . . . .	66

# Chapter 1

## Introduction

This document contains the user manuals for the different tools bundled in the ELP. ELP stands for E\_Library Package. It is a series of general purpose finite element analysis and design tools developed by the Structural Engineering Computational Technology (SECT) research group at Heriot-Watt University Edinburgh. All the tools in ELP have in common that they were created to the same standard, the E\_Library.

The purpose of this manual is to explain the usage and behaviour of the different components of ELP. It is not the aim of this manual to explain the code or supply developer information.

## Chapter 2

# ELP program interaction

### 2.1 Overview

Figure 2.1 shows the different programs which are currently in the ELP. It also shows what kind of files every program expects as input and which ones it generates. More details about the requirements of each program are explained in the appropriate chapters for each program.

The ELP actually splits up into three major components.

- programs for 2D structures and meshes, where the focus is set on efficient and adaptive high quality multi purpose mesh generation.
- programs for 3D meshes, mainly concerned with design and analysis of membrane structures using dynamic relaxation.
- the programs that are common to both approaches and that are merely tools to inspect, correct, view and print meshes at any time in the interactive process.

### 2.2 ELP programs

The following programs are described in the chapters of this manual:

**MGN:** a multi purpose unstructured 2D mesh generator with a variety of ways to control mesh density at any place in the domain using different mesh generation algorithms. (Chapter 4)

**CUBIC:** a structured mesh generator, which is mainly used as a pre-processor for DR because of its capacity to handle typical types of membrane features, such as cables, g-strings and clothes. (Chapter 5)

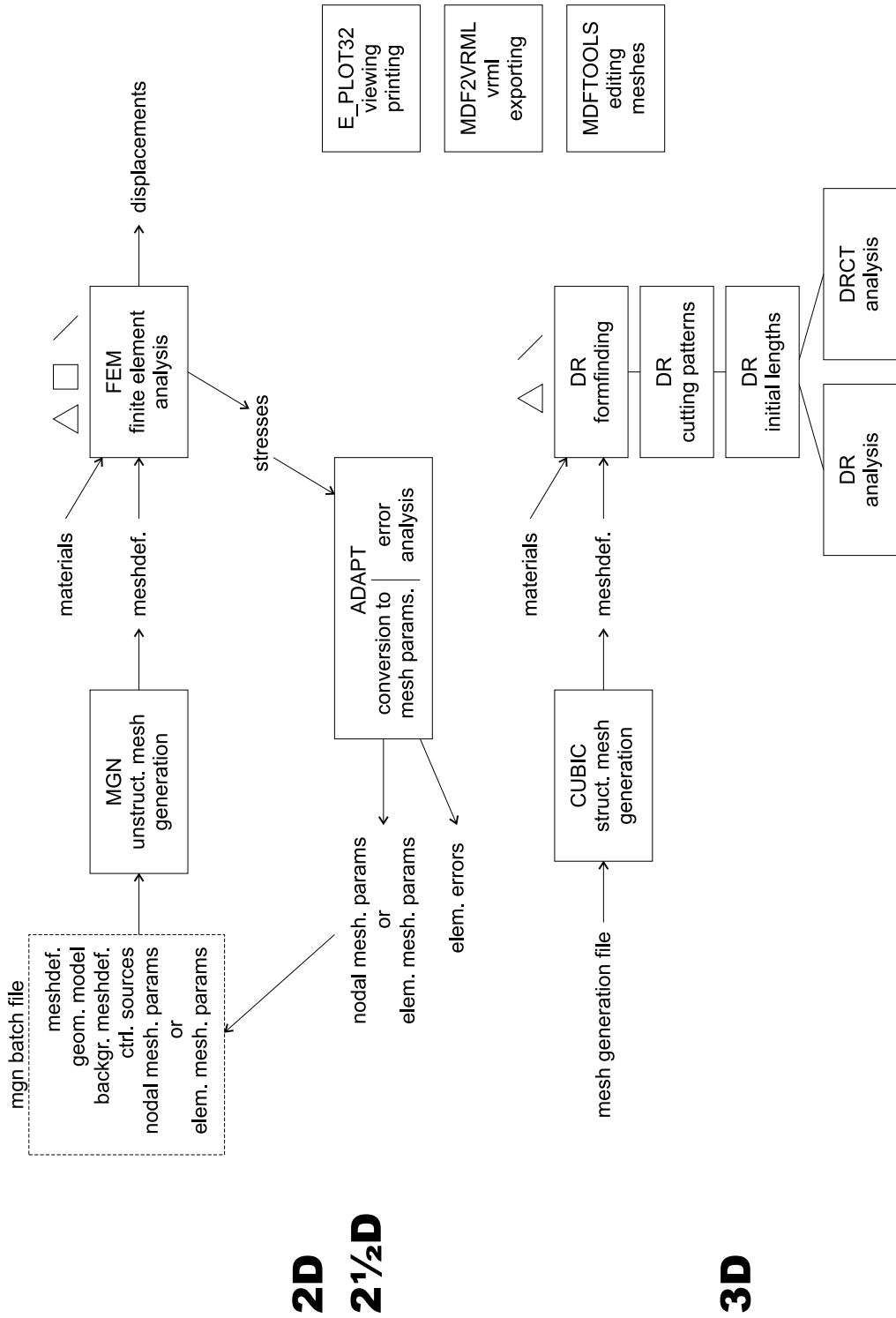
**FEM:** a very basic, robust finite element analysis program for 2D meshes. (Chapter 6)

**ADAPT:** generates finite element errors and mesh parameters for an adaptive remeshing of a 2D mesh. (Chapter 7)

**E\_PLOT32:** viewing and printing of meshes, subdomains and mesh parameters in an easy to use windows interface. (Chapter 8).

**MDFTOOLS:** a series of basic editing tools to check and modify mesh definition files (Chapter 9).





# E-LIBRARY PROGRAMS

Figure 2.1: The ELP programs

## Chapter 3

# E\_Library file types

The E-Lib format supports meshes of a single element type and mixed meshes. The E-Lib mesh file format takes the form of a series of keywords followed by one or more data items. For example,

```
NELEMENTS_TRIANG1    120
```

denotes that the mesh contains 120 elements of type **TRIANG1**.

**IMPORTANT** Most keywords are optional, but those which are declared must be in a strict, predefined order. This order automatically accounts for dependencies between the data.

The format also allows the use of file comments. If the first non-white character on a line is the `#` character, then all remaining text on that line is ignored. Two or more lines can be commented out by enclosing the lines inside a `/*` and `*/`. The opening `/*` must be the first two characters on the first comment line and `*/` the first two characters on the last comment line. All text on the line following `/*` and `*/` is ignored. The `/*...*/` comments must not be nested, although they can contain `#` comments. Blank lines are also permitted.

There are currently seven types of data file – the mesh definition (**.mdf**), geometric definition (NURBS curve) (**.gmf**), materials (**.mat**), domain decomposition (**.dom**), element or nodal mesh parameters (**.mpr**), stresses (**.ste**) and finite element errors files (**.fee**). The mesh definition file contains the main description of the mesh. The other files describe additional properties of the mesh or they represent a state of the mesh.

The following sections describe the keywords and keyword data for the above data files. Each section contains a table which lists the keywords **in the order** they must be declared and defines the associated keyword data. The data type – whether it is an integer (i), real (r) or a character string (s) – is also given, where, for example, ‘i 3\*r’ denotes that the data consist of an integer followed by three reals. Units, where relevant, are enclosed in square brackets [...].

**IMPORTANT** Keyword data consisting of a single data item must follow the keyword on the same line. For vectors and matrices, each vector component or matrix row must start on a new line and each line must begin with an integer index. Unless otherwise stated, this index is either the vector component number or matrix row index. All keyword data strings must be enclosed in double quotes.

### 3.1 The Mesh Definition File

The mesh definition file (**.mdf**) contains the main description of the mesh.

At present twelve different elements types are supported – five one dimensional (**LINK1**, **LINK2**, **LINK3**, **LINK4**, **LINK5**), five triangular (**TRIANG1**, **TRIANG2**, **TRIANG3**, **TRIANG4**, **TRIANG5**), two quadrilateral (**QUAD1** and **QUAD3**), one tetrahedral (**TETRAH1**) and one three dimensional block element (**BLOCK1**). These element types are defined in Figure 3.1 which shows the order in which the element nodes must be labelled and in Table 3.1 which gives a short description as well. With the exception of the **TRIANG2** and **QUAD3** elements, individual elements have uniform material properties. **TRIANG2** and **QUAD3** elements are composed of layered composite materials referred to as composite materials of type 1.

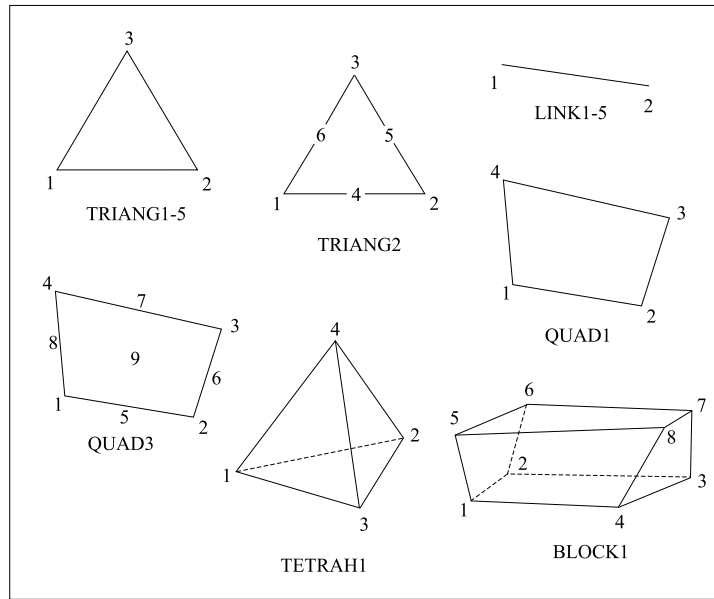


Figure 3.1: The element node orders

An example mesh (Figure 3.2) definition file is shown in the following

```
# An example mesh definition file

TITLE "An example mesh"
NMESHPPOINTS      6
NNODES            6
NELEMENTS_TRIANG1 4

MESHPOINT_COORDINATES
1  0.000  0.000  0.000
2  3.000  0.000  0.000
3  6.000  0.000  0.000
4  0.000  3.000  0.000
5  3.000  3.000  0.000
6  6.000  3.000  0.000

NODES_TRIANG1
1  1  2  4
2  2  5  4
3  2  3  5
4  3  6  5
```

Element Type	Description	Number of Vertices	Number of Nodes
<b>LINK1</b>	Truss	2	2
<b>LINK2</b>	Cable	2	2
<b>LINK3</b>	Fixed tension 1-D link	2	2
<b>LINK4</b>	Fixed force density 1-D link	2	2
<b>LINK5</b>	Geodesic string	2	2
<b>TRIANG1</b>	Constant strain triangular	3	3
<b>TRIANG2</b>	A combined constant strain plane stress and constant moment plate bending simple facet triangular	3	6
<b>TRIANG3</b>	Solid triangular	3	3
<b>TRIANG4</b>	Membrane triangular	3	3
<b>TRIANG5</b>	Constant stress triangular	3	3
<b>QUAD1</b>	Plane stress quadrilateral	4	4
<b>QUAD3</b>	Mindlin plate quadrilateral	4	9
<b>TETRAH1</b>	Tetrahedral	4	4
<b>BLOCK1</b>	Block	8	8

Table 3.1: The element types

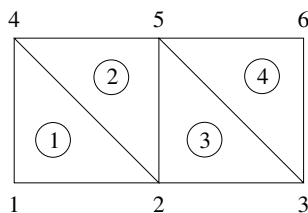


Figure 3.2: An example with TRIANG1 elements

The first keyword **TITLE**, is compulsory and specifies the title of the mesh. The title can be used to annotate output such as the display in a plot program and, as with all string arguments, must be enclosed in double quotes.

A mesh is defined in terms of a set of *mesh-points* which are joined by straight lines to form the edges or surfaces of the elements.

**IMPORTANT** A distinction is made between the terms *mesh-point*, *vertex* and *finite element node*. A mesh-point is a point in space which may or may not form an element vertex, whereas a finite element node (or simply *node*) is a point on an element used for function interpolation during the finite element analysis. Unlike mesh-points, nodes need not coincide with the element vertices. (See Figure 3.3.)

**NMESHPOINTS** is the keyword for the number of mesh-points defined in the mesh definition file and **NNODES** defines the total number of finite element nodes in the mesh. The coordinates of the mesh-points follow the keyword **MESHPOINT\_COORDS**. All three of the keywords **NMESHPOINTS**, **NNODES** and **MESHPOINT\_COORDS** are compulsory. The keyword **NELEMENTS\_TRIANG1** act both to declare the type of elements in the mesh and to give the number of the element. If this number is zero then the keyword should be omitted. At least one of the keywords which defines the

element type must be declared and for each such keyword present there must also be the corresponding keyword like **NODES\_TRIANG1** etc. The data for these keywords define the node indices of each element in turn in the order given in Figure 3.1. Element vertex nodes are always labelled before non-vertex nodes and for two dimensional elements, labelling occurs in an anti-clockwise direction.

Another example is presented to demonstrate the difference between *mesh-point*, *vertex* and *finite element node*. The geometric arrangement of the elements can be seen in Figure 3.3 while the generated mesh file is the following:

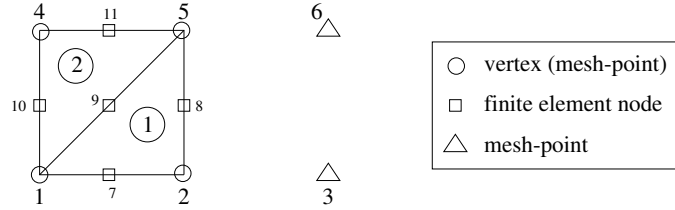


Figure 3.3: An example with TRIANG2 elements

```
# Example with TRIANG 2 elements

TITLE "An example mesh"
NMESHPOINTS      6
NNODES           10
NELEMENTS_TRIANG2  2

MESHPOINT_COORDINATES
1  0.000  0.000  0.000
2  3.000  0.000  0.000
3  6.000  0.000  0.000
4  0.000  3.000  0.000
5  3.000  3.000  0.000
6  6.000  3.000  0.000

NODES_TRIANG2
1  1  2  5  7  8  9
2  1  5  4  9  11  10
```

**NOTE** *All other keywords are optional* except in cases where the declaration of one keyword implies that one or more further keywords will be declared later. For example, if the number of boundary condition nodes is defined (**NBOUNDARY\_CONDITION\_NODES**) then so must the nodal boundary conditions (**BOUNDARY\_CONDITIONS**), see example below.

```

# Example with extra keywords

TITLE "An example mesh"
NMESHPOINTS          3
NNODES               3
NELEMENTS_TRIANG1    1
NBOUNDARY_CONDITION_NODES  2

MESHPOINT_COORDINATES
  1    0.000    0.000    0.000
  2    6.000    0.000    0.000
  3    6.000    6.000    0.000

NODES_TRIANG1
  1      1      2      3

BOUNDARY_CONDITIONS
  1  "FIXED" 0.000  "FIXED" 0.000  "FREE"
  2  "FREE"      "FIXED" 0.000  "FREE"

```

An exhaustive list of possible keywords in an **.mdf** file is presented in Table 3.2a,b,c .  
 For the use of keywords not discussed here see the relevant section in the documentation.

Keyword	Keyword Data	Data Type
<b>TITLE</b>	Mesh title	s
<b>NMESHPOINTS</b>	Number of mesh-points	i
<b>NNODES</b>	Number of nodes	i
<b>NELEMENTS_LINK1</b>	Number of <b>LINK1</b> elements	i
<b>NELEMENTS_LINK2</b>	Number of <b>LINK2</b> elements	i
<b>NELEMENTS_LINK3</b>	Number of <b>LINK3</b> elements	i
<b>NELEMENTS_LINK4</b>	Number of <b>LINK4</b> elements	i
<b>NELEMENTS_LINK5</b>	Number of <b>LINK5</b> elements	i
<b>NELEMENTS_TRIANG1</b>	Number of <b>TRIANG1</b> elements	i
<b>NELEMENTS_TRIANG2</b>	Number of <b>TRIANG2</b> elements	i
<b>NELEMENTS_TRIANG3</b>	Number of <b>TRIANG3</b> elements	i
<b>NELEMENTS_TRIANG4</b>	Number of <b>TRIANG4</b> elements	i
<b>NELEMENTS_TRIANG5</b>	Number of <b>TRIANG5</b> elements	i
<b>NELEMENTS_QUAD1</b>	Number of <b>QUAD1</b> elements	i
<b>NELEMENTS_QUAD3</b>	Number of <b>QUAD3</b> elements	i
<b>NELEMENTS_TETRAH1</b>	Number of <b>TETRAH1</b> elements	i
<b>NELEMENTS_BLOCK1</b>	Number of <b>BLOCK1</b> elements	i
<b>NBOUNDARY_CONDITION_NODES</b>	Number of boundary condition nodes	i
<b>NLOADED_NODES</b>	Number of loaded nodes	i
<b>NMATERIALS</b>	Total number of materials including those declared in composite materials	i
<b>NCOMP_MATERIALS_TYPE1</b>	Number of type 1 composite materials	i
<b>NNURBS_CURVES</b>	Number of NURBS curves	i
<b>TIMESTEP</b>	The value of the timestep	r
<b>NTIMESTEPS</b>	Number of time-steps	i
	This keyword and the pair of keywords <b>NINTERNAL_TIMESTEPS</b> and <b>NEXTERNAL_TIMESTEPS</b> are mutually exclusive	
<b>NINTERNAL_TIMESTEPS</b>	Number internal and external time-steps	i
<b>NEXTERNAL_TIMESTEPS</b>	These data enable time-stepping to be broken down into a series of <i>NExternalTimesteps</i> sets of <i>NinternalTimesteps</i> time-steps (See <b>NTIMESTEPS</b> )	
<b>DAMPING_FACTOR</b>	Viscous damping factor	r
<b>BETA</b>	Newmark's $\beta$ integration constant	r
<b>GAMMA</b>	Newmark's $\gamma$ integration constant	r

Table 3.2: (a) The mesh definition file keywords and keyword data (cont.)

Keyword	Keyword Data	Data Type
<b>NSTRESS_POINTS_LINK1</b>	Number of stress points in a <b>LINK1</b> element	i
<b>NSTRESS_POINTS_LINK2</b>	Number of stress points in a <b>LINK2</b> element	i
<b>NSTRESS_POINTS_LINK3</b>	Number of stress points in a <b>LINK3</b> element	i
<b>NSTRESS_POINTS_LINK4</b>	Number of stress points in a <b>LINK4</b> element	i
<b>NSTRESS_POINTS_LINK5</b>	Number of stress points in a <b>LINK5</b> element	i
<b>NSTRESS_POINTS_TRIANG1</b>	Number of stress points in a <b>TRIANG1</b> element	i
<b>NSTRESS_POINTS_TRIANG2</b>	Number of stress points in a <b>TRIANG2</b> element	i
<b>NSTRESS_POINTS_TRIANG3</b>	Number of stress points in a <b>TRIANG3</b> element	i
<b>NSTRESS_POINTS_TRIANG4</b>	Number of stress points in a <b>TRIANG4</b> element	i
<b>NSTRESS_POINTS_TRIANG5</b>	Number of stress points in a <b>TRIANG5</b> element	i
<b>NSTRESS_POINTS_QUAD1</b>	Number of stress points in a <b>QUAD1</b> element	i
<b>NSTRESS_POINTS_QUAD3</b>	Number of stress points in a <b>QUAD3</b> element	i
<b>NSTRESS_POINTS_TETRAH1</b>	Number of stress points in a <b>TETRAH1</b> element	i
<b>NSTRESS_POINTS_BLOCK1</b>	Number of stress points in a <b>BLOCK1</b> element	i
<b>MESHPPOINT_COORDS</b>	x-,y- and z-coordinates of the mesh-point	i 3*r
<b>NODES_LINK1</b>	Node indices of each <b>LINK1</b> element	i 2*i
<b>NODES_LINK2</b>	Node indices of each <b>LINK2</b> element	i 2*i
<b>NODES_LINK3</b>	Node indices of each <b>LINK3</b> element	i 2*i
<b>NODES_LINK4</b>	Node indices of each <b>LINK4</b> element	i 2*i
<b>NODES_LINK5</b>	Node indices of each <b>LINK5</b> element	i 2*i
<b>NODES_TRIANG1</b>	Node indices of each <b>TRIANG1</b> element	i 3*i
<b>NODES_TRIANG2</b>	Node indices of each <b>TRIANG2</b> element	i 6*i
<b>NODES_TRIANG3</b>	Node indices of each <b>TRIANG3</b> element	i 3*i
<b>NODES_TRIANG4</b>	Node indices of each <b>TRIANG4</b> element	i 3*i
<b>NODES_TRIANG5</b>	Node indices of each <b>TRIANG5</b> element	i 3*i
<b>NODES_QUAD1</b>	Node indices of each <b>QUAD1</b> element	i 4*i
<b>NODES_QUAD3</b>	Node indices of each <b>QUAD3</b> element	i 9*i
<b>NODES_TETRAH1</b>	Node indices of each <b>TETRAH1</b> element	i 4*i
<b>NODES_BLOCK1</b>	Node indices of each <b>BLOCK1</b> element	i 8*i
<b>BOUNDARY_CONDITIONS</b>	Nodal boundary conditions  For each boundary condition node, the node index followed by, for each degree of freedom, the string " <b>FREE</b> " if no boundary conditions are imposed, " <b>FIXED</b> " followed by a real displacement [m], or " <b>SPRING</b> " followed by a non-negative spring-constant [Nm <sup>-1</sup> ]. The displacement argument specifies an initial displacement of a node. The spring constant enables elastic forces to be modelled.	←

Table 3.2: (b) The mesh definition file keywords and keyword data (cont.)



Keyword	Keyword Data	Data Type
<b>LOADS</b>	Nodal loads For each loaded node, the node index followed by the applied load [N] for each degree of freedom	i r
<b>MATERIALS.LINK1</b>	Material name of each <b>LINK1</b> element	i s
<b>MATERIALS.LINK2</b>	Material name of each <b>LINK2</b> element	i s
<b>MATERIALS.LINK3</b>	Material name of each <b>LINK3</b> element	i s
<b>MATERIALS.LINK4</b>	Material name of each <b>LINK4</b> element	i s
<b>MATERIALS.LINK5</b>	Material name of each <b>LINK5</b> element	i s
<b>MATERIALS.TRIANG1</b>	Material name of each <b>TRIANG1</b> element	i s
<b>COMP_MATERIALS.TRIANG2</b>	Composite material name of each <b>TRIANG2</b> element	i s
<b>MATERIALS.TRIANG3</b>	Material name of each <b>TRIANG3</b> element	i s
<b>MATERIALS.TRIANG4</b>	Material name of each <b>TRIANG4</b> element	i s
<b>MATERIALS.TRIANG5</b>	Material name of each <b>TRIANG5</b> element	i s
<b>MATERIALS.QUAD1</b>	Material name of each <b>QUAD1</b> element	i s
<b>COMP_MATERIALS.QUAD3</b>	Composite material name of each <b>QUAD3</b> element	i s
<b>MATERIALS.TETRAH1</b>	Material name of each <b>TETRAH1</b> element	i s
<b>MATERIALS.BLOCK1</b>	Material name of each <b>BLOCK1</b> element	i s
<b>REMESH_DATA</b>	Defines the beginning of NURBS definition	

Table 3.2: (c) The mesh definition file keywords and keyword data

## 3.2 The Material File

The materials file (**.mat**) contains the properties of the material and composite material declared in the mesh definition file. Any number of materials can be defined and not just those used by the current mesh. This enables a number of different meshes to use a single materials file. The materials and composite materials can be defined in any order.

Each definition of material properties begins with the keyword **MATERIAL** and ends with the keyword **END**. Type 1 composite material properties begin with the keyword **COMP\_MATERIAL\_TYPE1** and must also end with the keyword **END**. Tables 3.3 and 3.4 define the property keywords.

Keyword	Keyword Data	Data Type
<b>MATERIAL</b>	Material name	s
<b>MAT_TYPE</b>	Material model index	i
<b>DENSITY</b>	Density	r
<b>YMOD</b>	Isotropic Young's modulus This keyword and the x-,y- and z-direction Young's moduli keywords are mutually exclusive	r
<b>YMOD_X</b>	x-direction Young's modulus	r
<b>YMOD_Y</b>	y-direction Young's modulus	r
<b>YMOD_Z</b>	z-direction Young's modulus	r
<b>THICKNESS</b>	Thickness	r
<b>POISSONS_RATIO</b>	Poisson ratio	r
<b>IPARAM<math>n</math></b>	$n$ -th integer property, $n = 1, \dots, 6$	i
<b>DPARAM<math>n</math></b>	$n$ -th double precision property, $n = 1, \dots, 20$	r
<b>END</b>		

Table 3.3: The material property keywords and keyword data

Keyword	Keyword Data	Data Type
<b>COMP_MATERIAL_TYPE1</b>	Type 1 composite material name	s
<b>NLAYERS</b>	Number of layers	i
<b>LAYER_MATERIALS</b>	Material name of each layer	i s
<b>LAYER_THICKNESSES</b>	Thickness of each layer	i r
<b>END</b>		

Table 3.4: The type 1 composite material property keywords and keyword data

The structure offers a possibility to introduce new material properties which are not listed among the keywords. For integer values **IPARAM $n$**  and for real values **DPARAM $n$**  should be used.

**IMPORTANT** When new material property is defined, do not forget to document that which parameters correspond to which material property.

In the case of the composite material the thicknesses can have any value and their sum is not checked.

An example **.mdf** with materials would be the following.

```

# Example with TRIANG 1 elements and materials

TITLE "An example mesh"
NMESHPOINTS      6
NNODES           6
NELEMENTS_TRIANG1 4
NMATERIALS        2

MESHPOINT_COORDINATES
  1    0.000    0.000    0.000
  2    3.000    0.000    0.000
  3    6.000    0.000    0.000
  4    0.000    3.000    0.000
  5    3.000    3.000    0.000
  6    6.000    3.000    0.000

NODES_TRIANG1
  1      1      2      4
  2      2      5      4
  3      2      3      5
  4      3      6      5

MATERIALS_TRIANG1
  1      "steel"
  2      "concrete"
  3      "concrete"
  4      "steel"

```

While the corresponding **.mat** file looks like this.

```

MATERIAL "steel"
DENSITY 2000.0
YMOD    210000.0
# Yielding stress
DPARAM1 20
END

MATERIAL "concrete"
DENSITY 1000.0
YMOD    50000.0
# Yielding stress
DPARAM1 10
END

```

### 3.3 The Domain Decomposition File

The domain decomposition file (**.dom**) specifies the subdomains into which the elements have been partitioned (Table 3.5). The keywords **NSUBDOMAINS** and **SUBDOMAINS\_TRIANG1** etc. specify the number of subdomains to be created and the partition indices for the various element types. The indices must lie between 1 and the number of subdomains inclusive and the number of each element type must be consistent with the number of that type defined in the mesh definition file.

Keyword	Keyword Data	Data Type
<b>NSUBDOMAINS</b>	Number of subdomains	i
<b>SUBDOMAINS_LINK1</b>	Subdomain index of each <b>LINK1</b> element	i i
<b>SUBDOMAINS_LINK2</b>	Subdomain index of each <b>LINK2</b> element	i i
<b>SUBDOMAINS_LINK3</b>	Subdomain index of each <b>LINK3</b> element	i i
<b>SUBDOMAINS_LINK4</b>	Subdomain index of each <b>LINK4</b> element	i i
<b>SUBDOMAINS_LINK5</b>	Subdomain index of each <b>LINK5</b> element	i i
<b>SUBDOMAINS_TRIANG1</b>	Subdomain index of each <b>TRIANG1</b> element	i i
<b>SUBDOMAINS_TRIANG2</b>	Subdomain index of each <b>TRIANG2</b> element	i i
<b>SUBDOMAINS_TRIANG3</b>	Subdomain index of each <b>TRIANG3</b> element	i i
<b>SUBDOMAINS_TRIANG4</b>	Subdomain index of each <b>TRIANG4</b> element	i i
<b>SUBDOMAINS_TRIANG5</b>	Subdomain index of each <b>TRIANG5</b> element	i i
<b>SUBDOMAINS_QUAD1</b>	Subdomain index of each <b>QUAD1</b> element	i i
<b>SUBDOMAINS_QUAD3</b>	Subdomain index of each <b>QUAD3</b> element	i i
<b>SUBDOMAINS_TETRAH1</b>	Subdomain index of each <b>TETRAH1</b> element	i i
<b>SUBDOMAINS_BLOCK1</b>	Subdomain index of each <b>BLOCK1</b> element	i i

Table 3.5: The domain decomposition file keywords and keyword data

An example domain decomposition file is shown for Figure 3.2

```
# An example domain decomposition file
NSUBDOMAINS 2

SUBDOMAINS_TRIANG1
1 1
2 1
3 2
4 2
```

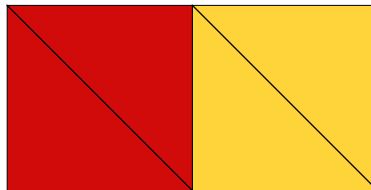


Figure 3.4: An example domain decomposition with TRIANG1 elements

### 3.4 The Mesh Parameters File

The mesh parameters file (**.mpr**) defines the element or nodal mesh parameters of each element or node. The number of element mesh parameters for each element type must equal the number of elements defined in the mesh definition file. The keywords for the different element types is shown in Table 3.6.

When the mesh parameters are defined on a nodal basis the **NODAL\_MESHPARAMS** keyword should be specified and after that the mesh parameter for each mesh-point should also be defined.

Keyword	Keyword Data	Data Type
<b>NODAL_MESHPARAMS</b>	Mesh parameters for each node	i r
<b>MESHPARAMS_LINK1</b>	Element mesh parameter of each <b>LINK1</b> element	i r
<b>MESHPARAMS_LINK2</b>	Element mesh parameter of each <b>LINK2</b> element	i r
<b>MESHPARAMS_LINK3</b>	Element mesh parameter of each <b>LINK3</b> element	i r
<b>MESHPARAMS_LINK4</b>	Element mesh parameter of each <b>LINK4</b> element	i r
<b>MESHPARAMS_LINK5</b>	Element mesh parameter of each <b>LINK5</b> element	i r
<b>MESHPARAMS_TRIANG1</b>	Element mesh parameter of each <b>TRIANG1</b> element	i r
<b>MESHPARAMS_TRIANG2</b>	Element mesh parameter of each <b>TRIANG2</b> element	i r
<b>MESHPARAMS_TRIANG3</b>	Element mesh parameter of each <b>TRIANG3</b> element	i r
<b>MESHPARAMS_TRIANG4</b>	Element mesh parameter of each <b>TRIANG4</b> element	i r
<b>MESHPARAMS_TRIANG5</b>	Element mesh parameter of each <b>TRIANG5</b> element	i r
<b>MESHPARAMS_QUAD1</b>	Element mesh parameter of each <b>QUAD1</b> element	i r
<b>MESHPARAMS_QUAD3</b>	Element mesh parameter of each <b>QUAD3</b> element	i r
<b>MESHPARAMS_TETRAH1</b>	Element mesh parameter of each <b>TETRAH1</b> element	i r
<b>MESHPARAMS_BLOCK1</b>	Element mesh parameter of each <b>BLOCK1</b> element	i r

Table 3.6: The element mesh parameters file keywords and keyword data

An example element mesh parameters file is shown below and in Figure 3.5, which corresponds to the mesh in Figure 3.2.

```
# An example element mesh parameters file

MESHPARAMS_TRIANG1
1 0.1
2 0.2
3 0.2
4 0.3
```

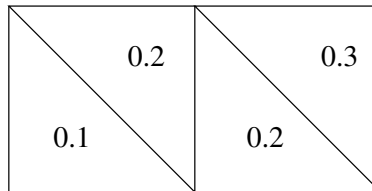


Figure 3.5: An example element mesh parameter definition

An example nodal mesh parameters file is shown below and in in Figure 3.6, which corresponds to the mesh in Figure 3.2. It should be noted that Figure 3.6 is **NOT** equivalent to Figure 3.5. However it is possible to convert nodal mesh parameters to element mesh parameters and vica versa by the tools **MPR2EMP** and **EMP2MPR**, see Sections 9.10 and 9.3.

```
# An example nodal mesh parameters file

NODAL_MESHPARAMS
1 0.1
2 0.2
3 0.3
4 0.1
5 0.2
6 0.3
```

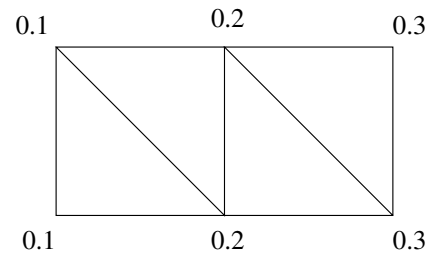


Figure 3.6: An example nodal mesh parameter definition

### 3.5 The Finite Element Error File

The finite element errors are stored in the finite element error file (**.fee**). One error value must be defined for each element (Table 3.7). Even in the case of **TRIANG2** and **QUAD3** elements which are layered finite elements only one value should be defined for one element.

Keyword	Keyword Data	Data Type
<b>FEERRORS_LINK1</b>	Finite element error of each <b>LINK1</b> element	i r
<b>FEERRORS_LINK2</b>	Finite element error of each <b>LINK2</b> element	i r
<b>FEERRORS_LINK3</b>	Finite element error of each <b>LINK3</b> element	i r
<b>FEERRORS_LINK4</b>	Finite element error of each <b>LINK4</b> element	i r
<b>FEERRORS_LINK5</b>	Finite element error of each <b>LINK5</b> element	i r
<b>FEERRORS_TRIANG1</b>	Finite element error of each <b>TRIANG1</b> element	i r
<b>FEERRORS_TRIANG2</b>	Finite element error of each <b>TRIANG2</b> element	i r
<b>FEERRORS_TRIANG3</b>	Finite element error of each <b>TRIANG3</b> element	i r
<b>FEERRORS_TRIANG4</b>	Finite element error of each <b>TRIANG4</b> element	i r
<b>FEERRORS_TRIANG5</b>	Finite element error of each <b>TRIANG5</b> element	i r
<b>FEERRORS_QUAD1</b>	Finite element error of each <b>QUAD1</b> element	i r
<b>FEERRORS_QUAD3</b>	Finite element error of each <b>QUAD3</b> element	i r
<b>FEERRORS_TETRAH1</b>	Finite element error of each <b>TETRAH1</b> element	i r
<b>FEERRORS_BLOCK1</b>	Finite element error of each <b>BLOCK1</b> element	i r

Table 3.7: The finite element error file keywords and keyword data

An example finite element error file is shown for Figure 3.2

```
# An example finite element error file

FEERRORS_TRIANG1
1 0.0021
2 0.0032
3 0.0123
4 0.0001
```

### 3.6 The Finite Element Stress File

The element stresses are defined in the stress file (**.ste**). The number of stress points for each element must be defined in the **.mdf** file with the corresponding keyword, like **NSTRESS\_POINTS\_LINK1**, etc. For each stress point a line is defined consisting of 6 components, e.g. normal stresses in the x, y and z directions, the three shear stresses. Other interpretation is possible as well, e.g. the first three components store the principal stresses while the other three components store the angles of the principal directions. Any of the components can be ignored, e.g. in a plane problem only the first three components will be used, as the two normal stresses and a shear stress, or in the case of truss elements only one component is used.

**IMPORTANT** In the case of **TRIANG2** and **QUAD3** elements the stresses are defined at the stress points per layer per element, therefore it is a requirement that the material file is available for these elements. For all other elements the existence of the material file is not necessary to load/write stresses.

Table 3.8 shows the possible keywords.

Keyword	Keyword Data	Data Type
<b>STRESSES_LINK1</b>	6 available components of stresses	i i 6*r
<b>STRESSES_LINK2</b>	6 available components of stresses	i i 6*r
<b>STRESSES_LINK3</b>	6 available components of stresses	i i 6*r
<b>STRESSES_LINK4</b>	6 available components of stresses	i i 6*r
<b>STRESSES_LINK5</b>	6 available components of stresses	i i 6*r
<b>STRESSES_TRIANG1</b>	6 available components of stresses	i i 6*r
<b>STRESSES_TRIANG2</b>	6 available components of stresses	i i i 6*r
<b>STRESSES_TRIANG3</b>	6 available components of stresses	i i 6*r
<b>STRESSES_TRIANG4</b>	6 available components of stresses	i i 6*r
<b>STRESSES_TRIANG5</b>	6 available components of stresses	i i 6*r
<b>STRESSES_QUAD1</b>	6 available components of stresses	i i 6*r
<b>STRESSES_QUAD3</b>	6 available components of stresses	i i i 6*r
<b>STRESSES_TETRAH1</b>	6 available components of stresses	i i 6*r
<b>STRESSES_BLOCK1</b>	6 available components of stresses	i i 6*r

Table 3.8: The stress file keywords and keyword data

An example finite element stress file is shown for Figure 3.2

```
# An example stress file

# Triang1 stresses
# element index
# Stress point index
#      Ssigma x      Ssigma y      Ssigma z      Tau xy      Tau yz      Tau zx
# -----
STRESSES_TRIANG1
  1  1      1.000e+00      2.000e+00      3.000e+00      4.000e+00      5.000e+00      6.000e+00
  1  2      7.000e+00      8.000e+00      9.000e+00     10.000e+00     11.000e+00     12.000e+00
  2  1     13.000e+00     14.000e+00     15.000e+00     16.000e+00     17.000e+00     18.000e+00
  2  2     19.000e+00     20.000e+00     21.000e+00     22.000e+00     23.000e+00     24.000e+00
  3  1     25.000e+00     26.000e+00     27.000e+00     28.000e+00     29.000e+00     30.000e+00
  3  2     31.000e+00     32.000e+00     33.000e+00     34.000e+00     35.000e+00     36.000e+00
  4  1     37.000e+00     38.000e+00     39.000e+00     40.000e+00     41.000e+00     42.000e+00
  4  2     43.000e+00     44.000e+00     45.000e+00     46.000e+00     47.000e+00     48.000e+00
```



### 3.7 The Element Geometric Definition File

The element geometric definition file **.gmf** defines the boundary of a finite element problem using Non-Uniform Rational B-Splines (NURBS).

There are two most common nonlinear mathematical forms in geometric modeling for curve and surface representation, one is implicit and another is parametric polynomial forms. The implicit form has the advantage that circles, conics, and primitive quadric surfaces, such as cylinders, spheres and cones can be concisely and precisely represented. A disadvantage of the implicit form is that free-form curves and surfaces, which also important in geometric modeling can not be represented. With parametric polynomials, such as polynomial B-splines, one can represent and manipulate free-form curves and surfaces; but unfortunately, circles, conics and the quadric primitives cannot be represented precisely. Non-uniform Rational B-spline (NURBS) is a geometric modeller that offers the advantages of both forms.

Despite the versatility of NURBS in our implementation only “cubic” NURBS are implemented. Cubic NURBS are defined by two end points and two control points. (All other manipulation of NURBS, such as degree elevation, Bezier Curves conversion, knot-removal and local smoothing or modification is not present in the current implementation.)

The element geometric definition file (**.gmf**) file contains the following keywords:

- **NENDPOINTS**: Number of end points used for the geometric modelling.
- **NNURBS\_CURVES**: Number of NURBS curves.
- **ENDPOINT\_COORDINATES**: A list of coordinates of end points. These nodes will be referred by their node number in the curve specifications. (Good practice to use the same mesh points here as they are in the mesh. In this case the compatibility between the mesh and the geometric definition can always be ensured.)
- **NURBS\_CURVE**: Defines each NURBS curve.
- **DEGREE**: The number of freedom for the NURBS curve. At the moment it must always be equal to three!
- **CONTROL\_POINTS**: The four control points for Cubic NURBS curve.
- **WEIGHTS**: The value of weights for the four control points.

The geometric definition file should only be used together with the remeshing data extensions of the mesh definition file. These extensions are not covered in the main definition of the MDF syntax and can be best explained by the following example.

Figure 3.7a shows one coarse triangle defined by nodes 1,2 and 3. On the side between nodes 1 and 2 a NURBS curve **C1** is defined. The curve is cubic (degree 3) and stretches from node 1 where it has paramete 0.0 to node 2 where it has parameter 1.0. Two control points determine the shape of the curve. They have coordinates (0,-10,0) and (10,-10,0). The weights for these control points are set to 0.5. Figure 3.7b shows how the remeshed triangle may look like when three nodal meshparameters were defined with value  $\delta=0.8$ .

The following three files show how to create the curve definition in the geometric definition file and how to assign it in the mesh definition file.

```
# coarse.mdf : mesh definition file
```

```
TITLE "coarse mesh"
```

```
NMESHPOINTS      3
```

```
NNODES           3
```

```
NELEMENTS_TRIANG1  1
```

```
NNURBS_CURVES     1
```

```
MESHPOINT_COORDINATES
```

```
1  0.0  0.0  0.0
```

```
2 10.0  0.0  0.0
```

```
3  5.0  8.67 0.0
```

```
NODES_TRIANG1
```

```
1  1      2      3
```

```
REMESH_DATA
```

```
1  "EXTERIOR"
```

```
    NNURBS_CURVES  1
```

```
    1 NURBS_CURVE  "C1"
```

```
      PARAMETER    0.0
```

```
2  "EXTERIOR"
```

```
    NNURBS_CURVES  1
```

```
    1 NURBS_CURVE  "C1"
```

```
      PARAMETER    1.0
```

```
3  "EXTERIOR"
```

```
# coarse.gmf : geometric definition file
```

```
NENDPOINTS  3
```

```
ENDPOINT_COORDINATES
```

```
1  0.0  0.0  0.0
```

```
2 10.0  0.0  0.0
```

```
3  5.0  8.67 0.0
```

```
NURBS_CURVE  "C1"
```

```
DEGREE  3
```

```
CONTROL_POINTS
```

```
1  1
```

```
2  0.0 -10.0 0.0
```

```
3 10.0 -10.0 0.0
```

```
4  2
```

```
WEIGHTS
```

```
1  1
```

```
2  0.5
```

```
3  0.5
```

```
4  1
```

```
END
```

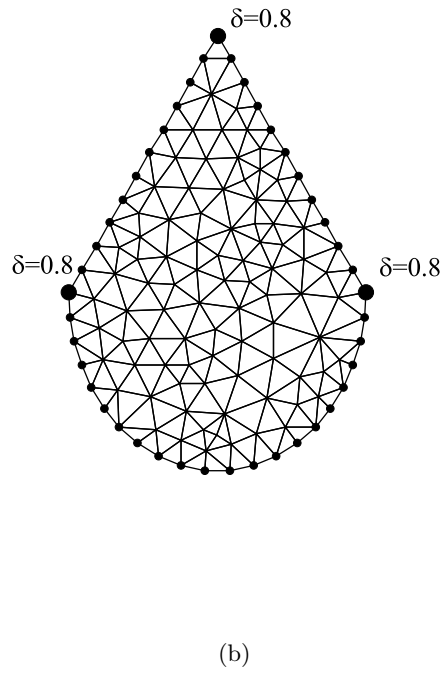
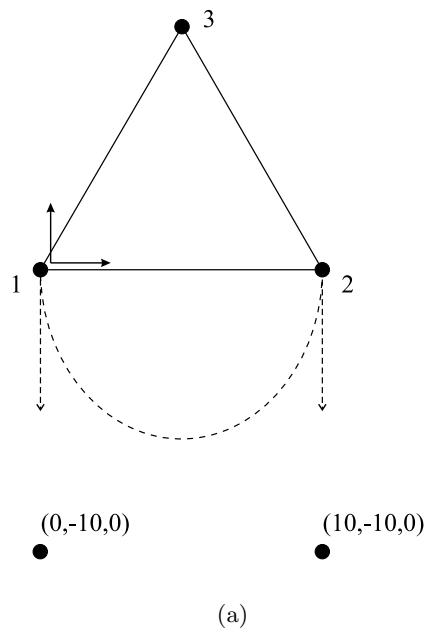


Figure 3.7: An example for geometric definition

## Chapter 4

# Unstructured mesh generation: MGN

### 4.1 Status

name of executable(s)	mgn   mgn.exe
platform(s)	irix   linux   win32
current version	v1.0
date	July 2003
release	full release
Elib filetypes	*.mdf *.mpr *.gmf
own filetypes	*.mgn *.ctl

### 4.2 Syntax

SECT Research Group, Heriot-Watt University Edinburgh

usage: mgn filename [-options]

filename : name of mesh generation batch file without extension

-g : activates the graphical interface

### 4.3 Overview

MGN is a multi-purpose unstructured 2D mesh generator. It generates, refines or remeshes 2D meshes according to four different algorithms:

- advancing front mesh generation;
- Delaunay triangulation;
- paving; and
- regular grid method.

The density of the generated mesh can be controlled by four different mechanisms:

- The mesh is uniform in size and a uniform mesh parameter is specified.
- The mesh parameter is determined from a corresponding value in a background mesh.
- The mesh parameter is directly specified for each element or each node in a mesh parameter file.
- The mesh size is controlled by a series of local control sources, points or series of points in which neighbourhood the density is specified.

Every triangular mesh, whether it was newly generated or just refined or remeshed can be converted into a quadrilateral mesh by fusion or fission of triangles. Every quadrilateral mesh, whether it was newly generated or just refined or remeshed can be converted into a triangular mesh by splitting of quadrilaterals into triangles.

Smoothing of the mesh in a specified number of iterative steps is also supported.

## 4.4 MGN batch file syntax

The tasks set for the mesh generator are specified in a batch file. The MGN batch file controls the entire behaviour of the MGN program by listing all the instructions and filenames for the input and output files. The syntax of the batch file is given in the following table. (Although this syntax is not incorporated in the E\_Library it follows the same syntax rules set for all the E\_Library files: i.e. the use of keywords the rules for comments and empty lines.)

TASK	"REMESH"   "REFINE"   "EBE_REFINE"   "POSTPROC"
METHOD	"ADV_FRONT"   "DELAUNAY" "PAVING"   "REG_GRID"   "NONE"
ALFA	d
BETA	d
DLN_OPTION	"CENTROID"   "CENTR_RAD"   "PROJECTION"   "CHEW1"
CONTROL_TYPE	"BACKGROUND"   "MPR"   "SOURCES"   "UNIFORM"
MESH_OPTION	"SPLIT_TO_TRIANG"   "FISSION_TO_QUAD"   "RANK_TO_QUAD"   "HALF_TO_QUAD"   "NONE"
GEOM_MODEL_FILE	"*"
COARSE_MESH_FILE	"*"
RESULT_MESH_FILE	"*"
BKGRND_MESH_FILE	"*"
BKGRND_MESH_PARAMS_FILE	"*"
COARSE_MESH_PARAMS_FILE	"*"
LOCAL_MESH_CTRL_FILE	"*"
UNIFORM_MESH_PARAM	d
SMOOTHING_TYPE	"LAPLACIAN"   "LAPLACIAN_NDE"   "NODAL_AVG"   "PAVING"   "NONE"
SMOOTHING_ITER	n
OUTPUT_FORMAT	"E_LIB"   "SMESH"

### 4.4.1 Tasks

First of all one has to decide what the mesh generator has to do. Four tasks are available:

- Choose "REMESH" as TASK setting for the remeshing of an existing mesh, keeping only the boundaries of the mesh and discarding any existing points on the interior of the domain. If a geometric model is supplied, a new approximation of the boundary will be made as well.
- Choose "REFINE" as TASK setting for the refinement of an existing mesh, keeping all existing nodes and refining every existing triangle or quad.
- Choose "EBE\_REFINE" as TASK setting for the refining of an existing mesh on an element by element basis.
- Choose "POSTPROC" as TASK setting if only postprocessing and mesh conversion is required.

### 4.4.2 Methods

The mesh generator supports five methods for generating the unstructured mesh:

- Choose "ADV\_FRONT" as METHOD setting for the advancing front technique which is rather slow, but creates very regular high quality meshes.
- Choose "DELAUNAY" as METHOD setting for the Delaunay triangulation algorithm (by point insertion) which is on average ten times faster, but for which regularity and quality of the mesh can be less than advancing front.
- Choose "PAVING" as METHOD setting for the direct generation of quadrilaterals using the paving technique.
- Choose "REG\_GRID" as METHOD setting for the regular grid method, which is the subject of reference [2]

When the Delaunay triangulation or regular grid method is chosen, the user has to specify two extra parameters ALFA and BETA and an extra option DLN\_OPTION.

- Alfa controls the conformity of the mesh density with the parameters set by the control type. For DLN\_OPTION set to "CENTROID" the requirement for alfa is  $1.0 < \alpha$ . For DLN\_OPTION set to "CENTR\_RAD" the requirement for alfa is  $0.5 \leq \alpha \leq 2.0$ . Values close to 1.0 are a good starting point. For METHOD set to "REG\_GRID" alfa takes the function of parameter `gprm` from reference [2].
- Beta ( $0.1 \leq \beta \leq 15.0$ ) controls the regularity of the mesh. Lower values of beta generate more chaotic meshes whereas higher values would give a more regular mesh. A good default value is 2.0. For METHOD set to "REG\_GRID" beta takes the function of parameter `rprm` from reference [2].

### 4.4.3 Delaunay options

When the Delaunay triangulation is chosen the user must define which Delaunay refinement strategy is used. This is controlled by the DLN\_OPTION keyword. The following options are implemented:

- "CENTROID" controls the mesh size on the basis of side lengths of triangles. Centroidal nodes are inserted into a triangle when at least one of its side lengths is longer than allowed.
- "CENTR\_RAD" controls the mesh size on the basis of the lengths of the radial connection between the centroid node and the corner node of triangles. Centroid node is inserted into a triangle when any of the radial connections are longer than allowed.
- "PROJECTION" behaves like "CENTROID" but uses projective point placement instead of centroidal nodes.
- "CHEW1" uses the point insertion method suggested by Chew [1].

### 4.4.4 Control types

An efficient control of the mesh density at any place inside the domain and on the borders is essential for any good mesh generator. MGN offers four control types:

- Choose "UNIFORM" as CONTROL\_TYPE for the generation of uniform meshes. The uniform size will be taken from the value specified for the UNIFORM\_MESH\_SIZE keyword.
- The use of a background mesh which sits behind the domain is a common technique. For every new point to be generated in the mesh the meshparameter is derived from the value this point would have if it belonged to the background mesh. Therefore care has to be taken that the background mesh covers the complete area of the domain, including potential extensions of the domain by a new approximation of the boundary curves. The background mesh also has to be convex. Choose "BACKGROUND" as CONTROL\_TYPE setting and don't forget to define the BKGRND\_MESH\_FILE and the BKGRND\_MESH\_PARAMS\_FILE keywords.

- Mesh parameter controlled meshing is equivalent to background controlled meshing where the background mesh is the coarse mesh itself. However the mesh parameters specified in the `COARSE_MESH_PARAMS_FILE` have to be nodal mesh parameters to make sense. (If element mesh parameters are specified they will be converted to nodal parameters by averaging and will therefore lose effectiveness.) Choose "MPR" as `CONTROL_TYPE` and don't forget to define the `COARSE_MESH_PARAMS_FILE` keyword.
- The use of local control sources is the third type. It is an extension to background controlled meshing whereby the user can define some local control sources in a separate file that will influence the mesh density on a very small scale. Section 4.5 will explain the syntax and the effect of these sources. Choose "SOURCES" as `CONTROL_TYPE` and don't forget to define the `LOCAL_MESH_CTRL_FILE` keyword.

#### 4.4.5 Mesh options

When the coarse mesh is not triangular but contains quadrilaterals, you have to specify a mesh option when you generate the mesh using Delaunay triangulation. This option will determine the way the Delaunay triangles are converted into quads so that the output is of the same element type as the input.

When the coarse mesh is triangular, you can specify a mesh option anyway if you want the output to be a mesh of quads. This works for both mesh generation methods.

Currently three quad conversion options are supported:

- "FISSION\_TO\_QUAD": In this method triangles are merged to quad elements. The method also considers the case when by merging two triangles two subregions would form with odd number of elements in them. In this case the method inserts a fission element.
- "RANK\_TO\_QUAD": In this method triangles are also merged to quad elements, regardless of any other constraint. In this case there can be cases when individual triangles are left in the mesh surrounded by quad elements which cannot be merged with any other triangle. To solve this problem the method also halves every element in the mesh (quads into four and triangles into three quad elements) after there is no more triangle merge is possible.
- "HALF\_TO\_QUAD": This method simply halves every triangle element into three quad elements.
- Choose "NONE" if you don't want any triangle to quad conversion.

If you are directly generating quadrilaterals using the "PAVING" method, and you want a triangular mesh as output you can set the mesh option to "SPLIT\_TO\_TRIANG" to activate a splitting procedure that will cut each quadrilateral into two triangles along the shortest diagonal.

#### 4.4.6 Filenames

Filename references have to be absolute or relative to the location of the executable. Filenames are always given without extension as it is assumed that the extension matches the filetype syntax. Filenames are enclosed in double quotes.

The following filenames may be defined:

- `GEOM_MODEL_FILE`: the location and name of the geometric model file without the `*.gmf` extension. This file is required for new mesh generation from geometric model only. It may be, but doesn't have to be supplied for the other tasks.
- `COARSE_MESH_FILE`: the location and name of the coarse mesh definition file without the `*.mdf` extension. This file is not required when performing new mesh generation from geometric model only. It is required in the other tasks.
- `RESULT_MESH_FILE`: the location and name of the resulting mesh to which an extension will be added according to the output format. If the file exists already, it will be overwritten. This filename always has to be supplied.

- **BKGRND\_MESH\_FILE**: the location and name of the background mesh definition file without the **\*.mdf** extension. This file is required when background or sources controlled mesh generation is chosen.
- **BKGRND\_MESH\_PARAMS\_FILE**: the location and name of the background mesh parameters file without the **\*.mpr** extension. This file is required when background or sources controlled mesh generation is chosen.
- **COARSE\_MESH\_PARAMS\_FILE**: the location and name of the mesh parameters file without the **\*.mpr** extension. This file is required when mesh parameter controlled mesh generation is chosen.
- **LOCAL\_MESH\_CTRL\_FILE**: the location and name of the local control sources file without the **\*.ctl** extension. This file is required when local control sources are requested.

#### 4.4.7 Smoothing

The mesh that results from any of the operations mentioned above can be smoothed. The only smoothing type that is currently supported is optimized Laplacian smoothing. Choose "LAPLACIAN" or "LAPLACIAN\_NDE" if Laplacian smoothing with or without diagonal exchange is required. "NODAL\_AVG" will activate pure nodal averaging. if "NONE" is chosen as **SMOOTHING\_TYPE** setting no smoothing is carried out at all. If smoothing is activated, you also have to specify the number of smoothing iterations to be carried out. This number is set by **SMOOTHING\_ITER** and has to be bigger than one. Two iterations are in most cases sufficient.

In the case of paving the smoothing type must be set to "PAVING" as the paving method has its own smoothing procedures.

#### 4.4.8 Output format

The output format of the resulting mesh can be specified by **OUTPUT\_FORMAT**. Currently only "E\_LIB" and "SMESH" are supported. "E\_LIB" will generate E\_Library compatible output and will add the **\*.mdf** extension to the resulting mesh filename. "SMESH" will generate meshes in the format for the acoustics software **son3d** and will add the **\*.smsh** extension to the resulting mesh filename.

### 4.5 CTL local control sources syntax

The local mesh density control data file (**.ctl**) defines a set of local control points for point-wise and line-wise mesh size control. For the point-wise local control the point coordinates, the radius of inner circle, the radius of outer circle and nodal mesh size parameter are specified. For line-wise local control a chain of control points will be specified.

The syntax will be explained by an example file:

```
NCONTROL_SETS      2

NCONTROL_POINTS    2
COORDINATES        8.5 2.0 0.0
INNER_RADIUS        0.2
OUTER_RADIUS        1.0
MESH_PARAMETER      0.3
COORDINATES        1.5 2.0 0.0
INNER_RADIUS        0.2
OUTER_RADIUS        1.5
MESH_PARAMETER      0.2

NCONTROL_POINTS    1
COORDINATES        2.0 5.5 0.0
INNER_RADIUS        0.2
OUTER_RADIUS        1.0
MESH_PARAMETER      0.2
```



## Chapter 5

# Structured mesh generation: CUBIC

### 5.1 Status

name of executable(s)	cubic   cubic.exe
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	*.mdf
own filetypes	*.gen

### 5.2 Syntax

Usage: cubic filename [options]

Options:

-h          Print help, this screen

### 5.3 File format

This chapter describes the file format for the **CUBIC** structured mesh generator program. The extension of the input file must be **.gen**. This file also follows the **E\_Library** conventions with regards to keywords, comments and empty lines.

### 5.4 Entity definitions

Figure 5.1 shows the conventions for the definition of the different entity types. The origin of the local, element coordinate system is the first node of the geometric entity. The 2D surfaces are defined in an anti-clockwise manner which ensures that the final finite elements will also be defined in a counter-clockwise manner. The first local coordinate direction ( $\xi$ ) is parallel with the edge defined by the first and the second nodes of the geometric entity. In the case of 2D geometric entities the second local coordinate direction ( $\eta$ ) is parallel with the edge determined by the first and the last node of the entity. The first and second nodes determine the first side, the second and third node specifies the second side and so on. The nodes are created and numbered in a row as it is also shown in Figure 5.1 for all 1D and 2D entity types.

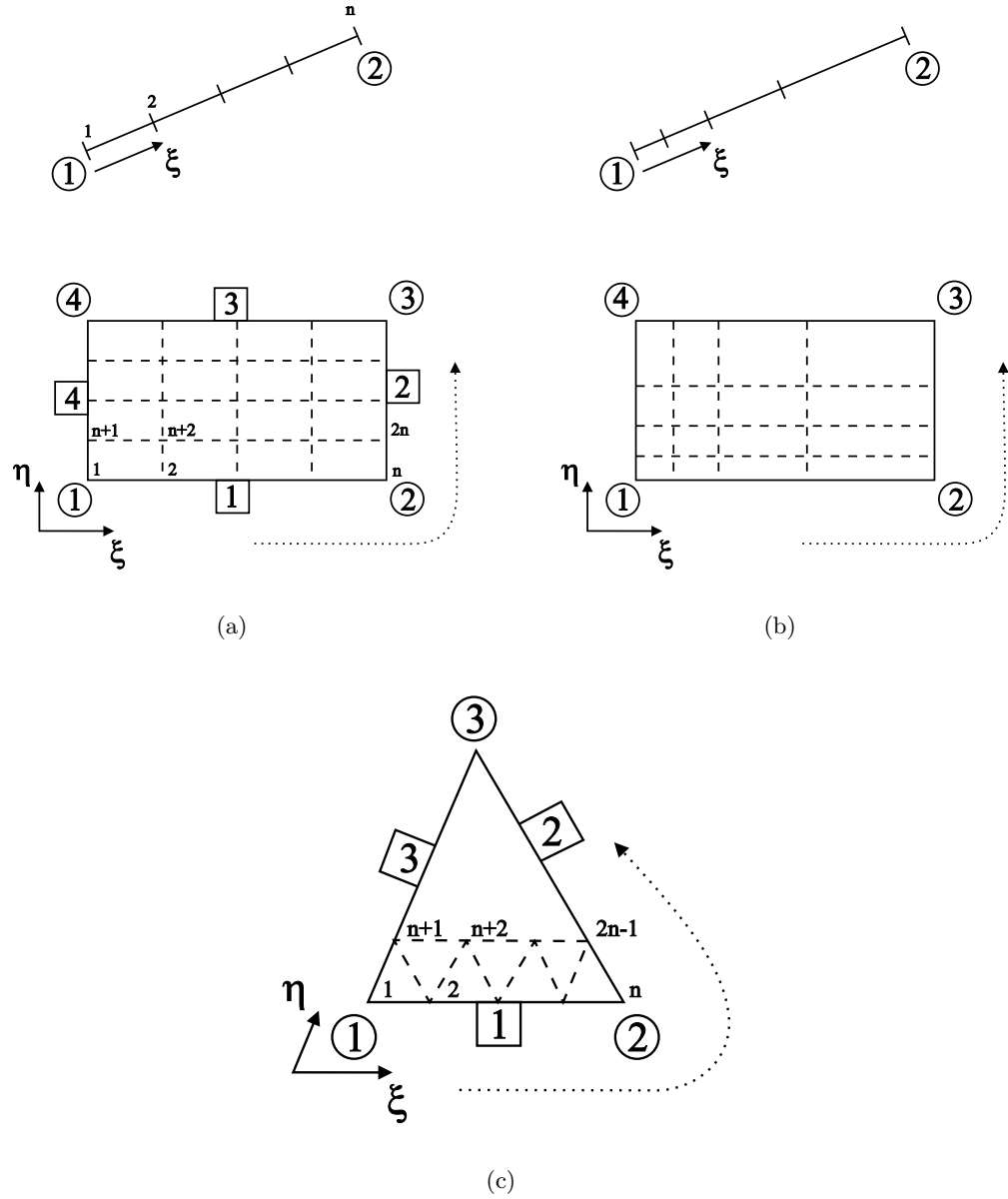


Figure 5.1: Conventions for the entity definition, (a) uniform quadrilateral, (b) nonuniform quadrilateral and (c) triangular entities

## 5.5 Data file format

The input to the program is a textual data file which describes the geometric model of the structure. The data structure has been designed in such a way that the sequence of mesh definition can be carried out easily at one step. The file starts with a header section, then lists the point entities, the two and the one dimensional entities and finally the boundary conditions. Comment and empty lines can be inserted at any line in the file, because these lines are automatically ignored during the reading of the input file. The comment lines start with a “#” sign. Most of the data is provided in a keyword-data format. The information should be placed into one line, new lines cannot be started in the middle of the data definition.

```

TITLE "this is the descriptive title"
NNODES      i
NSURFACES   j
NLINKS      k
( GEN_MATERIAL )
( NMATERIALS 1 )
( NBC_NODES  m )
( NBC_SIDES  n )

```

Table 5.1: Header section of the geometric data file

### 5.5.1 Header section

The header section of the input file is defined in Table 5.1 where the **TITLE** keyword defines the title of the project, the **NNODES** keyword defines the number of point entities, the **NSURFACES** keyword defines the number of 2D entities and the **NLINKS** keyword defines the number of 1D entities. Later in the data file the above given number of entities should be specified. The entities are not defined in the order of their dimension, like first points, one dimensional and finally two dimensional elements, because the two dimensional entities are thought to be the main components of the structure, while the one dimensional entities are considered as added components. In spite of this, it is possible to generate a structure containing only one dimensional elements.

The bracketed keywords are optional. The **GEN\_MATERIAL** keyword simply specifies that for the final mesh material information should be generated. The next **NMATERIALS** keyword provides the maximum number of user definable material types. If only the first keyword is specified, then all finite elements will have the default material definition, which is assigned to them according to their type. In this case all of the same element types will have the same material. If the second keyword is defined, then extra material definition is allowed by the user. This means that it is possible to assign different material to all finite elements generated from the same geometric entity. The geodesic strings (link type 5 according to the E-lib specification [4]) are exceptions, because their material cannot be overdefined, they will always have the default material definition. The last two keywords **NBC\_NODES** and **NBC\_SIDES** define the number of boundary condition points and the number of boundary condition sides.

### 5.5.2 Point entity definition

In this section those points are defined by their three coordinate values, which are used in the later sections to define higher dimension entities. The number of points given in this section must be the same as **NNODES** defined in the header section, but it is not necessary to use all the defined points. Those points, which are not referenced by their index in the later sections are ignored and they will not appear in the final mesh.

The format of this section is shown in Table 5.2, where **x1-coord**, **y1-coord** and **z1-coord** provide the position of the first point in the global coordinate system. The definition of all points are similar.

```

NODE_COORDINATES
1  x1-coord  y1-coord  z1-coord
2  x2-coord  y2-coord  z2-coord
.
.
.
i  xn-coord  yn-coord  zn-coord

```

Table 5.2: Point entity definition in the geometric data file

### 5.5.3 Surface entity definition

In this section the two dimensional entities, the quadrilateral and the triangular surfaces are defined. They can be defined in any order, but their numbering should be consecutive starting from one and the total number of these entities should be equal to **NSURFACES** specified in the header section. At the moment the triangular surfaces can be defined only with straight edges, while the quadrilateral surfaces are defined as cubic patches. A special case of the cubic patch definition is when the quadrilateral surface has straight edges. To help the insertion of one dimensional elements into the final mesh (like geodesic strings) those one dimensional elements which are part of the surface are also defined in this section. The format of the surface entity definition is presented in Table 5.3.

```

SURFACE n
  SHAPE          TRIANG | QUAD | 3 | 4
  ELEMENT_TYPE   TRIANG1 | TRIANG3 | TRIANG4 | TRIANG5 |
                  [ QUAD1 ]
  DEFINITION     [ LINEAR | PARAMETRIC | POINT ]
  nodes | [ nodes consts ]
  N_DIVISIONS n [ n ]
  ( DIVISION_WEIGHTS
    weights
    weights
    < weights > )
  ( LINK_DIRECTION 1 | 0   1 | 0   < 1 | 0 >
    LINK_DIVISION
      division
      division
      < division >
    LINK_TYPE
      types
      types
      < types > )
  ( MATERIAL n )
  [ DIAGONAL 1 | 2 | SHORTEST | LONGEST ]
END

```

Table 5.3: Surface entity definition in the geometric data file

The bracketed sections are optional, the square brackets ('[',']') surround options which can be defined only for quadrilateral geometric entities and the ('<', '>') brackets contains expressions only for the triangular geometric entities. The vertical line ('|') is written between options, from which at least one should be defined. **n** represents an integer value. The **SHAPE** keyword specifies the form of the surface which can be quadrilateral or triangular in shape. A triangular geometric entity is simply defined by its corner nodes after the **DEFINITION** keyword. On the other hand the quadrilateral geometric entity can be defined in three different ways. In the case of the **LINEAR** definition the four corner nodes should be supplied, in the case of the **POINT** 16 nodes should be specified to define a cubic patch and in the case of the **PARAMETRIC** four corner nodes and the derivatives are required to define the cubic patch. The order of the derivatives for the parametric definition is:

$$X_{\xi} \ Y_{\xi} \ Z_{\xi} \ X_{\eta} \ Y_{\eta} \ Z_{\eta} \ X_{\xi\eta} \ Y_{\xi\eta} \ Z_{\xi\eta} \quad (5.1)$$

A **node** can be a reference to a previously defined point entity or to a generated node of one of the entities. Allowing the later case of point referencing the definition of the surface topology can be simplified. Two examples are shown in Figure 5.2 where the surfaces do not have to be cut up into further pieces to ensure the compatibility between the adjacent surfaces. The format of the *reference point* is either **s n x y** if it references a node in a two dimensional entity or **l n x** if it references a node in a one dimensional entity, where **n** is the index of the entity and **x** and **y**

define the position of the node in the local coordinate system of the referenced geometric entity (see Figure 5.1). An entity can reference only such an entity in which the mesh generation has been finished. For example, `s 1 1 3` references a node in the first surface entity which is in the first column and in the third row of the to be generated points. The example is shown in a quadrilateral and in a triangular surface in Figure 5.3.

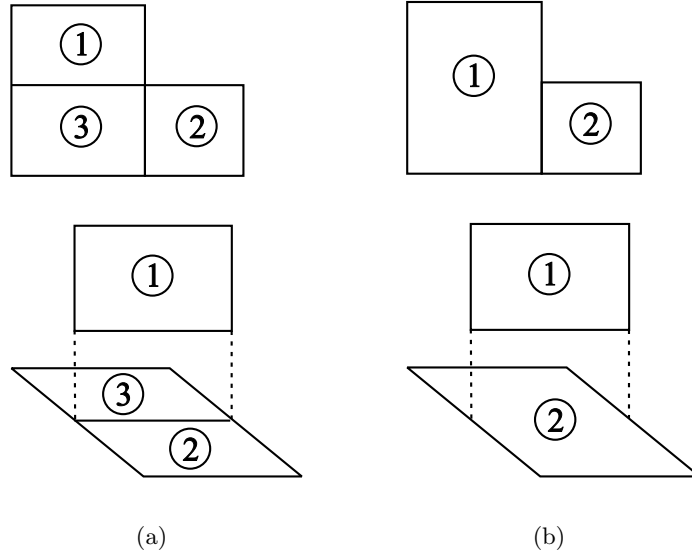


Figure 5.2: Surface definition (a) without and (b) with “referencing points”

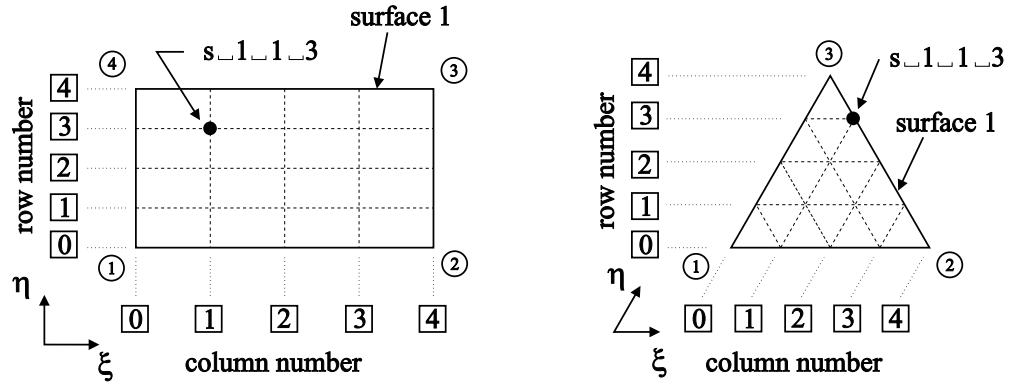


Figure 5.3: Referenced node in the first surface entity

The `ELEMENT_TYPE` keyword specifies the type of the finite element which will be generated in the geometric entity. Quadrilateral elements cannot be generated in a triangular geometric entity. The name of the element types are references to the available element types in the E-lib specification [4] because the generated final finite element mesh will comply the E-lib specification. The `DIVISION` keyword specifies the number of divisions on the sides of the entities. In the case of a triangular entity all sides are divided into equal number of divisions, but if required the divisions can have different lengths on all sides. In the case of the quadrilateral entities the first integer number specifies the division of the first and the third sides, while the second number specifies the division on the second and the fourth sides. If no more data is specified before the `END` keyword, then the geometric entities are divided uniformly and no one dimensional element will be inserted into the surface or to the edge of the surface.

The lines after the optional `DIVISION_WEIGHT` keyword specify the division weights on the sides. The `weights` can be any real numbers except zero. If the weight is zero, the program will use one as a weight. The specified `weights` are normalised. The `weights` are either specified as

a series of double numbers or using the syntax [3] of “**n ( d )**” where the **d** real number is considered **n** times after each other. Any combination of the two syntaxes is also acceptable. The **LINK\_DIRECTION** keyword specifies whether there are one dimensional elements to insert into the surface. The position of the one (‘true’) values specify the side which the one dimensional elements will be parallel with (see Figure 5.4). The **LINK\_DIVISION** and **LINK\_TYPE** keywords specify the position and the type of the one dimensional elements. The available types comply with the E-lib specification [4] which allows five types of one dimensional element at the moment. The positions of the one dimensional elements are given as a series of integer values.

The first integer value in the list specifies the distance from the corresponding side. The distance is expressed in terms of the generated rows of elements. The other values in the list are increments from the previous position. An example is shown in Table 5.4. Finally the material can be defined by the **MATERIAL** keyword. The number **n** specifies that which user defined material should be assigned to all of the generated surface elements in the surface entity. The number cannot be larger than **NMATERIALS** defined in the header section. The final keyword **DIAGONAL** can be used to control the division of quadrilateral elements into triangles in a quadrilateral geometric entity. The default value is **SHORTEST** which defines that the shortest diagonal of the two diagonals will be used in the geometric entity. The other options allow to use the longer, first or second diagonal.

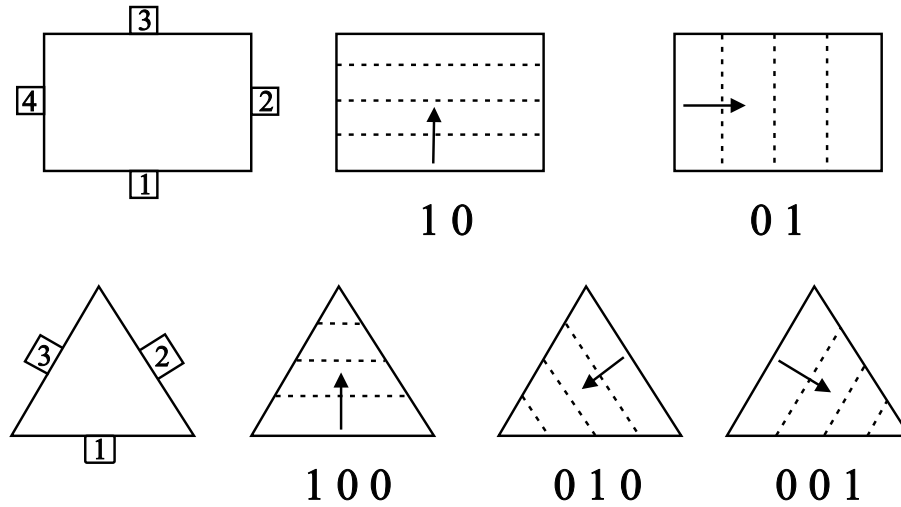


Figure 5.4: **LINK\_DIRECTION** values and the corresponding direction of the one dimensional elements, arrows also show the order of the definition of the elements according to **LINK\_DIVISION**

#### 5.5.4 Link entity definition

In this section the one dimensional entities are defined. They can be defined in any order, but their numbering should be consecutive starting from one and the total number of these entities should be equal to **NLINKS** specified in the header section. At the moment these elements can be defined only as straight lines. The format of the link entity definition is presented in Table 5.5. Comments about the **nodes**, element type, division of the element and user definable material can also be applied for these types of element. The difference is that for the one dimensional elements it is not necessary to define the **DIVISION**.

#### 5.5.5 Boundary condition definition section

The final section contains the boundary condition definitions. Geometrically two types of boundary condition can be defined: point or line (or area for quadrilateral surfaces). In the final mesh format only point boundary conditions exist, according to the E-lib specification [4], therefore all boundary conditions are translated into a point boundary condition. The format of the boundary condition

```

LINK_DIRECTION 1 1
LINK_DIVISION
2 2
0 2 1
LINK_TYPE
3 3
3 5 5

```

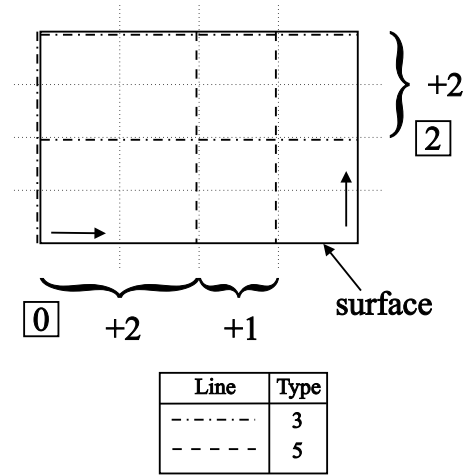


Table 5.4: Example for the definition of one dimensional elements in a surface entity

```

LINK n
  ELEMENT_TYPE LINK1 | LINK2 | LINK3 | LINK4 | LINK5
  DEFINITION nodes
  ( N_DIVISIONS n )
  ( DIVISION_WEIGHTS
    weights )
  ( MATERIAL n )
END

```

Table 5.5: Link entity definition in the geometric data file

definition section is presented in Table 5.6. First the point boundary conditions are listed. The point boundary conditions can be defined only for the point entities and they can be listed in any order. The only requirement is that the number of the point boundary conditions should be equal to `NBC_NODES` specified in the header section. The `bc_type` can be `"FREE"`, `"FIXED"` `d` or `"SPRING"` `d` where `d` is a real number and expresses the support displacement or the spring constant respectively.

```

BOUNDARY_CONDITIONS
point_entity_index_1  bc_type  bc_type  bc_type
point_entity_index_2  bc_type  bc_type  bc_type
.
.
.
point_entity_index_m  bc_type  bc_type  bc_type

ref_pnt_1  ref_pnt_2  bc_type  bc_type  bc_type
ref_pnt_3  ref_pnt_4  bc_type  bc_type  bc_type
.
.
.
ref_pnt_5  ref_pnt_6  bc_type  bc_type  bc_type

```

Table 5.6: Boundary condition definition in the geometric data file

The boundary condition sides are defined by two “reference points” in an entity and they must lie on a straight line. The “reference point” definition is explained in section 5.5.3. A special case is when the entity is a quadrilateral surface and the two reference points mark an area as shown in

Figure 5.5. In this case the same boundary condition will be generated for all points inside the area, including the side and corner points.

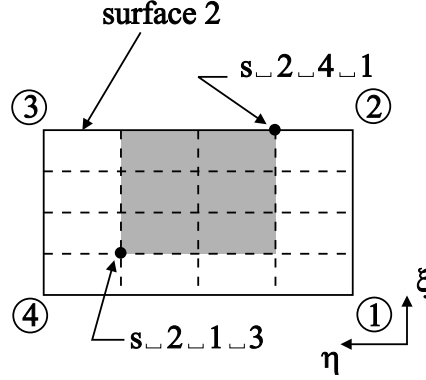


Figure 5.5: Boundary condition generation defined by two reference points for an area, the same boundary conditions will be generated for the points inside and on the edge of the shaded area

First the boundary condition “sides” are generated in the order of their definition. If there are overlapping areas then the boundary conditions of the points in the overlapping area will be redefined and only the later definition will be stored. Finally the boundary condition points are defined and they may redefine the boundary condition for a point again. At every occasion of a redefinition of a boundary condition the program issues a warning message. The order of the generation of the boundary conditions and the possibility of the redefinition of previously defined boundary conditions allows, that at corners of areas or at intersection of boundary condition lines such a boundary condition can be defined which may be a combination of the different crossing boundary conditions.

## 5.6 Examples for parametric definition

An example is shown in Table 5.7. On the left-hand side the geometric file definition and on the right-hand side the generated mesh is shown. The values of the **PARAMETRIC** definition are determined according to Figure 5.6. In Figure 5.6a the real domain ( $\Omega$ ) is shown. The straight lines determine the quadrilateral surface and the dashed lines are the required shape of the surface. In Figure 5.6b the unit square domain ( $\bar{\Omega}$ ) is presented. The orientation of the real domain and the unit square domain is the same, in this case it is counter-clockwise. To determine the derivative values for the geometric file which are the components of the tangential vectors in the  $\xi$  and  $\eta$  directions in the global coordinate system someone has to inspect the orientation of the  $\xi, \eta$  coordinate system at the particular node.

For example at node 1 in the  $\bar{\Omega}$  domain the  $\xi$  coordinate direction points from node 1 to node 2, which in the real domain  $\Omega$  is a vertical vector. This is represented in the file as 0 3 0. The  $\eta$  direction in the  $\bar{\Omega}$  domain points from node 1 to node 4 which in the real domain  $\Omega$  can be determined in a similar way (continuous  $\eta$  vector at node 1 in Figure 5.6). On the other hand in the real domain the side between node 1 and 4 is not a straight line and the  $\eta$  derivative (tangential vector) at node 1 is actually a horizontal vector as shown in Figure 5.6a by a dashed vector. The representation of this vector in the geometric file is -4.71 0 0. The length of the vector should be equal to the real side length to ensure equidistant distribution of the points. In this case the quarter length of the perimeter of a circle with a 3 unit radius is ( $2r\pi/4 \approx 4.71$ ). The last three values for node 1 in the geometric file are zero since there is no “twist” defined. The effect of the “twist” is demonstrated in Table 5.14

Another example where the same square surface is defined by the **LINEAR** and the **PARAMETRIC** method is shown in Table 5.8. Some other possible variations of the square are shown in Table 5.9-5.14. In the tables all examples are in 2D except the last one, which is a 3D surface.



# NODE\_COORDINATES

```

1 6.0 3.0 0.0
2 6.0 6.0 0.0
3 0.0 0.0 0.0
4 3.0 0.0 0.0

```

# SURFACE 1

```

SHAPE QUAD
ELEMENT_TYPE TRIANG3

```

# DEFINITION PARAMETRIC

```

1 0 3 0 -4.71 0 0 0 0 0
2 0 3 0 -9.42 0 0 0 0 0
3 -3 0 0 0 -9.42 0 0 0 0
4 -3 0 0 0 -4.71 0 0 0 0
N_DIVISIONS 10 10

```

END

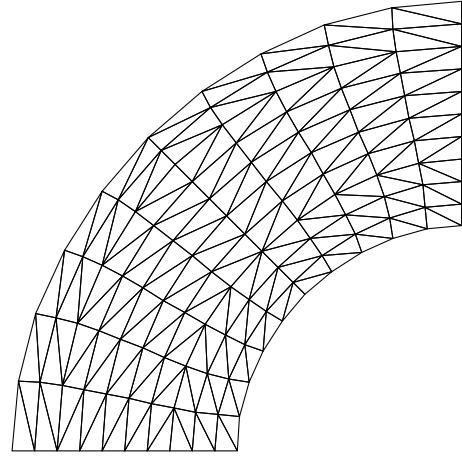
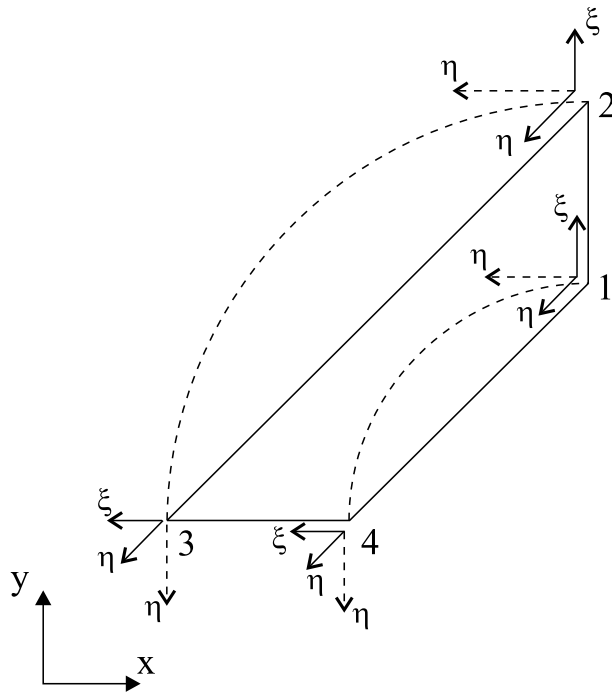
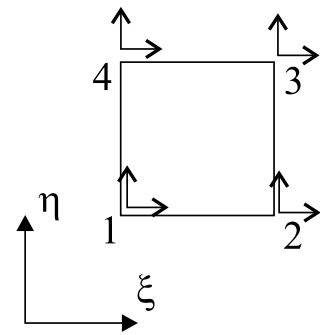


Table 5.7: A complex example to show the geometric file definition and the generated mesh



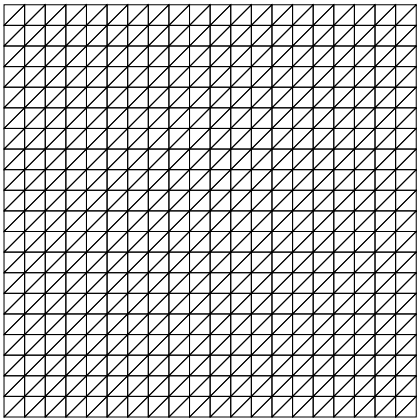
(a)



(b)

Figure 5.6: The real domain  $\Omega$  (a) and the unit square domain  $\bar{\Omega}$  (b) for the complex example

DEFINITION	LINEAR
1	2 3 4



DEFINITION	PARAMETRIC
1	1 0 0 0 1 0 0 0 0
2	1 0 0 0 1 0 0 0 0
3	1 0 0 0 1 0 0 0 0
4	1 0 0 0 1 0 0 0 0

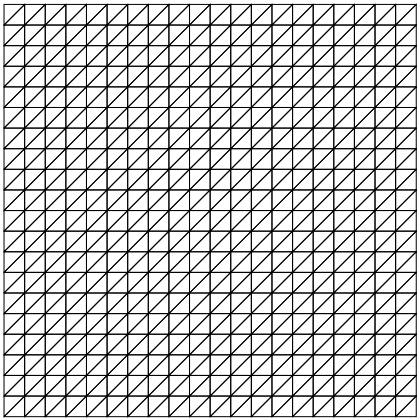


Table 5.8: LINEAR and PARAMETRIC definition of a square

DEFINITION	PARAMETRIC
1	1 1 0 0 0 1 0 0 0 0
2	1 0 0 0 0 1 0 0 0 0
3	1 0 0 0 0 1 0 0 0 0
4	1 0 0 0 0 1 0 0 0 0

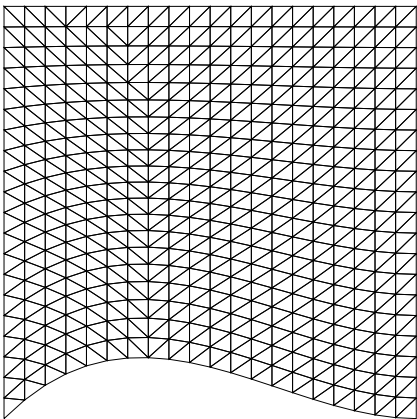


Table 5.9: Example for the definition of cubic surfaces

DEFINITION	PARAMETRIC
1 1 0 0	1 1 0 0 0 0
2 1 0 0	0 1 0 0 0 0
3 1 0 0	0 1 0 0 0 0
4 1 0 0	0 1 0 0 0 0

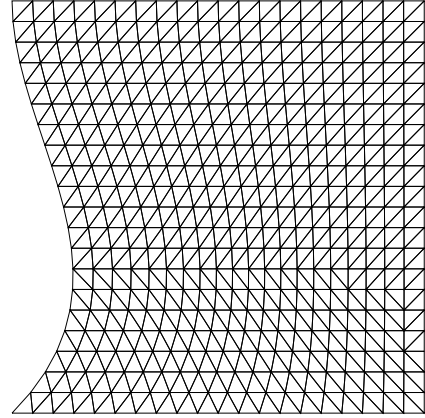


Table 5.10: Example for the definition of cubic surfaces

DEFINITION	PARAMETRIC
1 1 1 0	1 1 0 0 0 0
2 1 0 0	0 1 0 0 0 0
3 1 0 0	0 1 0 0 0 0
4 1 0 0	0 1 0 0 0 0

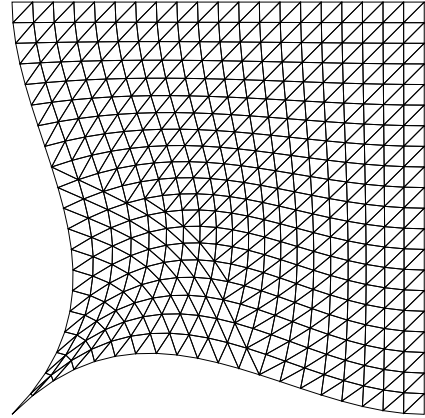


Table 5.11: Example for the definition of cubic surfaces

DEFINITION	PARAMETRIC
1 1 0 0	0 1 0 10 0 0
2 1 0 0	0 1 0 0 0 0
3 1 0 0	0 1 0 0 0 0
4 1 0 0	0 1 0 0 0 0

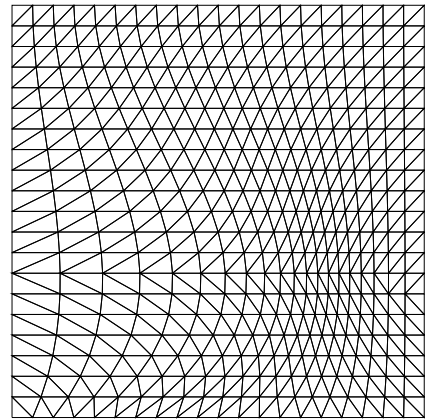


Table 5.12: Example for the definition of cubic surfaces

DEFINITION	PARAMETRIC
1	1 0 0 0 1 0 10 10 0
2	1 0 0 0 1 0 0 0 0
3	1 0 0 0 1 0 0 0 0
4	1 0 0 0 1 0 0 0 0

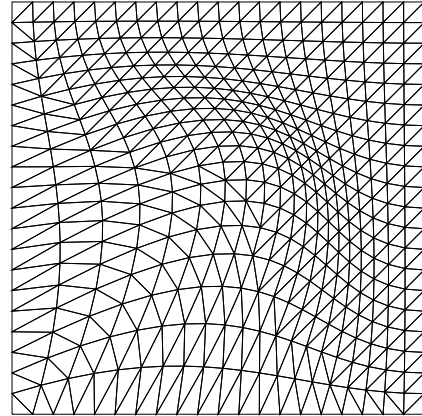


Table 5.13: Example for the definition of cubic surfaces

DEFINITION	PARAMETRIC
1	1 0 -1 0 1 0 0 0 0
2	1 0 1 0 1 0 0 0 0
3	1 0 1 0 1 0 0 0 0
4	1 0 1 0 1 0 0 0 0

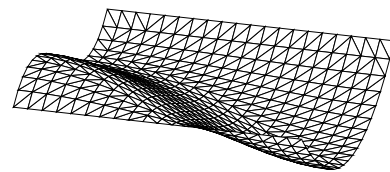


Table 5.14: Example for the definition of cubic surfaces

## Chapter 6

# 2D Finite element analysis: FEM

### 6.1 Status

name of executable(s)	fem   fem.exe
platform(s)	irix   linux   win32 command line
current version	v1.1
date	July 2003
release	unfinished
E_lib filetypes	*.mdf *.mat *.ste
own filetypes	-

### 6.2 Syntax

fem v1.1 SECT Research Group, Heriot-Watt University Edinburgh

usage: fem input output ds

note all filenames are referred to without extensions

input : name of the mesh definition and material file

output : name of the output mesh definition file

ds : displacement scale (e.g. 10)

you might want to renumber the mesh first!

and remove unused nodes!

### 6.3 Overview

**fem** is a very limited 2D finite element program which calculates stresses and displacements for meshes consisting of **TRIANG1** and **QUAD1** elements only. Loads, boundary conditions and material assignments should be specified in the input mesh file. Material information should be given in the material file with the same name as the mesh definition file.

As **TRIANG1** and **QUAD1** are constant stress elements the output stress file will consist of one stress-point per element. As **fem** is a 2D FE package only three stresses will appear in the output stress file:  $\sigma_x$ ,  $\sigma_y$  and  $\tau_{xy}$ .

In order to increase the efficiency of the computation and to reduce the calculation times you should not include any unused nodes in the coordinates array. Renumbering of the nodes to decrease the bandwidth of the stiffness matrix is also advised. The **renumber** program can be used for this purpose. **renumber** is explained in Section 9.11.

## 6.4 Program limitations

- Only meshes consisting entirely of one of the two supported element types can be analysed.
- From the material file only `YMOD`, `POISSONS_RATIO` and `THICKNESS` will be taken into account.
- The program gives no status reports and might take ages if you have not renumbered your mesh.
- The program will crash if the mesh contains unconnected nodes. These are nodes which are not used in any of the element definitions. Use the `cleanup` program to remove unconnected nodes. `cleanup` is explained in Section 9.2

## Chapter 7

# Error analysis: ADAPT

### 7.1 Status

name of executable(s)	adapt   adapt.exe
platform(s)	irix   linux   win32 command line
current version	v1.1
date	July 2003
release	development
E_lib filetypes	*.mdf *.mat *.ste *.fee *.mpr
own filetypes	-

### 7.2 Syntax

adapt v1.1 SECT Research Group, Heriot-Watt University Edinburgh

usage: adapt input d mprtype

note all filenames are referred to without extensions

input	: name of the mesh definition and stress file
d	: permissible error value
mprtype	: 1 for element mesh params
	: 2 for nodal mesh params

### 7.3 Overview

adapt is small error analysis program for the sort of meshes that can be handled by fem. It generates finite element errors on the basis of the stress file by comparing averaged and non averaged nodal stresses.

The finite element element errors are then converted into mesh parameters either nodal or element parameters as specified in the command line. Make sure you choose the right type depending on what remeshing or viewing you want to do next.

It is always advised to check the generated mesh parameters before a new remeshing run is launched. Some mesh parameters might be far too small to be realistic. The floor program might be used to set a lower limit to the mesh parameters. floor is explained in Section 9.4.

### 7.4 Program limitations

- Only meshes consisting entirely of one of the two supported element types can be analysed.
- The material file should also be there as material information is required for processing the finite element errors.

- Both material and stress files are expected to have the same name as the mesh definition file.



## Chapter 8

# Viewing and printing: E\_PLOT32

### 8.1 Status

name of executable(s)	e_plot32.exe
platform(s)	win32
current version	v2.52
date	July 2003
release	complete
E_lib filetypes	*.mdf *.mpr *.dom *.fee *.ste
own filetypes	-

### 8.2 Syntax

run e\\_plot32.exe from explorer or start menu - run...  
or make a shortcut to e\\_plot32.exe

### 8.3 Overview

E\_plot32 is a native windows 32bit graphical user interface program. It will run on Windows95, Windows98 and Windows NT 4. The program provides a viewer for all sorts of meshes in the E\_Library mesh definition format.

The user can view the mesh in the following plot configurations:

- plain mesh plot: just shows the mesh connectivity.
- subdomain plot: colours the different subdomains according to the subdomain information stored in the appropriate subdomain file.
- stress plot: shows a colour representation of all the stresses that are available in the stress file.
- FEError plot: plots the distributions of the finite element errors as specified in the FEError file.
- mesh parameter plot: makes a plot of the mesh parameters as stored in the mesh parameter file.

The program can print directly from the menu and also has an export facility to the \*.wmf format which can be read by many vector graphics packages such as CorelDraw and AutoCAD 2D.

## 8.4 Help

For a more detailed explanation of each menu option we refer to the online help system within the program. Help is available by selecting the Help menu or by simply pressing **F1** at any time in the program.

## 8.5 Program limitations

- All files should have the same name as the input **\*.mdf** files with the appropriate extensions.
- All filetypes are accepted except **QUAD3** and **BLOCK1**.
- For meshes containing **TRIANG2** elements: only the stresses for **layer 0** are shown in a stress plot.
- When exporting a subdomain plot to **\*.wmf** format the subdomain colours are converted to grayscale. This does not happen with the stress, **FError** or mesh parameter plots.
- An export to **\*.wmf** might mess up the scaling of the image. Scale it by a factor 100 to obtain normal sizes. However this depends on the graphics package you use.
- **E\_plot32** will only accept element mesh parameters for a mesh parameter plot.

## 8.6 Alternative platforms

The viewer technology of **E\_plot32** was also ported into a Linux and Irix version. These unix versions of the viewer have limited features and are continuously updated. If required obtain the most recent information from the author concerning the status of **ep1x** and **qmv**.

## Chapter 9

# Editing and modifying E\_Library files: MDFTOOLS

The MDFTOOLS are a series of checking, editing and conversion tools to help creating and maintaining E\_Library files. Some of them are batch files, some are small programs. A good knowledge of scripting or shell languages is advised in the preparation of E\_Library files as there is (so far) no graphical design utility for the E\_Library standard.

## 9.1 CHKMDF

### 9.1.1 Status

name of executable(s)	chkmdf
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	*.mdf
own filetypes	-

### 9.1.2 Syntax

Validity checking of an e\_lib mesh definition

Usage: `chkmdf filename`

`filename` : name of mesh definition file

All files should be specified without extension!

### 9.1.3 Overview

`chkmdf` is a tool that checks the validity of a mesh definition file. It displays mesh statistics and performs the following checks:

- Are there any duplicate nodes?
- Are there any invalid elements, i.e. elements in whose definition the same node number appears twice or more?
- Are there any duplicate elements? Here the node order is not taken into account.

## 9.2 CLEANUP

### 9.2.1 Status

name of executable(s)	cleanup   cleanup.exe
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf
own filetypes	-

### 9.2.2 Syntax

Remove unconnected nodes from a mesh.

Usage: cleanup input output

input : name of input mesh definition file

output : name of resulting mesh definition file

All files should be specified without extension!

### 9.2.3 Overview

When meshes are adaptively remeshed with **mgn** using the task **REMESH** all the nodes of the input coarse mesh are copied into the resulting file though they might not be used in any of the new triangles. These unconnected nodes cause problems when renumbering the mesh for bandwidth reduction.

**cleanup** removes all unconnected nodes from a mesh definition file. The program only works with meshes containing only **TRIANG1** elements. The topology of the mesh is not altered, only the numbering of points and the point numbers in the element definition. Boundary conditions and loads are also renumbered appropriately.

## 9.3 EMP2MPR

### 9.3.1 Status

name of executable(s)	emp2mpr
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf mpr
own filetypes	-

### 9.3.2 Syntax

Convert element mesh parameters to nodal mesh parameters  
by averaging all element mesh parameters connected to a node.

Usage: emp2mpr input output

input : name of input mesh file and mesh parameter file

output : name of output mesh parameter file

All files should be specified without extension!

### 9.3.3 Overview

emp2mpr converts an element mesh parameter file into a nodal mesh parameter file. The program requires, as an input, the mesh definition file and the element mesh parameter file, while the output is only a new nodal mesh parameter file. The program only works with meshes containing only TRIANGLE1 elements.

## 9.4 FLOOR

### 9.4.1 Status

name of executable(s)	floor   floor.exe
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	*.mdf *.mpr
own filetypes	-

### 9.4.2 Syntax

Limit the maximum and minimum values of the mesh parameter.

Usage: floor input output

input : name of input mesh file and mesh parameter file

output : name of output mesh parameter file

All files should be specified without extension!

### 9.4.3 Overview

floor is a little utility to alter mesh parameter files. Both nodal and element mesh parameter file are accepted. The file is parsed and minimum, maximum and average mesh parameter are printed on the screen. Then the user is asked to specify a new minimum and maximum parameter for this file.

## 9.5 MAKEDOM

### 9.5.1 Status

name of executable(s)	makedom
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf dom
own filetypes	-

### 9.5.2 Syntax

Create a domain decomposition file (\*.dom).

Usage: makedom [-v|-h|-n<domain>] mdf\_file  
-v - print version information  
-h - print this message  
-n<domain> - set domain number (default 1)

### 9.5.3 Overview

**makedom** creates a domain decomposition file which is compatible with the input mesh definition file. By default it generates a domain decomposition where all elements are in one domain. However it is also possible to specify that all elements should be assigned to a user defined subdomain. For example **makedom -n 2 mesh** will generate a domain decomposition file where all elements belong to subdomain two and the maximum number of subdomains is also equal to two.



## 9.6 MAKEMAT

### 9.6.1 Status

name of executable(s)	<code>makemat</code>
platform(s)	<code>irix   linux   win32 command line</code>
current version	<code>v1.0</code>
date	<code>July 2003</code>
release	<code>stable</code>
E_lib filetypes	<code>mdf mat</code>
own filetypes	<code>-</code>

### 9.6.2 Syntax

Creates material information and an example material file

Usage: `makemat input output`

input : name of input mesh file

output : name of output mesh and material file

All files should be specified without extension!

### 9.6.3 Overview

`makemat` creates material information for a mesh definition file and it also generates an example material file (`.mat`). The program only accepts meshes which contain a single element type. Moreover the acceptable element types are TRIANG1, TRIANG3 and LINK1.

## 9.7 MAKEMPR

### 9.7.1 Status

name of executable(s)	makempr
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf mpr
own filetypes	-

### 9.7.2 Syntax

Creates mesh parameter file (element or nodal)

Usage: makempr [-v|-h|-n<meshparam>|-e<meshparam>|-a] mdf\_file  
-v - print version information  
-h - print this message  
-n<meshparam> - set value of nodal mesh parameter  
-e<meshparam> - set value of element mesh parameter  
-a - automatic calculation of nodal mesh parameter (default)

### 9.7.3 Overview

**makempr** can create a nodal or element mesh parameter file. When the mesh parameter is explicitly specified on the command line all mesh parameters in the file will be the specified value. In the case of the **-a** option the program determines the minimum edge length at each point and this minimum edge length will become the nodal mesh parameter at each point.

## 9.8 MDF2K

### 9.8.1 Status

name of executable(s)	mdf2k
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	development
E_lib filetypes	mdf
own filetypes	-

### 9.8.2 Syntax

Converts and MDF file to an LS-DYNA keyword file describing only the geomtery of the mesh.

Usage: mdf2k file(.mdf) n > out.k  
n : the numbering of the nodes and elements starts at  $n*10000+1$

### 9.8.3 Overview

The mdf2k program converts a mesh geometry to an LS-DYNA keyword file. The created LS-DYNA file only describes the geometry, no other data is created. The numbering of the nodes and elements starts at  $n * 10000 + 1$ .

## 9.9 MDFMERGE

### 9.9.1 Status

name of executable(s)	mdfmerge
platform(s)	irix   linux
current version	v1.0
date	July 2003
release	development
E_lib filetypes	mdf
own filetypes	-

### 9.9.2 Syntax

Merges two mdf files.

Only merges geometry and domain decomposition,  
but no loads, bc, etc.

Usage: mdfmerge mesh1 mesh2 out [d]  
mesh1: first mesh definition file  
mesh2: second mesh definition file  
out: output mesh definition file  
d : when not zero it also merges decomposition data

### 9.9.3 Overview

The `mdfmerge` program merges the geometry of two mesh definition files. When the optional argument is specified and it is not zero then the program also merges the domain decomposition data in the mesh. However the program does not merge loads, boundary conditions, materials, etc.

## 9.10 MPR2EMP

### 9.10.1 Status

name of executable(s)	mpr2emp
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf mpr
own filetypes	-

### 9.10.2 Syntax

Convert nodal mesh parameters to element mesh parameters  
by averaging all nodal mesh parameters of an element.

Usage: mpr2emp input output

input : name of input mesh file and mesh parameter file

output : name of output mesh parameter file

All files should be specified without extension!

### 9.10.3 Overview

mpr2emp converts a nodal mesh parameter file into an element mesh parameter file. The program requires, as an input, the mesh definition file and the nodal mesh parameter file, while the output is only a new element mesh parameter file. The program only works with meshes containing only TRIANGLE1 elements.

## 9.11 RENUMBER

### 9.11.1 Status

name of executable(s)	renumber   renumber.exe
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	*.mdf
own filetypes	-

### 9.11.2 Syntax

Renumbers the mesh nodes to reduce the matrix bandwidth.

Usage: renumber input output

input : name of the input mesh definition file

output : name of the resulting mesh definition file

All files should be specified without extension!

### 9.11.3 Overview

**renumber** is a small program that changes the node indices of a mesh in order to minimize the bandwidth of the stiffness matrix when this one is build in any finite element package. It changes the node numbers accordingly in loads and boundary condition nodes.

### 9.11.4 Program limitations

- **renumber** works only for single element type **TRIANG1** or **QUAD1** meshes.
- Only the information within the mesh definition file is altered. Subdomain, stress, finite element error or mesh parameter information is left untouched.
- The program will crash if the mesh contains unconnected nodes. These are nodes which are not used in any of the element definitions. Use the **MDFTOOL cleanup** to remove unconnected nodes. **cleanup** is explained in Section 9.2

## 9.12 REV

### 9.12.1 Status

name of executable(s)	rev
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	development
E_lib filetypes	mdf
own filetypes	-

### 9.12.2 Syntax

Revolves quad elements around an axis to create blocks.  
The mesh must be in the x-y plane.

Usage: rev mesh n [-o angle] [-x -y]  
-o angle: Open revolution by angle [degree]  
-x : Revolution around x axis  
-y : Revolution around y axis

Defaults are rotation around x axis and rotation by 360 degrees  
to form a closed torus

### 9.12.3 Overview

The **rev** program revolves a mesh containing only QUAD1 elements around the  $x$  or the  $y$  axis to form block (hexahedral) elements. The program is capable to generate a closed torus or by using the **-o** option only a part of a torus. This later case is called open revolution. By default the program revolves the mesh around the  $x$  axis, but specifying the **-y** option it will revolve the mesh around the  $y$  axis. For example, the command **rev test 10 -o 90 -y** will generate the mesh as shown in Figure 9.1. The output of the program is written into the **rev.mdf** file.

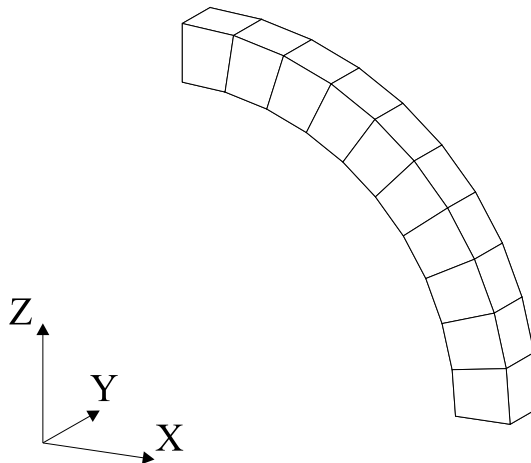


Figure 9.1: Result of **rev test 10 -o 90 -y**

## 9.13 SPHERE

### 9.13.1 Status

name of executable(s)	<b>sphere</b>
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	development
E_lib filetypes	mdf
own filetypes	-

### 9.13.2 Syntax

Generate a surface mesh of a sphere.

Usage: sphere n [-q|-t]  
n = number of subdivisions per side  
-q = generate QUAD1 elements  
-t = generate TRIANG1 elements

### 9.13.3 Overview

**sphere** generates a surface mesh of a sphere. The mesh may contain triangle or quad elements. The program initially generates a cube whose points are then displaced onto the surface of a sphere. The number of divisions specifies the number of divisions of an edge of the cube. The name of the output mesh is **triangs.mdf** when triangle elements are generated and **quads.mdf** when quad elements are generated.



## 9.14 VSPH

### 9.14.1 Status

name of executable(s)	vsph
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	development
E_lib filetypes	mdf
own filetypes	-

### 9.14.2 Syntax

Generate a volume mesh of a sphere.

Usage: vsph n  
n = number of subdivisions per side

### 9.14.3 Overview

**vsph** generates a volume mesh of a sphere. The output mesh will contain hexahedral elements. The program initially generates a cube whose points are then displaced to form a sphere. The number of divisions specifies the number of divisions of an edge of the cube. The name of the output mesh is **vsph.mdf**.

## 9.15 TRF

### 9.15.1 Status

name of executable(s)	trf
platform(s)	irix   linux   win32 command line
current version	v1.0
date	July 2003
release	stable
E_lib filetypes	mdf
own filetypes	trf

### 9.15.2 Syntax

Usage: trf mesh transformation [-N]

mesh : name of the mesh definition file  
transformation : name of the transformations file  
-N : also transform the geometric model

All files should be specified without extension!

### 9.15.3 Overview

**trf** is a small tool that allows you to perform geometrical operations on the coordinates array of a mesh. Mesh connectivity will not be touched. The output of the program is **mesh.trf.mdf** where **mesh** is the input filename.

The geometrical operations that **trf** can handle are:

- move: move every meshpoint over a fixed vector in space
- scale: scale all meshpoints in the three coordinate directions
- rotate: rotate meshpoints round the coordinate axes

### 9.15.4 Syntax of transformation file

The transformations file (**\*.trf**) has the following syntax:

```
NTRFS    n                /* the number of transformations defined below */

MOVE     dx dx dy          /* move operation over vector (dx,dy,dz) */
SCALE    sx sy sz          /* scale operation with scale factors si */
ROTATE   ax ay az          /* rotate operation around angles ai [-360,360] */
..
```

Moreover you can carry out as many operations as you want but some are equivalent:

```
MOVE     dx dy dz          ==      MOVE     dx 0  0
                                   MOVE     0  dy 0
                                   MOVE     0  0  dz

SCALE    sx sy sz          ==      SCALE    sx 1  1
                                   SCALE    1  sy 1
                                   SCALE    1  1  sz

ROTATE   ax ay az          ==      ROTATE   ax 0  0
                                   ROTATE   0  ay 0
                                   ROTATE   0  0  az
```

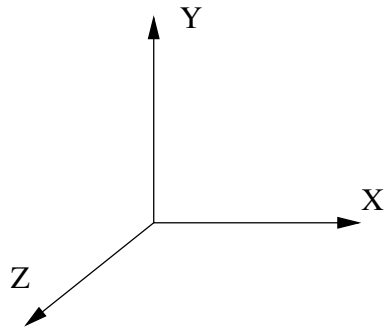


Figure 9.2: Coordinate system

Note that unlike moving or scaling the order of the rotations carried out does matter! So when you issue a `ROTATE ax ay az` where none of `ax`, `ay` and `az` are equal to zero, the rotation around the X axis will be carried out first, then the Y rotation and finally the Z rotation.

#### 9.15.5 Coordinate system

The coordinate system used is shown in Figure 9.2. The X-Y plane corresponds to the screen and the Z-axis is pointing towards you. The rotation angles are positive when they appear to be counter-clockwise when you look into the oriented axis. For example: `ROTATE 0 90 0` would transform the Z axis into the X axis.

# Bibliography

- [1] Chew, P.L., “Guaranteed-Quality Delaunay Triangulations”, Technical Report TR-89-983, Dept. of Computer Science, Cornell University, 1989.
- [2] Muylle, J., Iványi, P., Topping, B.H.V., “A new point creation scheme for uniform Delaunay triangulations”, *Engineering Computations, International Journal for Computer Aided Engineering and Software*, 19(6): 707–735, 2002.
- [3] PDA Engineering, PATRAN Division, 2975 Redhill Avenue, Costa Mesa, California, USA, *PATRAN Plus User Manual*, October 1990.
- [4] SECT Research Group, Heriot-Watt University, *E-lib User’s Guide*, 1999.