Technische Universität Chemnitz

Sonderforschungsbereich 393

Numerische Simulation auf massiv parallelen Rechnern

Thomas Apel Uwe Reichel

SPC-PM Po 3D V3.3 User's Manual

Preprint SFB393/99-06

Acknowledgement. The package *SPC-PM Po 3D* has been developed in the *Sonderforschungsbereich 393* of the *Technische Universität Chemnitz* under the supervision of A. Meyer and Th. Apel. Other main contributors are G. Globisch, D. Lohse, F. Milde, M. Pester, U. Reichel, and M. Theß.
Section 3.4 and Appendix A of this documentation were written together with F. Milde and G. Globisch, respectively, Section 3.6 was written by D. Lohse. The tests in Section 4 were partially carried out by A. Meyer and U. Reichel.

Preprint-Reihe des Chemnitzer SFB 393

SFB393/99-06 **Februar 1999**

Contents

Chapter 1

Introduction

At present time much effort is being spent in both developing and implementing parallel algorithms. The experimental package *SPC-PMPo 3D* is part of the ongoing research of the Chemnitz research group Scientific Parallel Computing (SPC), now part of SFB393, into finite element methods for problems over three dimensional domains. Special emphasis is paid to choose finite element meshes which exhibit an optimal order of the discretization error, to develop preconditioners for the arising finite element system based on domain decomposition and multilevel techniques, and to treat problems in complicated domains as they arise in practice.

The package *SPC-PMPo 3D* is based on a set of libraries which are still under development. They are documented in the Programmer's Manual [4] and in other separate papers [13, 17, 18, 19]. The aim of this User's Manual is to provide an overview over the program, its capabilities, its installation, and handling. Moreover, test examples are explained. This new release describes the changes from version 2 to version 3.3, the new standard file version 2.1, and new related tools.

In Version 3.x the program can solve the Poisson equation and the Lamé system of linear elasticity with in general mixed boundary conditions of Dirichlet and Neumann type, see Section 2.1. The domain $\Omega \subset \mathbb{R}^3$ can be a various curved bounded polyhedron, see [21]. The input is a coarse mesh, a description of the data and some control parameters. The program distributes the elements of the coarse mesh to the processors, refines the elements, generates the system of equations using linear or quadratic shape functions, solves this system and offers graphical tools to display the solution. Further, the behaviour of the algorithms can be monitored: arithmetic and communication time is measured, the discretization error is measured, different preconditioners can be compared. There exists special versions of *SPC-PMPo 3D* using a multigrid solver (M. Jung), having an error estimator (G. Kunert), or using Globisch-Nepomnyaschikh mesh transformation technique in the solver (G. Globisch). All these versions are adapted from the *SPC-PMPo 3D*-package but include major changes. Thats why only some of this features are part of the official distribution.

The version 3.x, documented here, is the last version with uniform mesh refinement. It is not developed further, only bugs will be fixed. The currently developed version 4 will include adaptive mesh refinement and dynamic load balancing.

The program has been developed for MIMD computers; it has been tested on Parsytec machines (GCPowerPlus–128 with Motorola Power PC601 processors and GCel–192 on transputer basis) and on workstation clusters using PVM. The special case of only one processor is included, that means the package can be compiled for single processor machines without any change in the source files. We point out that the implementation is based on a special data structure which allows that all components of the program run with almost optimal performance ($\mathcal{O}(N)$ or $\mathcal{O}(N \ln N)$).

In this documentation we use *slanted style* for really existing paths and filenames, *italic style* for program parameters, sans serif style to characterize buttons and menu items of programs with a graphical user interface, and `typewriter style` for the names of variables.

List of contributors

Dr. Thomas Apel	**email:** apel@mathematik.tu-chemnitz.de **contribution:** supervision, assembly, error assessment, communication, tests
Dr. Gerhard Globisch	**email:** globisch@mathematik.tu-chemnitz.de **contribution:** *PARMESH3D*, *renfindsun* (see 3.3)
Dr. Michael Jung	**email:** jung@mathematik.tu-chemnitz.de **contribution:** multigrid solver
Dr. Gerd Kunert	**email:** gkunert@mathematik.tu-chemnitz.de **contribution:** error estimator
Dag Lohse	**email:** lohse@mathematik.tu-chemnitz.de **contribution:** input files, *xbc* (see 3.2, 3.6)
Prof. Arnd Meyer	**email:** a.meyer@mathematik.tu-chemnitz.de **contribution:** supervision, solver, communication
Dr. Magdalene Meyer	**email:** m.meyer@mathematik.tu-chemnitz.de **contribution:** interface to *GRAPE*
Frank Milde	**email:** milde@physik.tu-chemnitz.de **contribution:** general frame of program, mesh refinement, tests, and *oldnetz* (see 3.4)
Dr. Matthias Pester	**email:** pester@mathematik.tu-chemnitz.de **contribution:** maintaining the libraries, time measurement, communication, handling of curved boundaries
Uwe Reichel	**email:** reichel@mathematik.tu-chemnitz.de **contribution:** domain decomposition via recursive spectral bisection, interface to *chaco*, tests, and *qnet* (see 3.5)
Michael Stübner	**email:** m.stuebner@mathematik.tu-chemnitz.de **contribution:** error norms
Michael Theß	**email:** thess@mathematik.tu-chemnitz.de **contribution:** solver

Chapter 2

Basic description

2.1 Mathematical background

Consider the Poisson problem in the notation

$$\begin{aligned} -\Delta u &= f && \text{in} && \Omega \subset \mathbb{R}^3, \\ u &= u_0 && \text{on} && \partial\Omega_1, \\ \frac{\partial u}{\partial n} &= g && \text{on} && \partial\Omega_2, \\ \frac{\partial u}{\partial n} &= 0 && \text{on} && \partial\Omega \setminus \partial\Omega_1 \setminus \partial\Omega_2, \end{aligned}$$

or the Lamé problem for $\underline{u} = (u^{(1)}, u^{(2)}, u^{(3)})^T$

$$\begin{aligned} -\mu\Delta\underline{u} + (\lambda+\mu) \text{ grad div } \underline{u} &= \underline{f} && \text{in} && \Omega \subset \mathbb{R}^3, \\ u^{(i)} &= u_0^{(i)} && \text{on} && \partial\Omega_1^{(i)}, \quad i = 1,2,3, \\ t^{(i)} &= g^{(i)} && \text{on} && \partial\Omega_2^{(i)}, \quad i = 1,2,3, \\ t^{(i)} &= 0 && \text{on} && \partial\Omega^{(i)} \setminus \partial\Omega_1^{(i)} \setminus \partial\Omega_2^{(i)}, \quad i = 1,2,3, \end{aligned}$$

where $\underline{t} = (t^{(1)}, t^{(2)}, t^{(3)})^T = S[u] \cdot \underline{n}$ is the normal stress, the stress tensor $S[u] = (s_{ij})_{i,j=1}^3$ is defined with $\underline{x} = (x^{(1)}, x^{(2)}, x^{(3)})^T$ by

$$s_{ij} = \mu \left[\frac{\partial u^{(i)}}{\partial x^{(j)}} + \frac{\partial u^{(j)}}{\partial x^{(i)}} \right] + \delta_{ij}\lambda\nabla \cdot \underline{u},$$

$\underline{n}$ is the outward normal, and δ_{ij} is the Kronecker delta. The domain $\Omega \subset \mathbb{R}^3$ must be bounded. In the present version curved boundaries can not be treated by the refinement procedure, thus Ω is restricted to be a polyhedron.

The boundary value problem is solved by a standard finite element method, using either tetrahedral or brick elements with linear or quadratic shape functions of the serendipity class, see Figure 2.1. The initial mesh must be generated outside *SPC-PMPo 3D*. After the file input it is distributed to the processors using a spectral bisection algorithm [24] or external information. That means, the domain Ω is decomposed in non-overlapping subdomains, the basis for our parallel algorithms. Then the elements are hierarchically refined to generate the final finite element mesh, for a description of the algorithm see Chapter 3 in [4].

The finite element stiffness matrix and the right hand side are generated locally in the subdomains by approximating the integrals using a quadrature rule, see Sections 4.1 and 4.2

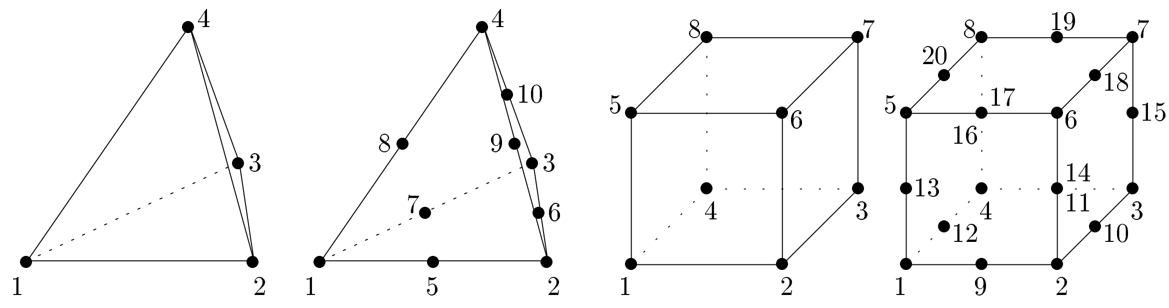


Figure 2.1: Finite elements implemented in *SPC-PMPo 3D*.

in [4]. The resulting system of equations is solved using a parallel version of the conjugate gradient method with Jacobi-, Yserentant- (hierarchical basis) or BPX preconditioning, which are described in [4, Chapter 5]. There exists also a special version using a multigrid method (M. Jung).

The post-processing includes a simple variant of error assessing. If in special test examples the exact solution of the problem is known then the error in L_2- and H^1-norms are calculated by numerical integration, additionally the error is measured in the discrete maximum norm, see [4, Subsection 4.4.1]. In general the exact solution of the problem is not available, thus we must rely on an error estimator. It exists a special version of *SPC-PMPo 3D* with an improved variant of the residual type error estimator, see [14]. At the moment this estimator is only available for linear tetrahedral elements and not part of the official release.

2.2 Installation

Provided AFS (the Andrew File System) is installed, any user can install the package by using the shell-script:

/afs/tu-chemnitz.de/home/urz/p/pester/bin/install3d `name_of_destdir [-version v.m]`

where `name_of_destdir` should be a name which does not yet exist, and the optional parameter `-version` provides the possibility to install different versions of the package. An explanation of various versions can be obtained by calling the script without parameters. For a quick start do the following:

1. Install the package in `name_of_destdir` by calling *install3d*.

2. If the installation of *SPC-PMPo 3D* is already on AFS no changes are necessary. Otherwise edit the Makefile in `name_of_destdir` and adjust the variables `$PARDEST` and `$PPCDEST` or more common the variable `$AFSDEST`; ensure that these directories exists at the corresponding machines. Moreover, it is useful to copy or link the directories *mesh3* and *mesh4* to the working directory of the remote machines or in AFS.

3. Choose the architecture you want to work with by calling one of the shell-scripts

   /usr/global/bin/setpvm,
   /usr/global/bin/setparix,
   or */usr/global/bin/setppc*.

   Some variables including `$archi` are now defined.

4. Call *make*.

Then, after successful compilation, the executable files *tet.*`$archi` (for tetrahedral meshes), *quad.*`$archi` (for cuboidal meshes), and *ggtet.*`$archi`/*ggquad.*`$archi` (both using Globisch/Nepomnyaschikh method for embedding unstructured meshes) should be contained in your directory. If defined, a copy should be in `$AFSDEST`, and, for `$archi=parix` and `$archi=ppc`, in the directories on the remote machines.

Before we are going to describe in some detail the use of the various files which were created during the installation we explain the diverse values of the variable `$archi`: It is used to distinguish the different architectures for which an executable file shall be compiled and linked, because the compiler, libraries and especially the communication routines are different.

- `$archi=SUN4` is set after calling *setpvm* on a SUN4 workstation. The executable files are *tet.SUN4* and *quad.SUN4*, they can run under pvm, or without the daemon of pvm, as single processor variant at a SUN workstation.
- `$archi=SUNMP` is set by calling *setpvm* on a Sun multiprocessor workstation.
- `$archi=HPPA` is set by calling *setpvm* on a HP workstation.
- `$archi=HPPAMP` is set by calling *setpvm* on a HP multiprocessor workstation.
- `$archi=SGI5` is set by calling *setpvm* on a SGI workstation running under Irix 5.x.
- `$archi=SGI64` is set by calling *setpvm* on a SGI workstation running under Irix 6.4.
- `$archi=LINUX` is set by calling *setpvm* on a PC or other machines running under Linux.
- `$archi=parix` is set by *setparix*. The executable files run at Parsytec transputer machines as the GCel–192 under the operating system PARIX.
- `$archi=ppc` is the setting after calling *setppc* which causes the compilation of an executable file for Parsytec machines based on the Motorola Power PC601 chip, as the Xplorer or the GCPowerPlus–128 under the operating system PARIX.
- `$archi=ppcmpi` could be set by the user after calling *setppc* which causes the compilation of an executable file for Parsytec machines based on the Motorola Power PC601 chip, as the Xplorer or the GCPowerPlus–128 using the message passing interface (MPI) instead of the PARIX routines.

In case of our latest version V3.2 after the installation a file structure as shown in Figure 2.2 is given. Except the files *pfem.f*, *bsp.f*, and *getdofs.f* the directory `name_of_destdir` contains no source files. All needed code is included in precompiled static link libraries which could be found in the *libs* directory separated by architectures. The default search path for this libraries is given in the Makefile by the variable `LIBDIR`. The *SPC-PMPo 3D* specific link libraries are:

libNetzA.a	libNetzT.a	libNetzQ.a
libSolve.a	libAssem.a	libElem3D.a
	libFehler.a.	

```
                     ┌ Bsp      — linked to /afs/tucz/project/sfb393/FEM/Bsp
                     ├ include  — linked to /afs/tucz/project/sfb393/FEM/include
                     ├ libs     — linked to /afs/tucz/project/sfb393/FEM/libs
                     ├ mylibs
                     ├ Makedir
name_of_destdir  ────┤ graph    — linked to /afs/tucz/project/sfb393/FEM/graph
                     ├ ass4     — linked to /afs/tucz/project/sfb393/FEM/ass4
                     ├ ass3     — linked to /afs/tucz/project/sfb393/FEM/ass3
                     ├ mesh3    — linked to /afs/tucz/project/sfb393/FEM/mesh3
                     └ mesh4    — linked to /afs/tucz/project/sfb393/FEM/mesh4
```

Figure 2.2: File structure after installation of *SPC-PMPo 3D*V3.2.

The way how to modify the library sources is described in section 2.3.

Sometimes it is necessary to describe problem data by function subroutines (right hand sides, exact solution if available). These routines are contained in the file *./bsp.f.* Our approach is to save example data in files *Bsp/bsp.*`example_name` and to copy the appropriate file to *./bsp.f.*

The directory *Makedir* contains some architecture specific files which are distinguished by the variable `$archi`, see also below. The file *variante.*`$archi` is included in the main source file and defines the length of a long vector for storing all vector data, its length must be adapted to the size of the memory of the machine to be used. The file *makefile.*`$archi` is included in the main *makefile* and contains specific options and directories which are machine dependent. The variable `$GRAF` can be set to `Graf` or `NoGraf`, thus the graphic libraries are linked or not, which results in a considerable difference in the size of the executable file.

Both, the handling of the *bsp.*`example_name` files and the setting of the `$GRAF` variable are provided by the shell-script *setup_ppc*. This script did all the needed changes automatically after the users choice.

A couple of meshes for tests are contained in the directories *mesh3* (tetrahedral meshes) and *mesh4* (cuboidal meshes): **.std.* The file structure is described in Section 3.2. These directories are linked to */afs/tucz/project/sfb393/FEM/mesh*[*3*,*4*], in order to prevent that the data files exist several times. In some cases there is a file `name`*.txt* which gives some information about the corresponding problem `name`*.std*.

Corresponding to the *mesh*[*3*,*4*] directories exists the directories *ass*[*3*,*4*] containing assignment informations for an optimal distribution of the mesh among the processors. These informations are considered, if `vertvar:3` is chosen in the control file, see 2.4. The assignment files are created by *chaco.sun* and must follow the naming convention described in [23]. The expected input for chaco is stored in the *graph* directory as `name`*.graph*. For new meshes a **.graph* file could be generated from the **.std* using the tool *std2graph.sun*.

All these AFS-directories are readable and executable for any user. M. Pester is administrating these directories and can include further AFS-users to a list of people who are allowed to add files in these directories.

The directory *mesh3* contains also a couple of files with the extension *.out*. These files were created with the mesh-generator *PARMESH3D*, see [11], and can be processed with the program *mesh3/renfindsun* on a SUN4 workstation. This program produces a file with the right data structure and with boundary conditions, which are set by a dialog with the user. Moreover, *renfindsun* can optionally re-numerate the nodes to minimize the bandwidth of the resulting stiffness matrix, see Section 3.3.

The program *mesh3/oldnetz* produces a restricted class of tetrahedral meshes, see Subsection 3.4. The program *mesh3/xbc* in an XView-application to view meshes and to set or to change boundary conditions interactively, see Section 3.6.

In the main directory `name_of_destdir` there is the main Makefile, the essential FORTRAN source files, some text files with helpful informations, the executables *f3_sgi* and *f3_sun* providing the graphical interface GRAPE (see A), the executable *chaco.sun* for pre-loadbalacing (see [23]), and the files *control.tet / control.quad* which are described in Section 2.4. The Makefile is used to compile source files, to create libraries, to link the executable file and to copy it to the appropriate machine (george.informatik or kain.hrz). The destination for the remote copy is defined by two variables `$PARDEST` and `$PPCDEST` in the Makefile, which should be adjusted by the user, see above. Note that it is possible to link only a special program by calling *make tet*, *make ggtet*, *make quad*, or *make ggquad*, respectively.

The Makefile can also be used to remove the libraries, tar-files, and executable files: *make clean* removes the target files for the current architecture, and *make CLEAN* removes them for all architectures. Only the files of the installation as well as user created files remain. The additional option *make tar* creates a archive with all sources, includes, Makefiles, and meshes. Some more information about the *Makefile* could be obtain by calling *make help*.

2.3 Modifying source files under the CVS

The whole source tree of *SPC-PMPo 3D* is under CVS control to keep the source management easy and transparent. As mentioned in section 2.2 the standard installation includes no source files except the main program *pfem.f*, the function subroutines *bsp.f*, and getdofs.f which influences the behaviour of the 2D-graphic.

To modify the sources of the libraries the source tree of each one had to be checked out of the CVS using the shell-command

cvs co `MODULNAME`

where `MODULNAME` is one of

NetzA, NetzT, NetzQ, Solve, Assem, Elem3D, Fehler.

Before performing this checkout the correct environment must be set by executing one of the scripts *setpvm*, *setppc* or *setparix*. The *cvs* command creates a directory `MODULNAME` containing the latest sources, the *LIBLISTE* and a *CVS* directory needed by the version control.

To link now the program with the local libraries instead of the global ones some modifications in the global Makefile are needed:

1. Refer to all local directories you want to process during compilation behind the variable `MYLIBS` at line 41.

2. Substitute $(ALIBDIR) with $(MYLIBDIR) in the lines 136 to 143 to link with the local version of a library.

After doing so, a simple *make* in the main directory should create the default 4 executables for the chosen architecture `$archi`. All makefiles work together with the *LIBLISTE* in their directory, a text file providing the source files to be included in the library. If a user wants to include additional source files, he/she should add it in the corresponding file *LIBLISTE*.

The CVS must also be used to redistribute modified sources. In the following a short description of possible transactions:

cvs co `MODULNAME`	creates a subdirectory `MODULNAME` including the latest sources, the *Makefile*, and *LIBLISTE* for the corresponding library in the subdirectory `MODULNAME`...
cvs update	updates the local source files with the global ones
cvs -n update	shows the changes since the last *cvs co* or *cvs update*
cvs diff -D today [files...]	shows differences between the local files and the last version checked in (also *-D yesterday* is possible, use *cvs diff -D today -D yesterday* to compare two archived versions)
cvs commit files...	checks in the named files
cvs release	propagates that no changes will be made in the next time

For further details on the CVS see `http://www.loria.fr/~molli/cvs/doc/cvs_toc.html`, `http://www.tu-chemnitz.de/~pester/cvs/cvs_exp.html`, or the manpages.

2.4 The files *control.tet* and *control.quad*

The mesh and the boundary conditions are described in files with the extension *.std*, see Subsection 3.2. Additionally, there is a couple of variables controlling the execution of the program. They are described together with their standard values in Table 2.1. Some of the variables contain numbers of quadrature formulas. They are given for the different types of elements in Tables 2.2 – 2.6. Note that the exactness of the quadrature formulas belongs to the master element, it may change for the actual element through transformation by multiplication with a nonconstant jacobian. The standard values have changed during the evolution of the program.

These standard values can be overwritten by defining other values in a file *control.tet* or *control.quad*, respectively. The lines in this file have the form

$$\begin{array}{rl} & \texttt{variable} : \texttt{value}, \\ \text{or} & \texttt{variable} : \texttt{value\_lin} \,/\, \texttt{value\_quad}. \end{array}$$

The ":" is relevant, `variable` must be written in lower case. There is no check of the usefulness of the `value`. Different values for the linear and the quadratic case can be given for all integer variables. This is especially useful for the quadrature rules and for `ndiag`. If a variable appears more than once in the file then the last value is taken.

Note that these files can be omitted, if only standard values shall be used. As an example consider the case that the user likes to change the stop criterion in the CG method to $\varepsilon < 10^{-10}$. He/she has two possibilities: Either one can change this during the execution, see the last paragraph in Section 2.5. Or he/she introduces the file *control.tet* (or *control.quad*) with one line

```
epsilon :  1.E-10
```

Variable	Standard values	Possible value	Description
`lin_quad`	1	[1,2]	kind of shape functions 1 : linear shape functions 2 : quadratic shape functions
`vertvar`	3	[1,2,3]	kind of coarse grid partitioning 1 : trivial partitioning 2 : partitioning via recursive spectral bisection 3 : read partitioning from file
`femakkvar`	3	[1,2,3]	there are three variants of accumulation of distributed data, see [2]
`loesvar`	5	[1...5]	choice of the preconditioner: 1 : Jacobi 2 : Yserentant without coarse grid solver 3 : Yserentant with coarse grid solver 4 : BPX without coarse grid solver 5 : BPX with coarse grid solver
`nint2ass`	14/31	[1...3][1...4]	number of the quadrature formula used for assembling Neumann boundary data. 1st digit : quadrilaterals, see Table 2.2 2nd digit : triangles, see Table 2.3
`nint3ass`	121/121	[1...5][1...5] [1...8]	number of quadrature formula for 3D elements used in the assembling. 1st digit : tetrahedra, see Table 2.4 2rd digit : hexahedra (bricks), see Table 2.6 3nd digit : pentahedra (triangular prisms), see Table 2.5
`nint2error`	14/31	[1...3][1...4]	as `nint2ass`, but used in the error estimator for the integration of the jump of the normal derivatives
`nint3error`	531/531	[1...5][1...5] [1...8]	as `nint3ass`, but used for the integration of 3D integrals in the error calculation
`ion`	1	integer	controls the amount of output of the program >0 : message after each `ion`-th CG-iteration ≤ 0 : no information about the iteration ≤ -1 : no startup screen and no problem info ≤ -2 : no information on numbers of coupling faces / edges / nodes ≤ -10 : no menus ≤ -11 : no input request messages
`iter`	200	integer>0	maximal number of iterations in the CG algorithm
`epsilon`	1.E−4	real>0	stop criterion for the CG (relative decrease of the norm of the residual)
`ndiag`	70	integer>0	upper estimate for the number of nonzero entries in any row of the stiffness matrix. If it is chosen too large, the program may suffer from lack of memory and if it is chosen too small, the number is iteratively increased ⇒ waste of time
`verf`	0	real∈[0,1]	mesh refinement parameter for a certain class of examples, see Subsection 4.1.7. 0 : no change of the mesh

Table 2.1: Variables in *control.tet / control.quad*.

Formula number	Number of points	Description	exact for x^iy^j with
1	1	midpoint (center of gravity)	$i,j \leq 1$
2	4	2x2 Gaussian points	$i,j \leq 3$
3	9	3x3 Gaussian points	$i,j \leq 5$

Table 2.2: Quadrature formulas for quadrilaterals.

Formula number	Number of points	Description	exact for x^iy^j with
1	1	center of gravity	$i,j \leq 1$
2	3	midpoints of the edges	$i,j \leq 2$
3	4	Gaussian points	$i,j \leq 3$
4	7	Gaussian points	$i,j \leq 5$

Table 2.3: Quadrature formulas for triangles.

Formula number	Number of points	Description	exact for $x^iy^jz^k$ with
1	1	center of gravity	$i+j+k \leq 1$
2	4	Gaussian points	$i+j+k \leq 2$
3	5	Gaussian points	$i+j+k \leq 3$
4	11	Gaussian points	$i+j+k \leq 4$
5	14	Gaussian points	$i+j+k \leq 5$

Table 2.4: Quadrature formulas for tetrahedra.

Formula	Number of	the formula is a cross product of the formulas		exact for
number	points	for triangle	for interval (z-direction)	$x^iy^jz^k$ with
1	$1 = 1 \cdot 1$	center of gravity	midpoint	$i+j \leq 1, k \leq 1$
2	$3 = 3 \cdot 1$	midpoints of edges	midpoint	$i+j \leq 2, k \leq 1$
3	$4 = 4 \cdot 1$	4 Gaussian points	midpoint	$i+j \leq 3, k \leq 1$
4	$6 = 3 \cdot 2$	midpoints of edges	2 Gaussian points	$i+j \leq 2, k \leq 3$
5	$8 = 4 \cdot 2$	4 Gaussian points	2 Gaussian points	$i+j \leq 3, k \leq 3$
6	$12 = 4 \cdot 3$	4 Gaussian points	3 Gaussian points	$i+j \leq 3, k \leq 5$
7	$14 = 7 \cdot 2$	7 Gaussian points	2 Gaussian points	$i+j \leq 5, k \leq 3$
8	$21 = 7 \cdot 3$	7 Gaussian points	3 Gaussian points	$i+j \leq 5, k \leq 5$

Table 2.5: Quadrature formulas for pentahedra.

Formula number	Number of points	Description	exact for $x^iy^jz^k$ with
1	1	midpoint (center of gravity)	$i,j,k \leq 1$
2	8	2x2x2 Gaussian points	$i,j,k \leq 3$
3	27	3x3x3 Gaussian points	$i,j,k \leq 5$
4	6	midpoints of the faces	$i+j+k \leq 3$
5	14	Irons formula	$i+j+k \leq 5$

Table 2.6: Quadrature formulas for hexahedra.

As an example, we display here the file *control.tet* as it is contained in the distribution of *SPC-PMPo 3D*:

```
! File zur Anpassung von Standardwerten fuer PFEM
!
! Kommentarzeilen sollten mit '!' beginnen
! Datenzeilen haben die Form:'schluesselwort:wert'
! Der Doppelpunkt ist wichtig
! Grosz-/Kleinschreibung ist signifikant
! Die Richtigkeit der Werte wird nicht ueberprueft
!
! Folgende Schluesselworte sind zulaessig, ihr Name entspricht der
! zu besetzenden Variable, deren Bedeutung und zulaessige Werte
! gehen aus dem Quelltext Netz/Tetraeder/control.f hervor.
!
!   lin_quad      1
!   vertvar       3
!   femakkvar     3
!   loesvar       5
!   nint2ass     14
!   nint3ass    311
!   nint2error   11
!   nint3error  311
!   ion           1
!   iter        200
!   epsilon       1E-4
!   ndiag        70
!   verf          0.
!
! Fuer alle Integer-Werte koennen zwei Werte fuer linear/quadratisch angegeben
! werden. Trennzeichen '/' erforderlich !!!
!
! Diese Liste musz bei Veraenderung von standard.f gegebenenfalls
! aktualisiert werden.
!
! Bei Mehrfachdefinition gilt die Letzte (Reihenfolge im File)
! Bei Nichtdefinition kommen die Werte aus control.f zur Anwendung

 loesvar : 5
 lin_quad : 1
 nint2ass : 34
 nint2error : 34
 nint3ass : 121/232
 nint3error : 531/531
 ion  : 10
 iter : 500
! epsilon : 1.e-6
 ndiag : 150/200
! verf : 0.5
 vertvar : 2
```

2.5 Output information/a typical run of the program

Output information can be classified into two groups:

- information that is printed in dependence of the variable `ion`, see Table 2.1,
- information that can be called by choosing a menu item.

We explain this information by following a typical run with `ion = 1`. After calling the program we get an introduction screen with the number of the version, the names of main authors, the length of the working vector, and the number of processors used. Then we get a copy of the control parameters and the input request for a problem file.

```
urei@kain:fem% xr8 tet.ppc
run  -a pp8 tet.ppc
run : Requesting network by calling nrm.
run : Creating 4 * 2 descriptor by calling mkdesc.
run : Starting D-Server at kain link 5.
  # ############################################################## #
  #                                                                #
  #  SSSS  PPPPP   CCCC      PPPPP  M       M   PPPPP        333  #
  # SS  SS PP  PP CC  CC     PP  PP MM     MM   PP  PP      33 33 #
  # SS     PP  PP CC         PP  PP MMM   MMM   PP  PP         33 #
  #  SSSS  PPPPP  CC     ### PPPPP  MM MMM MM   PPPPP OOO    333  #
  #     SS PP     CC         PP     MM  M  MM   PP   OO OO     33 #
  # SS  SS PP     CC  CC     PP     MM     MM   PP   OO OO  33 33 #
  #  SSSS  PP      CCCC      PP     MM     MM   PP    OOO    333  #
  #                                                                #
  # ############################################################## #
  #                                                                #
  #             Programm-Modul 3D-Potentialprobleme                #
  #                          Version:  3.30                        #
  #                                                                #
  #            TU Chemnitz, Sonderforschungsbereich 393            #
  #      Th.Apel, A.Meyer, M.Meyer, F.Milde, M.Pester, M.Thess     #
  #                                                                #
  #                  Fakultaet fuer Mathematik                     #
  #                                                                #
  #   16-MB-Variante (  3500000 Worte) - bis zu 1024 Prozessoren   #
  #                      in Benutzung:          8 Proz.            #
  #                      Gelinkt mit bsp.z                         #
  #                                                                #
  # ############################################################## #
        ****************************************************
        *          Belegung der Steuerparameter            *
        * (kann mittels File control.tet angepasst werden) *
        ****************************************************
        *                                                  *
        *  vertvar    =   2       lin_quad   =   1         *
        *  nen2d      =   3       nen3d      =   4         *
        *  femakkvar  =   3       loesvar    =   5         *
        *  nint2ass   =  34       nint3ass   = 121         *
        *  nint2error =  34       nint3error = 531         *
        *  iter       = 500       epsilon    = 0.10E-03    *
        *  ion        =  10       ndiag      =  70         *
        *                                                  *
        *  Verzeichnis fuer Netze :  mesh3/                *
        ****************************************************

Filename: cubus1
```

The file name is typed in, here *cubus1* (the input of a question mark generates a ls-command for the appropriate directory). Then we are asked for the number of refinement steps. It is also possible to escape by typing `-1` for a new mesh, `-2` to quit, or `-3` for new parameters.

```
GEWUENSCHTE ZAHL VON VERFEINERUNGSSCHRITTEN
   -1 = NEUES NETZ
   -2 = PROGRAMM BEENDEN
   -3 = NEUE PARAMETER
EINGABE : 2
```

After this we get information on the current state of the program and to the input mesh.

```
EINLESEN DER NETZDATEN AUS : mesh3/cubus1.std

Wuerfel, Kantenlaenge 10, oben/unten Dirichlet (Typ 2: u=z/10)
Gerhard Globisch
07.11.1994
Poisson-Gleichung
PARMESH, RENFINDSUN
3D
data_read done
No error
```

copy of the information of the input file (extension *.std*)

```
EINLESEN BEENDET, IER=    0

VERTEILUNG DER TETRAEDER DURCH REKURSIVE SPEKTRALBISEKTION.

Anzahl der Elemente in den Prozessoren:
   3   3   3   3   3   3   3   3
   1   1   2   2   1   1   2   2
   0   1   1   1   0   1   1   1
```

information on the progress of the recursive spectral bisection

```
 ==> in    2 Tetr.    2-mal Flaechen getauscht.
NETZ VERFEINERT VFS=1
NETZ VERFEINERT VFS=2

************************************************************
**                     AUSGABEMENUE                       **
************************************************************
*      0  : WEITER                                         *
*      4  : AUSGABE DER NETZDATEN                          *
*      5  : AUSGABE DER RANDKETTENDATEN                    *
*      8  : AUSGABE DER NETZDATEN IN STANDARDFILE          *
************************************************************
 -> EINGABE : 0
```

At this stage, the coarse mesh data are read in and distributed to the processors, the mesh is hierarchically refined. Now the possibility is given to print out some informations about the refined mesh and the chain fields, and to store the refined mesh as **.std* file. Choosing *0* we go on.

```
************************************************************
**                     AUSGABEMENUE                       **
************************************************************
*      0  : WEITER                                         *
*      1  : 3D-GRAFIK MIT GRAPE                            *
*      2  : 2D-GRAFIK SCHNITT/OBERFLAECHE                  *
*      4  : AUSGABE DER NETZDATEN                          *
*      5  : AUSGABE DER RANDKETTENDATEN                    *
************************************************************
 -> EINGABE : 0
```

Now the possibility to visualize the generated mesh either in 3D or 2D is given. We go on choosing *0* or just pressing $\boxed{\text{enter}}$.

```
START GENERIEREN/ASSEMBLIEREN
 Zeiten fuer Warten+Kommunikation [s]

 Prozessor
 log. /phys.    input :    in % :   output:    in % :   gesamt:
   0   0  0      0.00      0.00      0.00      0.00      0.00
   1   1  0      0.00      0.00      0.00      0.00      0.01
   2   3  0      0.00      0.00      0.00      0.00      0.01
   3   2  0      0.00      0.00      0.00      0.00      0.01
   4   0  1      0.00      0.00      0.00      0.00      0.00
   5   1  1      0.00      0.00      0.00      0.00      0.01
   6   3  1      0.00      0.00      0.00      0.00      0.01
   7   2  1      0.00      0.00      0.00      0.00      0.01
 reine Arithmetikzeit (max):      0.01
ASSEMBLIEREN BEENDET
 Coars-Grid-Matrix-Generation: Ier=          0
 Groesse der Matrix (VBZ)    :              30

* Probleminformationen (lokal Prozessor P):
 - globale Anzahl Crosspoints   :       8
 - Anzahl der Knoten (lokal)    :      35
 - davon:  lok. Crosspoints     :       4
           Summe der Randketten :      30
           Koppelknoten         :      34
           innere Knoten        :       1
 - Anzahl der Koppelkanten      :       6
 - Anzahl der Koppelflaechen    :       4

* Probleminformationen (  global ):
 - Anzahl der Prozessoren:      8
 - Anzahl der Knoten     :    125
 - davon :  Koppelknoten :    119
 -        interne Knoten :      6
 ->       Gesamtanzahl der Freiheitsgrade :    125

* Start der Simulation: Vorkonditionierung Nr. 5
                                            <enter>
```

time information for assembling process

information on the coarse grid matrix

information on data on processor 0

global information

Now the stiffness matrix as well as the coarse grid matrix are assembled. After an $\boxed{\text{enter}}$ the system of equation is solved, giving information on the convergence and on times for communication and arithmetics. After providing the possibility to save the graphic of time usage the program finally stops in the next menu.

```
  IT     (r,w)          (As,s)         ALFA           BETA         Eta
   1  3.850645E+01  5.718772E+01 -6.733342E-01  0.000000E+00   1.00
   2  2.147706E+01  1.432139E+02 -1.499650E-01  5.577524E-01   0.56
   3  3.863126E+00  2.177399E+01 -1.774193E-01  1.798722E-01   0.32
   4  5.871520E-01  2.966601E+00 -1.979208E-01  1.519888E-01   0.25
   5  1.974870E-01  1.042542E+00 -1.894283E-01  3.363473E-01   0.27
   6  4.499389E-02  2.041502E-01 -2.203960E-01  2.278322E-01   0.26
   7  7.271924E-03  2.561211E-02 -2.839252E-01  1.616203E-01   0.24
   8  1.445351E-03  5.632380E-03 -2.566146E-01  1.987577E-01   0.23
   9  2.842315E-04  9.748493E-04 -2.915645E-01  1.966522E-01   0.23
  10  9.355177E-05  2.828425E-04 -3.307558E-01  3.291394E-01   0.24
  11  2.334666E-05  1.145204E-04 -2.038646E-01  2.495588E-01   0.24
  12  2.409204E-06  1.059402E-05 -2.274118E-01  1.031926E-01   0.22
  13  4.041727E-07  1.625228E-06 -2.486868E-01  1.677619E-01   0.22
  14  6.576520E-08  1.625228E-06 -2.486868E-01  1.627156E-01   0.21
 IT=          14

 Zeiten fuer Warten+Kommunikation [s]

 Prozessor
 log. /phys.    input :    in % :   output:    in % :   gesamt:
    0   0  0      0.18     47.11      0.17     43.91      0.38
    1   1  0      0.26     69.20      0.09     25.98      0.38
    2   3  0      0.23     59.90      0.13     34.40      0.38
    3   2  0      0.29     78.01      0.06     17.08      0.38
    4   0  1      0.19     50.94      0.17     46.18      0.38
    5   1  1      0.29     76.30      0.07     18.96      0.38
    6   3  1      0.24     65.05      0.11     29.36      0.38
    7   2  1      0.29     77.64      0.06     17.08      0.38
 reine Arithmetikzeit (max):      0.03
            + HB2BPX (max):      0.00
 Filename for PS output:

************************************************************
**                      AUSGABEMENUE                      **
************************************************************
*      0  : WEITER                                         *
*      1  : 3D-GRAFIK MIT GRAPE                            *
*      2  : 2D-GRAFIK SCHNITT/OBERFLAECHE                  *
*      4  : AUSGABE DER NETZDATEN                          *
*      5  : AUSGABE DER RANDKETTENDATEN                    *
*      6  : AUSGABE DER LOESUNG                            *
*      7  : AUSGABE VON FEHLERNORMEN                       *
************************************************************
 -> EINGABE : 6
```

With item 0 we exit the menu, with item 1 we are asked for the host name for displaying, then we start the data transfer to the interactive graphics package *GRAPE*, see [19], provided the program *f3_sun* or *f3_sgi* runs at the own workstation (host name). In this case a control and a graphics window will appear in order to display the grid and / or solution. One solution (starting with the first degree of freedom) can appear at one time. Using the control window we can make visible the other degrees of freedom by pressing the buttons with the names of the corresponding functions. Pressing the continue button in the control window the program on the parallel computer is forced to continue, for example to compute a new solution. During this time the graphical program may go on displaying the old data until the FE3D neu button is pressed to receive new data from the parallel computer (again via menu item 1). With Exit we can finish the graphics program.

On item 2 the window of our 2D interactive graphics interface is opened and asked for the kind of visualization. Possible choices are the solution plot on the surface, the solution plot of a chosen plane of intersection, or quit. In the first two cases the user is prompted for some information on the perspective of the plot and related things. For more details how to operate with the 2D interface see [20].

The choice of item 4 leads to the output of the local mesh data to files

netzred.`number_of_processor`.*dat*

(one file per processor). The same is done by item 4 with the coordinates of the nodes stored in *Kette*s, for the term *Kette* see [2]; they are stored in files *kettinf.P*`number_of_processor`*.dat*.

With menu item 6 we get a table of values into a file *loesung.dat* or on screen. The table includes the local node numbers, their coordinates, the calculated solution, the solution using the function u from *bsp.f* (probably the previously known exact solution), and their difference for each processor, see the printout.

```
AUSGABE DER WERTETABELLE DER LOESUNG

AUSGABE IN FILE LOESUNG.DAT (J/N) :n

PROZESSOR   0: NUMNP=    0

PROZESSOR   1: NUMNP=   35
| NR  |    X     |    Y     |    Z     |BER. LOESUNG| EXAKTE LSG.| DIFFERENZ  |
|    1|  10.000|   0.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|    2|  10.000|  10.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|    3|   0.000|  10.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|    4|  10.000|   0.000|   0.000|     0.00000|     0.00000| 0.00000D+00|
|    5|  10.000|   0.000|   7.500|     0.74999|     0.75000|-0.77712D-05|
|    6|  10.000|   0.000|   5.000|     0.50000|     0.50000|-0.20307D-05|
|    7|  10.000|   0.000|   2.500|     0.24999|     0.25000|-0.10760D-04|
|    8|  10.000|   2.500|  10.000|     1.00000|     1.00000| 0.00000D+00|
|    9|  10.000|   5.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   10|  10.000|   7.500|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   11|   7.500|   2.500|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   12|   5.000|   5.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   13|   2.500|   7.500|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   14|  10.000|   7.500|   7.500|     0.75000|     0.75000|-0.14338D-05|
|   15|  10.000|   5.000|   5.000|     0.50000|     0.50000| 0.26510D-05|
|   16|  10.000|   2.500|   2.500|     0.24999|     0.25000|-0.59909D-05|
|   17|   7.500|  10.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   18|   5.000|  10.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   19|   2.500|  10.000|  10.000|     1.00000|     1.00000| 0.00000D+00|
|   20|   2.500|   7.500|   7.500|     0.75001|     0.75000| 0.59876D-05|
N - NAECHSTER PROZESSOR; A - ABBRUCH; SONST - WEITER (NUMNP=    35) ? : a
```

If the output is on screen it stops after displaying 20 nodes and the user is prompted how to proceed. You can switch to the next processor by pressing N, cancel output with A, or proceed displaying with any other key.

Menu item 7 gives the results of local and global error calculations/estimations.

```
************************************************************
**                        AUSGABEMENUE                    **
************************************************************
*      0  : WEITER                                         *
*      1  : 3D-GRAFIK MIT GRAPE                            *
*      2  : 2D-GRAFIK SCHNITT/OBERFLAECHE                  *
*      4  : AUSGABE DER NETZDATEN                          *
*      5  : AUSGABE DER RANDKETTENDATEN                    *
*      6  : AUSGABE DER LOESUNG                            *
*      7  : AUSGABE VON FEHLERNORMEN                       *
************************************************************
 -> EINGABE : 7

AUSGABE VON FEHLERNORMEN (LOKAL):
|PROZ|    MAX-NORM|     L2-NORM|     H1-NORM|
|   0| 0.00000E+00| 0.00000E+00| 0.00000E+00|
|   1| 0.10760E-04| 0.32158E-04| 0.40438E-04|
|   2| 0.24787E-04| 0.72468E-04| 0.69347E-04|
|   3| 0.24787E-04| 0.82577E-04| 0.60800E-04|
|   4| 0.00000E+00| 0.00000E+00| 0.00000E+00|
|   5| 0.73617E-05| 0.39079E-04| 0.35490E-04|
|   6| 0.78932E-05| 0.26933E-04| 0.28801E-04|
|   7| 0.24787E-04| 0.52470E-04| 0.63799E-04|

AUSGABE VON FEHLERNORMEN (GLOBAL):
|    MAX-NORM|     L2-NORM|     H1-NORM|
| 0.24787E-04| 0.13457E-03| 0.12767E-03|

************************************************************
**                       AUSGABEMENUE                     **
************************************************************
*      0  : WEITER                                         *
*      1  : 3D-GRAFIK MIT GRAPE                            *
*      2  : 2D-GRAFIK SCHNITT/OBERFLAECHE                  *
*      4  : AUSGABE DER NETZDATEN                          *
*      5  : AUSGABE DER RANDKETTENDATEN                    *
*      6  : AUSGABE DER LOESUNG                            *
*      7  : AUSGABE VON FEHLERNORMEN                       *
************************************************************
 -> EINGABE : 0

GEWUENSCHTE ZAHL VON VERFEINERUNGSSCHRITTEN
   -1 = NEUES NETZ
   -2 = PROGRAMM BEENDEN
   -3 = NEUE PARAMETER
EINGABE : -2


************************************************************
*                        PROGRAMMENDE                      *
************************************************************

run : Returning network by calling nrm.
run : Terminating with result = 0.
urei@kain:fem%
```

The choice of item 0 led to the main menu, see above.

Some of the information is also written in the files *fort.08* and *fort.09*, but this is only for test reasons and permanently changing. Furthermore, we note that at the stage

```
* Start der Simulation: Vorkonditionierung Nr. 5
                                            <enter>
```

some special letters can be entered to control the PCCG iteration process

`v`	for a change of the preconditioner (`loesvar`),
`i`	for a change of the maximal number of iterations (`iter`),
`e`	for a change of the stop tolerance (`epsilon`),
`d`	for a scaling of the coarse grid matrix,
`z`	for a change of the variable `ion`.

These corrections are valid only during the following CG iteration and do not overwrite the standard values of these variables, see Subsection 2.4. An exception is `ion`.

Chapter 3

Meshes and boundary conditions

3.1 General remarks

The program *SPC-PMPo 3D* has not been designed to generate coarse meshes or boundary data. It is assumed that these data are prepared before and stored in a file with extension *.std*. The structure of such files is described in [16]; we summarize it briefly in Section 3.2.

There are several ways to create such an input file. For the easiest domains one can just create it with an editor. Moreover, several mesh generators have been programmed in the past. Because they use different file structures there have been developed adapter programs, see Appendix A. In Section 3.3 we describe the adapter program *mesh3/renfindsun* (author G. Globisch) which writes files of the structure appropriate for *SPC-PMPo 3D*. This program has two additional features: re-numeration of the nodes to minimize the profile of the coarse grid matrix and an interactive definition of boundary conditions.

For five classes of meshes which were used already with the sequential program *FEM-PS3D*, there is the tool *mesh3/oldnetz* (author F. Milde) which is described in Section 3.4.

To generate hexahedral meshes exists the tool *mesh4/qnet* (author U. Reichel). In Section 3.5 we describe its functionality and the mesh classes it provides.

In Section 3.6 we introduce the tool *mesh3/xbc* (author D. Lohse) which is an XView-application to visualize meshes and boundary conditions which are stored in **.std* files. Furthermore, it is possible to (re-)define boundary conditions with this tool.

Another tool written by D. Lohse is the program *geo_conv*, which is explained in Section 3.7. It gives the possibility to move a given mesh in space (translation, rotation), and to combine two or more meshes to a new one. So it is relative easy to construct complex forms from simple geometric solids.

3.2 Structure of the input file **.std*

The input file is a (7-bit) ASCII-file which contains data lines, control lines and key word lines (both starting with a "`#`"), and comment lines (starting with "`##`"), see for example *Cubus1.std* in Table 3.3.

The file starts with a control line defining the version

```
#VERSION: 2.1
```

in order to circumvent incompatibilities when the data structure is extended or changed. The latest version of the standard file format is 2.1. described in citelohse:98a. The file input is stopped either by reaching the end of the file or the statement

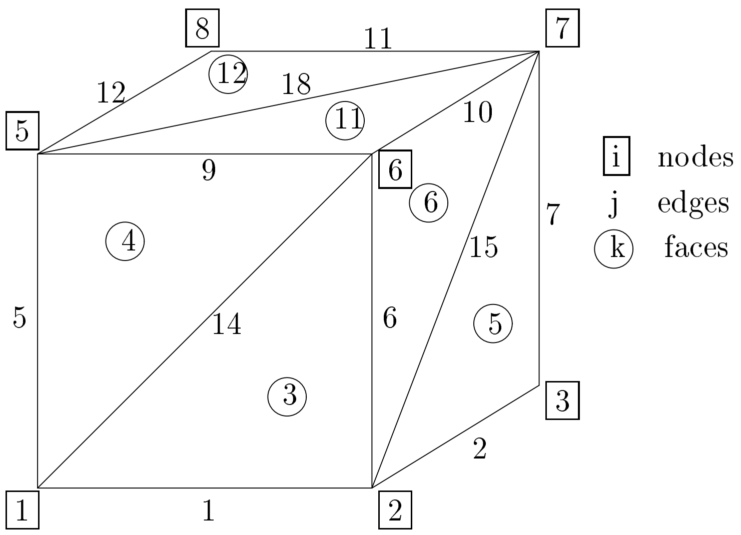


Figure 3.1: View of the cube, which is described in *Cubus1.std*.

```
#END_OF_DATA
```

After the `#VERSION` statement there may be *optional* information statements, see Table 3.1 for a selection. Moreover, it is possible to redefine some internal array dimensions via such statements, see [15, 16]. The information part and the data part of the file are separated by a `#HEADER` statement. It determines the maximal number of data lines of the different types.

`#HEADER:` *count*
vertices edges faces solids regions \
dirfaces neumfaces materials facegeoms

where *count* (4–9) is the number parameters in the header. Note that the backslash marks a continuation of the line, *dirfaces* and *neumfaces* means the number of faces with Dirichlet and Neumann data, respectively.

The actual data blocks follow now in any permutation. A block consists of a key word line and a number of data lines. Note that the key word line may contain an integer. The key words and the structure of the data lines is summarized in Table 3.2, for a full explanation see [15, 16].

The file *Cubus1.std* (see Table 3.3) may serve as an introductory example which describes the partition of a cube $\Omega = [0,1]^3$ into 6 congruent tetrahedra, compare Table 3.4 and Figure 3.1 for the understanding of the topology. Here, no `#REGION` is defined; a region name is useful to join some objects to one area. So defined regions could get special properties in the program. If undefined, all elements belong to one region with the name 1. Note that the meaning of the `#REGION` block has substantially changed from version 1 to version 2 standard files, see [15, 16].

Another essential change in version 2 of the standard file is the block `#FACE_GEO` referred by the face type. This feature provides a wide range of possibilities for easily creating meshes with curved boundaries. For a detailed explanation we refer to [21].

Statement	Description
`#DESCRIPTION` : string	description of the file for cataloguing
`#DATE` : date	date of creation of the file
`#USER` : username	Login name of the creator of the file
`#PROGRAM` : name	name of the creating program
`#DIMENSION` : 3D	geometrical dimension of the problem, here only 3D useful
`#EQN_TYPE` : string	problem type, defines e.g. the meaning of the material data
`#DEG_OF_FREE` : integer	number of degrees of freedom (standard: 5)

Table 3.1: Selection of information statements in the input file

Key word line	Description
`#VERTEX`:	*name* *xcoord* *ycoord* *zcoord*
	I R R R
`#EDGE`:	*name* *type* *start* *end* {[*middle*] | [*pointer* *data*]}
	I I I I I I arbitrary
	type = 1: ignored
`#FACE`:	*name* *type* *n* *edge*_1 … *edge*_n [*pointer* *data*]
	I I I I I I arbitrary
	type=1/0: plain face
	type>1: pointer to a face geometrie in `#FACE_GEO`:
`#SOLID`:	*name* *type* *n* *face*_1 … *face*_n [*pointer* *data*]
	I I I I I I arbitrary
	type = 1/0: standard material
	type > 1: pointer to material name in `#MATERIAL`:
`#REGION`:	*name* *type* *n* *solid*_1 … *solid*_n
	I I I I I
	type = 1 ignored
`#DIRICHLET`:	*name*
	I
	type *data* [*pointer* *data*]} (one line per d.o.f.)
	I R I arbitrary
	type = 0: no Dirichlet condition for this d.o.f.
	type = 1: constant value, given in *data*
	type = 2: boundary values are given by a linear function in global coordinates $u_0(x,y,z) = data[1] \cdot x + data[2] \cdot y + data[3] \cdot z + data[4]$.
	type ≥ 100: function pointer, boundary values are taken from function subroutine in *bsp.f*
`#NEUMANN`:	in analogy to `#DIRICHLET`
`#MATERIAL`:	*name* *n* *data*_1 … *data*_n
	I I R R
`#FACE_GEO`:	*name* *type_of_geo* *n* *data*_1 … *data*_n
	I I I R R
	for possible values of *type_of_geo* see [21]

Table 3.2: Structure of the data blocks in the input file

```
#VERSION: 1.0
#DESCRIPTION: 6 kongruente Tetraeder
#DATE: 13.7.1995
#USER: Thomas Apel
#DIMENSION: 3D
#EQN_TYPE: Poisson
#DEG_OF_FREE: 1
#HEADER:   8
   8  19  18  6  0  4  0  0
#VERTEX:   8
   1  0.  0.  0.
   2  1.  0.  0.
   3  1.  1.  0.
   4  0.  1.  0.
   5  0.  0.  1.
   6  1.  0.  1.
   7  1.  1.  1.
   8  0.  1.  1.
#EDGE:   19
   1  1  1  2
   2  1  2  3
   3  1  3  4
   4  1  4  1
   5  1  1  5
   6  1  2  6
   7  1  3  7
   8  1  4  8
   9  1  5  6
  10  1  6  7
  11  1  7  8
  12  1  8  5
  13  1  1  3
  14  1  1  6
  15  1  2  7
  16  1  4  7
  17  1  1  8
  18  1  5  7
  19  1  1  7
```

```
#FACE:  18
   1  1  3   1   2  13
   2  1  3   3   4  13
   3  1  3   1   6  14
   4  1  3   9   5  14
   5  1  3   2   7  15
   6  1  3  10   6  15
   7  1  3   3   7  16
   8  1  3  11   8  16
   9  1  3   4   8  17
  10  1  3  12   5  17
  11  1  3   9  10  18
  12  1  3  11  12  18
  13  1  3  13   7  19
  14  1  3  18   5  19
  15  1  3   1  15  19
  16  1  3  14  10  19
  17  1  3   4  16  19
  18  1  3  11  17  19
#SOLID:  6
   1  1  4   1   5  13  15
   2  1  4  15   3   6  16
   3  1  4  16   4  11  14
   4  1  4   2   7  13  17
   5  1  4  17   9   8  18
   6  1  4  18  10  12  14
#DIRICHLET:   4
   1
       1   0.0
   2
       1   0.0
   11
       1   1.0
   12
       1   1.0
#END_OF_DATA
```

Table 3.3: The file *Cubus1.std*

Tetrahedron	Names of faces				Names of edges						Names of nodes			
1	1	5	13	15	1	2	13	7	15	19	1	2	3	7
2	15	3	6	16	1	15	19	6	14	10	1	2	6	7
3	16	4	11	14	10	14	19	5	9	18	1	5	6	7
4	2	7	13	17	3	4	13	7	16	19	1	4	3	7
5	17	9	8	18	4	16	19	8	17	11	1	4	8	7
6	18	10	12	14	11	17	19	5	12	18	1	5	8	7

Table 3.4: Names of faces, edges, and nodes of the 6 tetrahedra in *Cubus1.std*.

3.3 The tools *renfindsun* and *renedgsun*

Because of the importance of the files `file`.*std* for the package *SPC-PMPo 3D* the program *renfindsun* shall be described in more detail here. The program *renfindsun* converts the ASCII output file **.out* (see [10, 22] for a description of the structure of the file) of the parallel mesh generator *parmesh3d* (tetrahedral meshes) into the file **.std*, see 3.2 for this data structure. This means a change of the node related data structure into the edge/face structure. Note that *renfindsun* may also store the output data as `file`.*edg*. This is another file type for the edge related data structure, see [10]. It organized similarly to `file`.*out*.

This transfer includes the setting of boundary conditions (type and data) to the boundary faces by a dialog with the user. There are two possibilities, namely face by face or by defining face groups. The second variant is described in 3.4.2. The first possibility consists in the face-wise screen output of the coordinates of the three nodes and in prompting for the description of the related boundary condition for each degree of freedom. For both methods, this information consists of the kind (Dirichlet, Neumann, 3rd kind/Robin), the type, and eventually some real values, see Table 3.2. The mesh and the boundary conditions can be visualized by means of the program *xbc*, which is also capable to impose/change boundary conditions, see Section 3.6.

Moreover, the user can determine whether he/she wants to re-numerate the nodal points of the mesh in order to reduce the bandwidth/profile of the corresponding matrix (adjacency matrix to the edge graph). The corresponding algorithm is implemented to be an efficient combination of minimal degree ordering and nested dissection, see [9]. The numerical expense is $\mathcal{O}(N^{\frac{3}{2}})$ for two-dimensional meshes, where N denotes the number of nodes; in the three-dimensional case we were not able to prove an estimate. Note that files which have already the structure `file`.*std* can be re-numerated by the program *renedgsun*, even a repeated application of *renedgsun* can further reduce the bandwidth/profile.

The mesh generator *parmesh3d* can also construct meshes consisting of tetrahedra having curved boundaries. The corresponding internal data structure is given in [10]. But to date there is no agreement about the file structure for curved elements. The corresponding extension of the related programs will be done in the future.

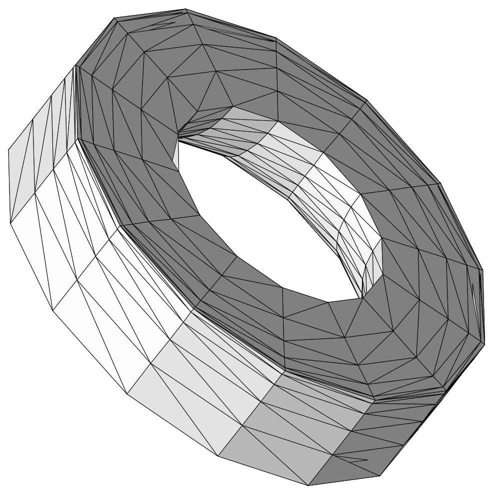
3.4 Generation of meshes via *oldnetz*

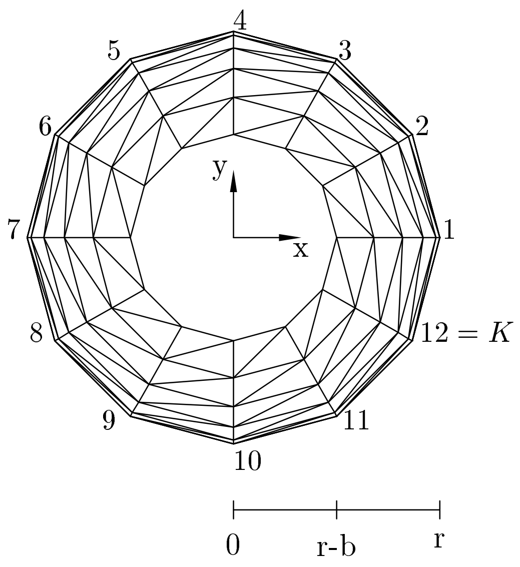
3.4.1 Mesh generation

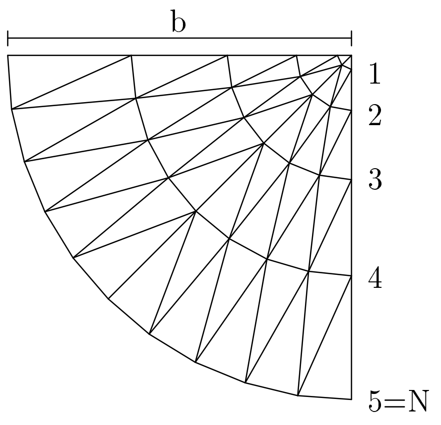
The program *oldnetz* is compiled for a SUN4 workstation and can be used interactively to generate 5 different families of meshes, to describe the boundary conditions, and to store this information in a file `file`.*std* with the data structure as given in Section 3.2. The user is requested to enter the number of the family and the corresponding parameters, for a short description see Figures 3.2 – 3.6. For refining meshes using the parameter μ see [3, 5, 6, 7].

3.4.2 Setting boundary conditions

If a mesh contains not only a few elements then it is boring to enter the boundary conditions face by face. Thus a dialog with the user was programmed to define groups of faces and to enter the type and the data of the boundary condition once for the whole group. This procedure is repeated for each degree of freedom.



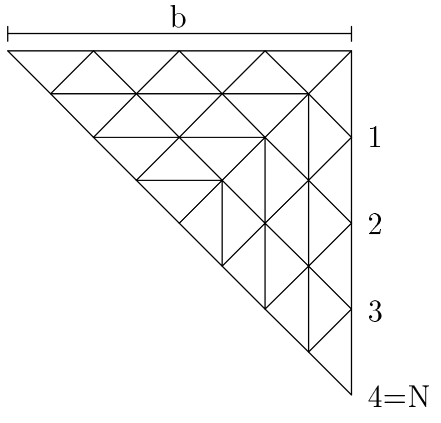




Input parameter:

μ	mesh refinement parameter (here $\mu = 0.4$)
N	number of circular arcs
K	number of nodes at the edge
R	outer radius (radius of the middle circle of the torus)
A	z-coordinate of this middle circle
B	radius of of the sector in the cross section

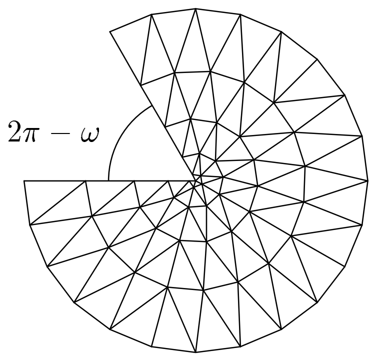
Figure 3.2: Description of the 1st family, a 90°-sector of a torus: perspective view, top view, and cross section.



Input parameter:

μ	mesh refinement parameter (here $\mu = 1$)
N	number of slices in the cross section (see figure)
K	number of nodes at the edge
R	outer radius (radius of the middle circle of the torus)
A	z-coordinate of this middle circle
B	length of the cathete in the cross section

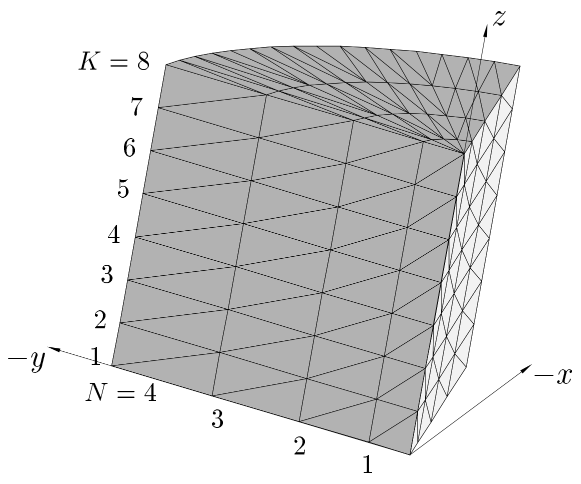
Figure 3.3: 2nd family: as before but with another cross section.



Input parameter:

ω	internal angle of the sector
μ	mesh refinement parameter (here $\mu = 0.4$)
N	number of circular arcs
K	number of nodes at the edge
R	radius of the circular edge (the middle circle of the torus)
A	z-coordinate of this middle circle
B	radius of of the sector in the cross section
IS	number of sectors for mesh generation (here 4)

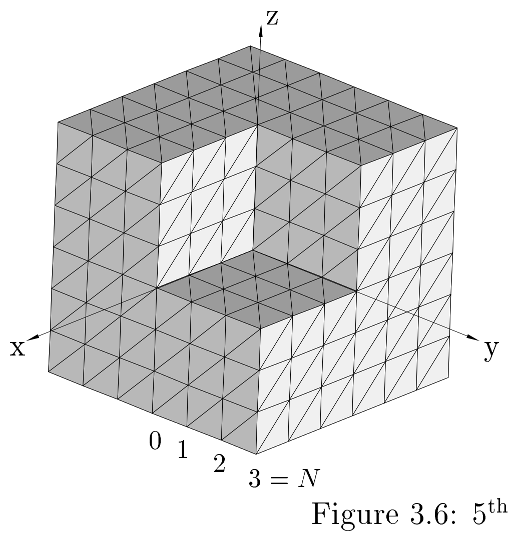
Figure 3.4: 3rd family: sector of a torus with arbitrary internal angle ω.



Input parameter:

ω	internal angle of the sector
μ	mesh refinement parameter (here $\mu = 0.6$)
N	number of circular arcs
K	number of nodes at the edge
R	radius of the cylinder
A	height of the cylinder
IS	number of sectors for mesh generation (here 4)

Figure 3.5: 4th family: sector of a cylinder with arbitrary internal angle ω.



Input parameter:

N	reciprocal value of the mesh size

Figure 3.6: 5th family: Fichera corner.

To define the group one enters conditions of the form

$$\begin{aligned} x^- &< x < x^+, \\ y^- &< y < y^+, \\ z^- &< z < z^+, \\ r_x^- &< r_x < r_x^+, \\ r_y^- &< r_y < r_y^+, \\ r_z^- &< r_z < r_z^+. \end{aligned}$$

In this way all nodes are marked which satisfy all the conditions given. The group consists of all faces which have only marked nodes. Note the special case when no condition is entered; then all boundary faces are in the group.

After defining the group of faces the user is asked for

- the kind of boundary condition (`1` - Dirichlet, `2` - Neumann[1]),
- the type and the data for the boundary conditions, see Table 3.2 for the explanation.

Then the next group of boundary faces can be defined or one may exit this menu. In the second case one is asked for a filename to store the data and the program terminates.

Note that faces can be included in groups several times, then the boundary condition is always redefined for these faces. This feature can be use for correcting errors or to enter complicated boundary data. For example, if all faces but one have Dirichlet conditions, one can first enter the Dirichlet condition for all faces and then redefine the exceptional face.

3.5 Mesh generation with *qnet*

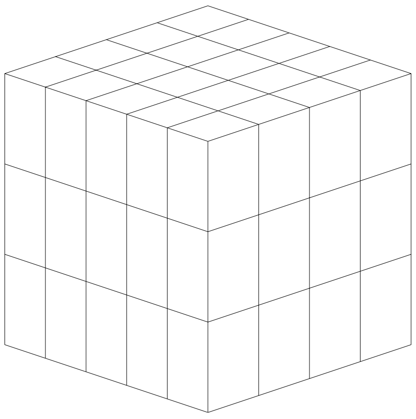
The program *qnet* is compiled for SUN4 and HPPA workstations and provides the interactive generation of 6 kinds of hexahedral meshes. The generated meshes can be stored in **.std* file with the structure as given in Section 3.2. After calling the program it provides a menu and the user is prompted for the mesh type. Appropriate choices are shown in Figures 3.7(a) - 3.7(f). In addition to this 6 kinds of meshes *qnet* provides the possibility to create a general cutting area, which means it can create any mesh which could be obtained by cutting elements from a cube mesh.

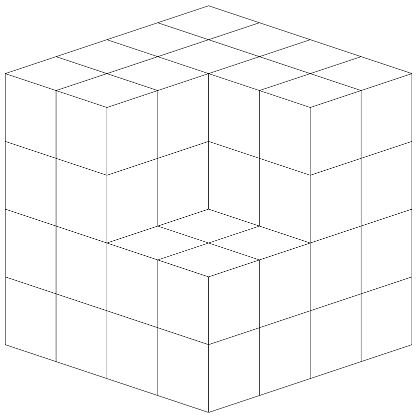
After a mesh type was chosen, *qnet* asks for the x-, y-, z-Dimension and for the number of sections in each direction. In case of the cube with a split it asks further for the angle of the split.

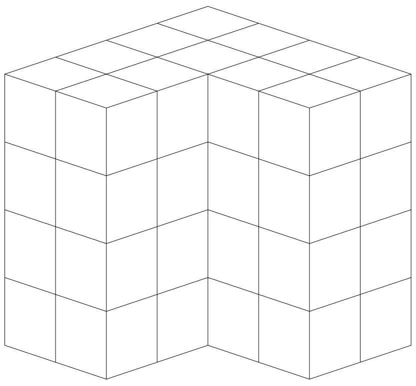
If the general cutting area was chosen first a cube is generated and the user is asked if elements should be caught. Saying yes the user must determine a range of elements to be caught. This is done by giving the from and to section of each direction. As an example, you have created a cube with three sections in each direction and now you want to cut a hole in y-direction. The correct choice is to cut from 2 to 2 in x-, from 1 to 3 in y-, and from 2 to 2 in z-direction. After each cut the user will be asked if he want to do another cut. If not, the program goes on offering a plane move. For the meaning of this feature another example. Say you have generated a cube $\Omega = [0,1]^3$ with 2 sections in each direction. So you have a cube consisting of 9 planes. The cube planes are:

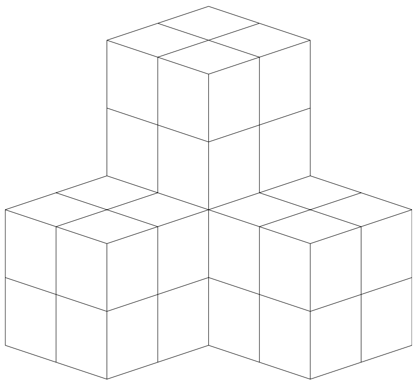
$$\begin{aligned} &x, y \in [0,1],\ z \in \{0, 0.5, 1\}, \\ &x, z \in [0,1],\ y \in \{0, 0.5, 1\}, \text{ and} \\ &y, z \in [0,1],\ x \in \{0, 0.5, 1\}. \end{aligned}$$

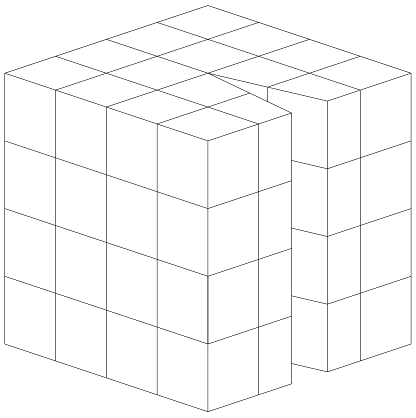
[1]The menu offers also boundary condition of 3rd kind, but *SPC-PMPo 3D* can not treat them yet.


(a) ordinary cube


(b) Fichera corner


(c) piano


(d) tripod


(e) cube with a slit

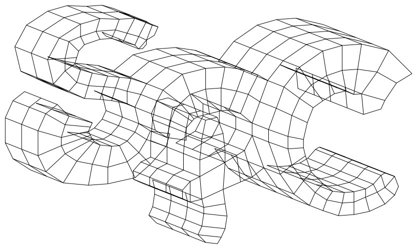

(f) SPC mesh

Figure 3.7: Mesh families offered by *qnet*.

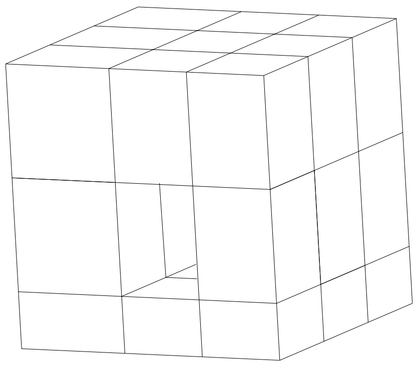


Figure 3.8: Example of a general cutting area with a hole in y-direction and moved planes.

Each of this planes could now be moved along an axis. After choosing the plane orientation *qnet* offers all possible planes in a menu and you can choose one and change its height. This could be continued until you cancel. Then the mesh will be saved. As an example see Figure 3.8.

If you want to create a so called SPC mesh you will be asked for the kind (1 to 3), which makes a difference in the used font and in element count. Then you must enter the section height and the number of sections for each letter. The SPC meshes have no real practical use, they are only for demonstration.

The generated meshes are all version 2 standard files with no boundary conditions set. To set boundary conditions the *xbc* program must be used.

3.6 The program *xbc*

3.6.1 Description

xbc was planned as a tool to check the integrity of files **.std* and as test environment for routines managing standard files. It is grown up to a visualization tool for objects stored in the standard file format (general polyhedra in boundary representation as well as 3D meshes) with the capability to create and to manipulate boundary values on that objects.

The program needs an XView environment, there is no plain X-Windows nor Motif based version.

3.6.2 Command Line Parameters

All the standard XView command line parameters are available, e.g. *-display* `displayname` or *-fg* `colorname`. *xbc -help* shows a list of these parameters. Although all of these parameters work, there is no test of bad usage implemented.

Two additional parameters allow a quick file access:

- *-InPath* `pathname` is the main path for the input files, if no *-InPath* is present, the actual working directory is used as path for the input files.

- *-InFile* `filename` : is the name of the input file relative to the input path.

A list of all implemented parameters is shown by *xbc -Help*. All other parameters will be interpreted as file names. If no input file is specified, the user has to enter file name and path manually in the File menu.

3.6.3 Loading and Saving Files

If a file name is specified in the command line, *xbc* loads this file automatically. The user can enter the file name manually by opening the File menu and choosing the Load File button.

Loading a file *xbc* first reads the information part, shows this information and asks for confirmation. During the loading process *xbc* checks the integrity of the data. Any problems will be shown in error messages and the user will be asked for continuing the reading procedure.

After a successful load procedure the Show-button becomes available. It switches to the view window.

To save a file it's necessary to choose the Save File button in the File menu and to enter the file name manually. There is no command line parameter for a standard save file.

3.6.4 The View Window

The view window, figure 3.9, is used to visualize the object and to choose faces to set boundary conditions. There are two buttons and two menus in the view window:

- The Back button switches to the main window.
- The Repaint button is reserved for a general hidden line algorithm that will be implemented soon.
- The Settings menu is used to control the behaviour of *xbc*.
- The BC menu contains tools to manipulate the boundary conditions.

The object in the view window can be rotated by moving the mouse holding the middle mouse button down. A single click with this button forces a refresh of the viewport.

Faces without boundary conditions are shown in gray, faces with Dirichlet conditions in red, and faces with Neumann conditions blue. The presence of both types of boundary conditions is represented by violet color.

3.6.5 The BC Menu

In the current release V0.9 only the Set BC button of the BC menu is active. It is used to manipulate values of boundary conditions. Pressing this button opens an Object Selection window and enables the selection mode for the mouse buttons.

Pressing the left mouse button in the viewport selects the visible face at the mouse pointer. The right mouse button unselects the face. It is also possible to select or unselect faces by editing the Face Name line in the Object Selection window. The Reset button in this window unselects all. Selected faces change their color to yellow.

The Cancel button terminates the whole value setting process, the OK button finishes the selection process and opens the Set BC Values window, see left side of figure 3.10 . This window allows the simple choice of the kind of boundary condition (Dirichlet/Neumann) as well as the input of the actual number of the degree of freedom, the equation type (Poisson or Lamé) and the set of values of that boundary condition.

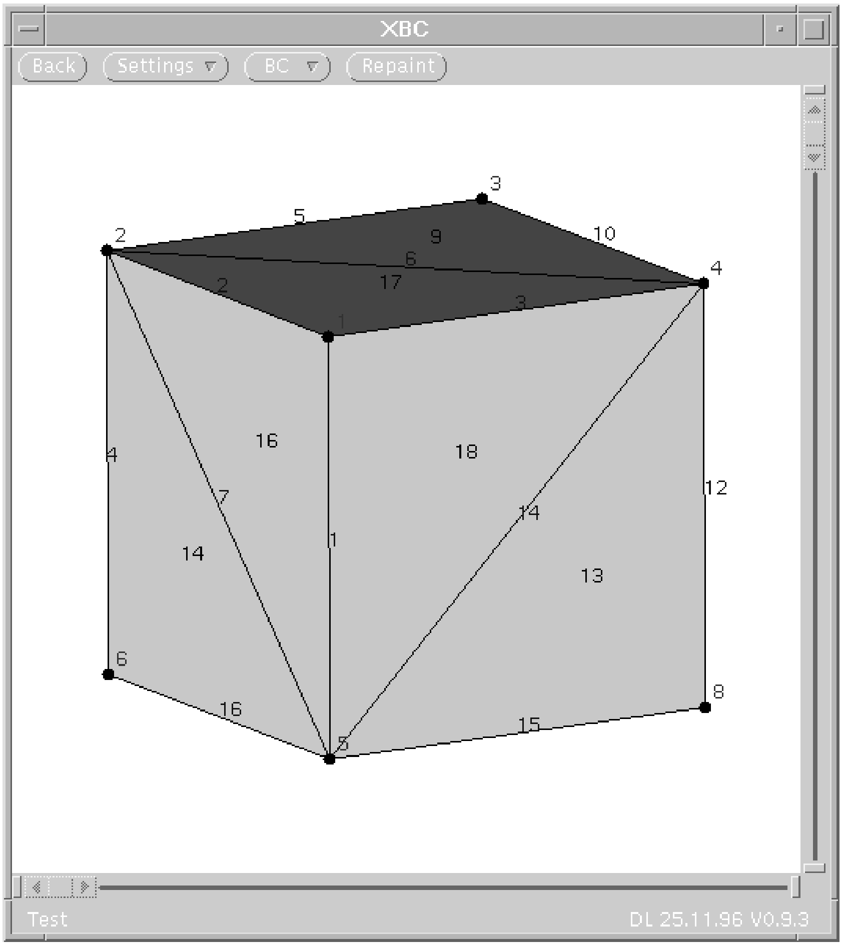


Figure 3.9: The View Window of *xbc*.

If more than one degree of freedom is used for some faces it is necessary to set **all** degrees of freedom in one step by using the Apply button of the Set BC Values window. A Set BC procedure finished by the OK button overwrites all settings of the chosen faces. Cancel stops the whole setting process.

The meaning of the equation types are:

- Free : No BC is given at the current degree.
- Const : The BC is constant on the surface, the value is given in the field Value 1.
- Lin Glob : The BC is given by $u = V1 * x + V2 * y + V3 * z + V4$, (x,y,z) are the coordinates in the global system.
- User: The values of the BC will be given by special routines of the user program.

3.6.6 The Settings Menu

The Settings menu controls the general behaviour of *xbc*. It includes two buttons, the View Control button and the Zoom menu.

The View Control button opens a window which allows to choose the drawing method (Solid; Hidden Line; Wire Frame), see right side of figure 3.10. This window is also used to control the visibility of the names (e.g. integers) of objects like vertices, edges, or faces.

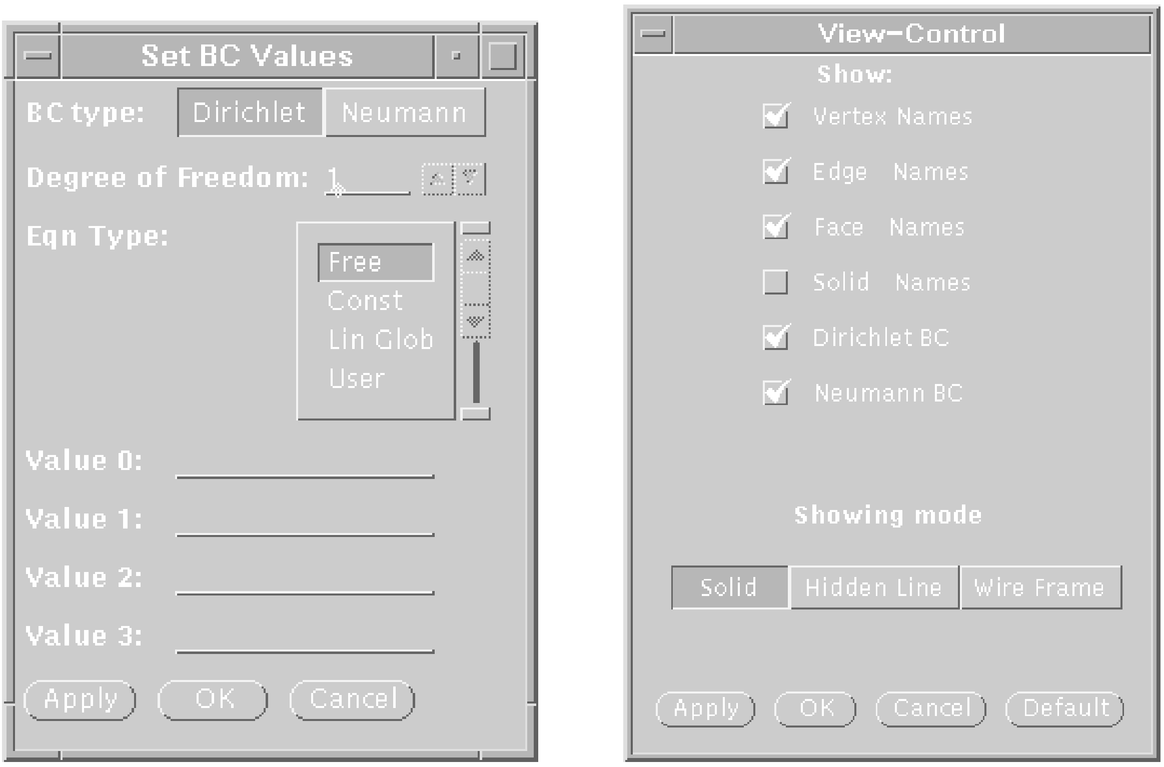


Figure 3.10: Left: The Set BC Values window. Right: The View Control window.

The Zoom menu offers some standard zoom factors and the capability to enter user defined factors (using the Other button).

3.7 Mesh construction and manipulation with *geo_conv*

The program *geo_conv* was designed to join two or more objects provided as standard files to one new object and to save this in a new standard file. To do so, the input objects could be displaced, rotated and mirrored. Notes, edges, and faces which are joined could be identified and fusioned. The program is text oriented, menu driven, and offers an online help. It is also able to operate in batch mode.

A program call should be as follows:

```
geo_conv [-v] [-t]  [-c <cfgfile>] [-p <protfile>] [-e <exefile>]
                      [<filename> [<filename> ...]];
```

is no `filename` given, the user is prompted for a filename after the program start.

The configuration file `<cfgfile>` contain `:set`-commands and additional types for face geometries.

All executed commands, its outputs and error messages could be saved in the protocol file `<protfile>`. An existing protocol file could be used as control file `<execfile>` in batch mode. With switch `-t` the program operates in trace-mode, which means the every line in the control file must be confirmed before execution. Is no control file given, the switch remains meaningless. The switch `-v` must given for displaying the current program version.

Note that the order of the given parameters is essential for there right recognition.

For a short command list type help at the command prompt after start. For a detailed description of the commands, the protocol, control, and config file we refer to the manual, which could be found under */afs/tucz/project/stb393/FEM/doc/geo_conv* in various forms.

Chapter 4

Examples

4.1 Poisson equation

4.1.1 Introduction

We consider the Poisson equation with in general mixed Dirichlet and Neumann boundary conditions:

$$\begin{aligned} -\Delta u &= f \quad \text{in } \Omega, \\ u &= u_0 \quad \text{on } \partial\Omega_1, \\ \frac{\partial u}{\partial n} &= g \quad \text{on } \partial\Omega_2, \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on } \partial\Omega \setminus \partial\Omega_1 \setminus \partial\Omega_2. \end{aligned}$$

In the next subsections we describe some test examples, which demonstrate that our code gives the right result and works very effectively.

4.1.2 *cubus1.std* with *bsp.z*

The file *cubus1.std* describes a cube $\Omega = (0,10)^3$ with Dirichlet boundary conditions $u_0 = 0$ at the bottom face $\{\underline{x} \in \bar{\Omega} : z = 0\}$ and $u_0 = 1$ at the top face $\{\underline{x} \in \bar{\Omega} : z = 10\}$. That means the boundary conditions are not taken from *bsp.f* but directly from the file, and for the successful test the program should be linked with *bsp.z*. This means a setting

$$f \equiv 0$$

for the right hand side and

$$u = \frac{z}{10}, \quad u_x = 0, \quad u_y = 0, \quad u_z = \frac{1}{10}$$

for the exact solution which is used to calculate error norms.

In this example, there is no discretization error, thus the error is proportional to error tolerance in the solver. If not, check first the integration rules, for example:

$$\begin{aligned} \texttt{Nint3ass} &= \begin{cases} 121 & \text{in the linear case,} \\ 231 & \text{in the quadratic case,} \end{cases} \\ \texttt{Nint3error} &= \quad 211 \qquad ((u-u_h)^2 \text{ is quadratic}). \end{aligned}$$

Level	linear elements			quadratic elements		
	L_∞	L_2	H^1	L_∞	L_2	H^1
0	0	5.77e+2	1.82e+2	1.84e−13	1.63e−12	6.35e−13
1	2.49e+0	1.42e+2	9.05e+1	3.85e−9	2.21e−8	2.25e−8
2	1.07e+0	3.60e+1	4.54e+1	5.66e−9	3.28e−8	3.59e−8
3	3.89e−1	9.07e+0	2.27e+1	1.37e−8	3.31e−8	5.70e−8
4	1.26e−1	2.27e+0	1.14e+1	1.05e−8	1.28e−7	1.01e−7
5	3.89e−2	5.70e−1	5.70e+0	—	—	—

Table 4.1: Discretization error for $u = z^2$.

Level	linear elements			quadratic elements		
	L_∞	L_2	H^1	L_∞	L_2	H^1
0	0	8.72e+3	2.82e+3	7.05e+1	9.69e+2	6.28e+2
1	8.60e+1	2.36e+3	1.50e+3	9.89e+0	1.22e+2	1.61e+2
2	3.28e+1	6.31e+2	7.76e+2	1.24e+0	1.56e+1	4.18e+1
3	1.15e+1	1.62e+2	3.93e+2	1.61e−1	1.99e+0	1.07e+1
4	3.58e+0	4.11e+1	1.97e+2	2.04e−2	2.52e−1	2.72e+0
5	1.06e+0	1.03e+1	9.87e+1	—	—	—

Table 4.2: Discretization error for $u = z^3$.

4.1.3 *cubusu.std* with *bsp.z*, *bsp.z2*, and *bsp.z3*

With these examples we test the discretization error orders. Again we have $\Omega = (0, 10)^3$ with Dirichlet boundary conditions at $\partial\Omega_1 = \{x \in \bar{\Omega} : z = 0 \text{ or } z = 10\}$ but this time the boundary values are taken from the corresponding function in the file *bsp.f*.

If we copy *bsp.z* ($u = \frac{z}{10}$) to *bsp.f* we get no discretization error, see Subsection 4.1.2. With *bsp.z2* the exact solution is $u = z^2$ $(f = -2)$ and we get an error with linear elements but no error with quadratic elements. The third example *bsp.z3* corresponds with $u = z^3$ $(f = -6z)$ and we observe in both cases the optimal order of the error h^{k-m+1} (k... degree of the shape functions, m... order of the Sobolev space $H^m(\Omega)$, $m = 0, 1$, to measure the error). Tables 4.1 and 4.2 contains the values.

The tests were carried out with

$$\texttt{Nint3ass} = \begin{cases} 221 & \text{for linear elements } (f\cdot\varphi \text{ is quadratic for } u = z^3), \\ 331 & \text{for quadratic elements } (f\cdot\varphi \text{ is cubic for } u = z^3), \end{cases}$$

`Nint3error` =	511	$((u-u_h)^2$ is of degree 6, but 5 is the best formula programmed),
`Epsilon` =	10^{-10},	
`LoesVar` =	4	BPX without coarse grid solver.

4.1.4 *cubusug.std*

The example differs from *cubusu.std* only by the boundary conditions. Again we have Dirichlet boundary conditions on $\partial\Omega_1 = \{x \in \bar{\Omega} : z = 0 \text{ or } z = 10\}$, but on the remaining part of the boundary we have Neumann conditions $\partial\Omega_2 = \partial\Omega \setminus \partial\Omega_1$. The values of u_0 and g are taken from *bsp.f*. The use of *bsp.z* yields no discretization error which can be used as a test.

Level	Nodes	Jacobi	Yserentant	BPX
1	27	12	16	10
2	125	24	32	16
3	729	46	49	19
4	4913	82	64	21
5	35937	159	85	22

Table 4.3: Numbers of iterations for *cubus2.std*, *bsp.xy* and 8 or 16 processors (Yserentant and BPX without coarse grid solver here).

Level	cube48	cube96	cube192	cube384	cube768
0	27	45	65	123	205
1	125	225	369	725	1305
2	729	1377	2465	4905	9265
3	4913	9537	17985	35931	69729
4	35937	70785	137345	274593	540865
5	274625	545025	1073409	2146625	—
6	2146689	—	—	—	—

Table 4.4: Number of nodes for different refinement levels.

4.1.5 *cubus2.std*

The domain and the mesh *cubus2.std* are identical to *cubus1*. The boundary conditions are

$$\partial\Omega_1 = \{\underline{x} \in \bar{\Omega} : z = 0\}, \qquad \partial\Omega_2 = \partial\Omega \setminus \partial\Omega_1,$$

where the values of u_0 and g are taken from *bsp.f*. For example, one can link with *bsp.xy* as *bsp.f* which corresponds to $u_0 = xy$, $f = g = 0$. Table 4.3 shows the number of iterations for different preconditioners in this case. We used `Epsilon` $= 10^{-4}$ and linear elements.

4.1.6 *cube*.std* with *bsp.xy*

The family of meshes *cube48.std*, *cube96.std*, *cube192.std*, *cube384.std*, and *cube768.std* was generated in order to have test examples with equidistributed coarse meshes on any number 2^k, $k = 0, \ldots, 7$, of processors and with numbers of nodes as large as possible, see Table 4.4. The number of elements of cube$n$ in level l is $n \cdot 2^l$. The domain is the cube $(0, 2)^3$.

The meshes were generated using the mesh-generator *PARMESH3D* [11] from 2D reference meshes, see Figure 4.1, which are reproduced several times into the third dimension. Thus prisms with triangular basis can be formed and divided in three tetrahedra each. The corresponding reference meshes and the number of their reproduction is given in Table 4.5. Note that cube384 and cube768 represent different meshes than cube48 and cube96, respectively, with one refinement step, though they have the same number of elements.

The mesh data are stored in the files *cube*.out* which can be processed by the program *renfindsun* in order to describe boundary conditions and to create the data structure of standard files. The boundary conditions in the standard files *cube*n*.std* are $\partial\Omega_1 = \{\underline{x} \in \bar{\Omega} : z = 0 \text{ or } z = 2\}$ where the boundary conditions are taken from the function u in *bsp.f*.

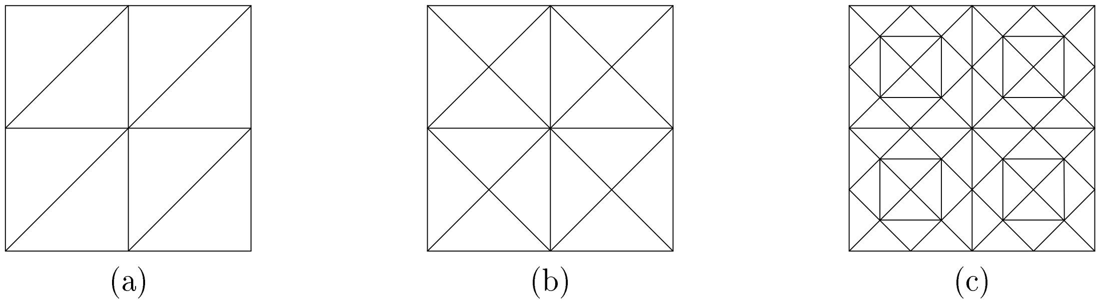


Figure 4.1: 2D reference meshes for the *cube** family.

mesh	cube48	cube96	cube192	cube384	cube768
2D reference	(a)	(a)	(b)	(c)	(c)
number of reproductions	2	4	4	2	4

Table 4.5: The *cube** family and their corresponding reference meshes.

In Table 4.6 we document tests with these meshes with *bsp.xy* used as *bsp.f* and different preconditioners.

The files *cubena.std* define $\partial\Omega_1 = \partial\Omega$ and were used in [1, 2] to compare two communication routines. The results are not reproducible because since then a scaling error in the hierarchical list was discovered and removed, which influenced the number of iterations. In Table 4.7 we give some results with the correct version of preconditioner. The tests were carried out with *bsp.xy* as *bsp.f*, which means

$$-\Delta u = 0 \text{ in } \Omega, \; u = xy \text{ on } \partial\Omega,$$

linear shape functions, *cube192a.std*, `Epsilon` $= 10^{-4}$, `LoesVar` $= 2$ (Yserentant without coarse grid solver).

4.1.7 *amw*.std* with *bsp.amw*

The *amw* family of meshes describes the domain

$$\Omega = \{\underline{x} = (r\cos\varphi, r\sin\varphi, z) \in \mathbb{R}^3 : 0 < r < 1, \; 0 < \varphi < \frac{3}{2}\pi, \; 0 < z < 1\},$$

which was used extensively in the papers [3, 5, 6] (but on serial computers). The two digits in the filename gives the number of intervals in r- and z-direction, that means their reciprocal value corresponds to the mesh size. The *d* as the last letter of the base name stands for global Dirichlet boundary conditions $\partial\Omega_1 = \partial\Omega$. Contrary, in *amw22.std* we have $\partial\Omega_1 = \{\underline{u} \in \partial\Omega : z = 0\}$.

The meshes are useful in connection with *bsp.amw*, where the exact solution is given by

$$u = (10 + z)r^\lambda \sin\lambda\varphi, \quad \lambda = \frac{2}{3}.$$

For this and only this domain Ω a value `verf` $\neq 0$ in *control.tet* is useful in order to control an anisotropic mesh refinement. The following coordinate transformation is carried out (l...refinement level, $\mu =$`verf` ...grading parameter):

$$t \;=\; 1 - \left(\frac{1}{2}\right)^{l+2},$$

cube48					
Level	LoesVar				
	1	2	3 $d = 0.07$	4	5 $d = 0.1$
1	17	19	21	14	13
2	34	33	36	17	17
3	67	47	52	21	21
4	131	69	70	23	23
5	249	93	97	25	25

cube192					
Level	LoesVar				
	1	2	3 $d = 0.05$	4	5 $d = 0.05$
1	27	25	22	18	18
2	52	40	38	21	18
3	98	58	57	24	22
4	191	80	80	25	25
5	364	101	98	26	26

cube384					
Level	LoesVar				
	1	2	3 $d = 0.05$	4	5 $d = 0.1$
1	40	35	37	31	31
2	85	53	53	45	45
3	172	74	76	60	60
4	338	100	110	71	71
5	—	138	—	77	—

cube768					
Level	LoesVar				
	1	2	3 $d = 0.05$	4	5 $d = 0.1$
1	38	33	27	27	18
2	80	46	41	32	24
3	161	67	61	38	35
4	323	95	84	42	38
—	—	—	—	—	—

Table 4.6: Iteration numbers for different preconditioners in different examples.

Input/Output time, FEMAKKVar=1			
Level	Number of processors		
	16 8 nodes	16 16 nodes	64 32 nodes
1	0.28	0.25	0.48
2	0.86	0.78	1.12
3	1.52	1.43	1.94
4	3.46	3.08	4.10

Input/Output time, FEMAKKVar=2			
Level	Number of processors		
	16 8 nodes	16 16 nodes	64 32 nodes
1	0.29	0.25	0.48
2	0.54	0.50	0.90
3	1.08	0.99	1.64
4	2.94	2.58	3.71

Total time, FEMAKKVar=1			
Level	Number of processors		
	16 8 nodes	16 16 nodes	64 32 nodes
1	0.30	0.28	0.51
2	0.94	0.84	1.16
3	1.96	1.83	2.11
4	8.36	7.11	5.34

Total time, FEMAKKVar=2			
Level	Number of processors		
	16 8 nodes	16 16 nodes	64 32 nodes
1	0.32	0.30	0.51
2	0.66	0.61	0.97
3	1.59	1.44	1.85
4	8.08	6.75	5.05

Table 4.7: Comparison of two data accumulation algorithms [2] for different numbers of processors and different problem sizes (running on Parsytec GCPP, time in seconds).

Level	amw11d		amw12d		amw21d		amw22d	
	Elements	Nodes	Elements	Nodes	Elements	Nodes	Elements	Nodes
0	12	12	24	18	48	30	96	45
1	96	45	192	75	384	135	768	225
2	768	225	1536	405	3072	765	6144	1377
3	6144	1377	12288	2601	24576	5049	49152	9537
4	49152	9537	98304	18513	196608	36465	393216	70785
5	393216	70785	786432	139425	1572864	276705	3145728	545025
6	3145728	545025	—	—	—	—	—	—

Table 4.8: Number of nodes for different refinement levels.

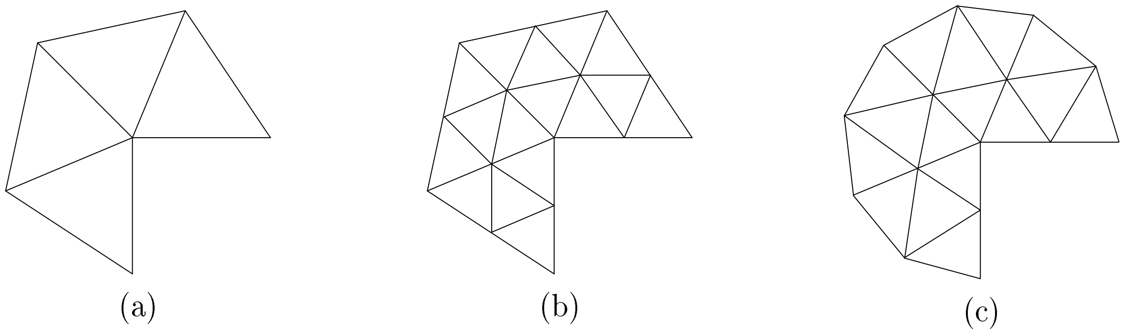


Figure 4.2: (a) coarse mesh with $z = 0$, (b) one refinement step with $\mu = 0$, (c) one refinement step with $\mu = 1$.

$$\begin{aligned} r &= (x_{old}^2 + y_{old}^2)^{\frac{1}{2}}, \\ h &= \begin{cases} r^{-1+\frac{1}{\mu}} & \text{if } r \le t \\ r^{-1} & \text{if } r > t \end{cases}, \\ x_{new} &= h \cdot x_{old}, \\ y_{new} &= h \cdot y_{old}. \end{aligned}$$

For $\mu =$ `verf` $= 1$ we get a change in the coordinates only for points with $r > t$, that means they are moved on the curved boundary, see Figure 4.2.

In Tables 4.9 and 4.10 we show some results for the error behaviour for different values of $\mu =$ `verf`. The tests were carried out with *amw22d.std* and the following parameters

$$\begin{aligned} \texttt{Nint3ass} &= \begin{cases} 121 & \text{for linear elements,} \\ 231 & \text{for quadratic elements,} \end{cases} \\ \texttt{Nint3error} &= \quad 511, \\ \texttt{Nint2ass} &= \quad 11, \\ \texttt{Nint2error} &= \begin{cases} 11 & \text{for linear elements,} \\ 12 & \text{for quadratic elements,} \end{cases} \\ \texttt{Epsilon} &= \quad 10^{-10}, \\ \texttt{LoesVar} &= \quad 4 \qquad \text{(BPX without coarse grid solver).} \end{aligned}$$

4.1.8 *fichera*.std*

The domain mesh is $\Omega = (-1,1)^3 \setminus [0,1]^3$ which is known as a Fichera corner. It was used with the sequential code for the tests in [7], but not yet on the parallel computer. The digit * means in analogy to 4.1.7 the reciprocal of the meshsize.

Level	linear elements			quadratic elements		
	L_∞	L_2	H^1	L_∞	L_2	H^1
1	4.0797e−1	1.6819e−1	2.3825e+0	2.0854e−1	3.5542e−2	9.2351e−1
2	3.3811e−1	7.6697e−2	1.5922e+0	1.4325e−1	1.3205e−2	5.6153e−1
3	2.3133e−1	3.2269e−2	1.0164e+0	9.3024e−2	4.8989e−3	3.4802e−1
4	1.5039e−1	1.3063e−2	6.4116e−1	5.9467e−2	1.8467e−3	2.1748e−1
5	9.5848e−2	5.1834e−3	4.0279e−1	—	—	—

Table 4.9: Discretization error for `verf` $= \mu = 1$.

Level	linear elements			quadratic elements		
	L_∞	L_2	H^1	L_∞	L_2	H^1
1	2.8163e−1	1.9642e−1	2.2927e+0	1.3677e−1	7.8165e−2	1.2327e+0
2	1.3241e−1	7.2266e−2	1.3627e+0	4.9917e−2	2.0029e−2	4.3328e−1
3	6.0739e−2	1.8634e−2	7.1794e−1	2.2336e−2	4.4908e−3	1.5496e−1
4	2.5277e−2	5.0494e−3	3.7083e−1	1.2162e−2	8.2507e−4	5.5923e−2
5	1.0240e−2	1.3128e−3	1.8848e−1	—	—	—

Table 4.10: Discretization error for `verf` $= \mu = 0.5$ with linear elements and `verf` $= \mu = 0.3$ with quadratic elements.

4.1.9 *fem.std*

The domain consists of the letters FEM which have a different size in the third direction. The coarse mesh consists of 93 elements with 122 nodes. We have Dirichlet boundary conditions at the bottom face $\partial\Omega_1 = \{\underline{x} \in \partial\Omega : z = 0\}$. In Figure 4.4 we demonstrate isolines at the surface of the domain (calculated with *bsp.xy*, Level=2).

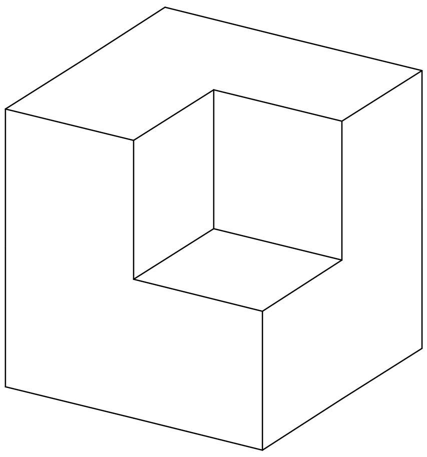


Figure 4.3: Fichera corner.

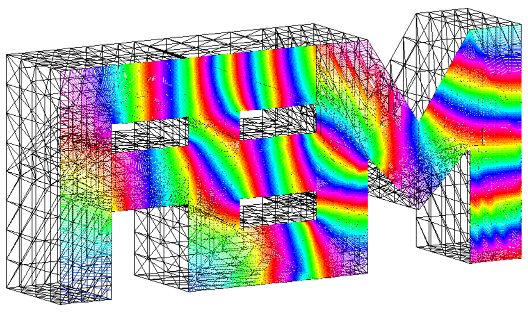


Figure 4.4: Isolines on FEM.

4.2 Lamé system

4.2.1 Introduction

We consider the Lamé equation system

$$-\mu\Delta\underline{u} + (\lambda+\mu)\ \text{grad}\ \text{div}\underline{u} = \underline{f}$$

for $u = (u^{(1)}, u^{(2)}, u^{(3)})^T$ with the boundary conditions

$$\begin{aligned} u^{(i)} &= u_0^{(i)} \quad \text{on } \partial\Omega_1^{(i)}, \quad i = 1,\dots,3, \\ t^{(i)} &= g^{(i)} \quad \text{on } \partial\Omega_2^{(i)}, \quad i = 1,\dots,3, \\ t^{(i)} &= 0 \quad \text{on } \partial\Omega \setminus \partial\Omega_1^{(i)} \setminus \partial\Omega_2^{(i)}, \quad i = 1,\dots,3, \end{aligned}$$

where $\underline{t} = (t^{(1)}, t^{(2)}, t^{(3)})^T$ is the normal stress.

4.2.2 *druck.std*

This is again the cube $\Omega = (0,10)^3$, divided into six tetrahedra. The boundary conditions are

$$\begin{aligned} \underline{u} &= \underline{0} && \text{on } \{\underline{x} \in \partial\Omega : z = 0\}, \\ \underline{u} &= \begin{pmatrix} 0 \\ 0 \\ -2 \end{pmatrix} && \text{on } \{\underline{x} \in \partial\Omega : z = 10\}, \\ \underline{t} &= \underline{0} && \text{elsewhere.} \end{aligned}$$

The exact solution is not known. With $\nu = 0.3$, $E = 2 \cdot 10^5$ we get a deformation as shown in Figure 4.5.

4.2.3 *zug.std* and *zug1.std*

The two files *zug.std* and *zug1.std* describe the same example, but they were created by different programs. We have again the cube $\Omega = (0,10)^3$. The boundary conditions are

$$\underline{u} = \underline{0} \qquad \text{on } \{\underline{x} \in \partial\Omega : z = 0\},$$

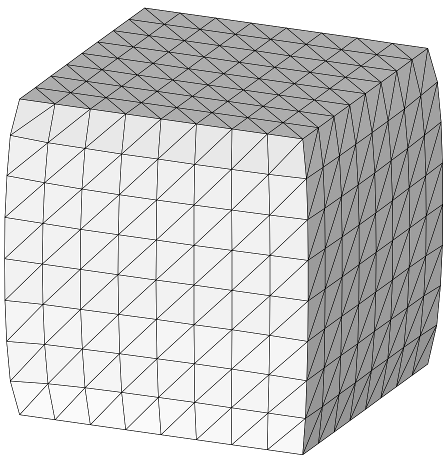


Figure 4.5: Cube under pressure.

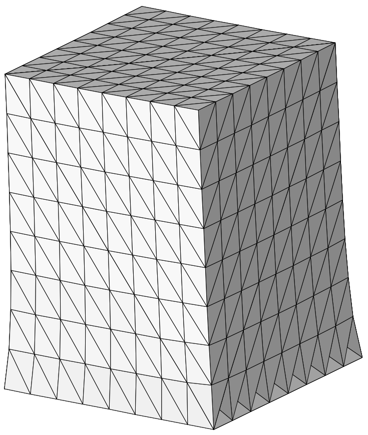


Figure 4.6: Cube under pull.

$$\begin{aligned} \underline{t} &= \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} && \text{on } \{\underline{x} \in \partial\Omega : z = 10\}, \\ \underline{t} &= \underline{0} && \text{elsewhere.} \end{aligned}$$

The exact solution is not known. We calculated again with $\nu = 0.3$, $E = 2 \cdot 10^5$, the result is shown in Figure 4.6.

4.2.4 The *etest* family

For tests of the validity of the computer results we use the following example, which is described by *bsp.etest*:

$$\lambda = \mu = 0.2, \quad \underline{u} = \begin{pmatrix} x \\ y \\ z \end{pmatrix},$$

consequently $\underline{f} = \underline{0}$, $\underline{t} = \underline{n}$. There is no discretization error, thus the error is in the range of the error of the solver.

We prepared two test examples: In *etestd.std* and *etestdu.std* the whole boundary is of Dirichlet type, while in *etest.std* and *etestu.std* also Neumann boundary conditions appear:

$$\begin{aligned} \underline{u} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} && \text{on } \{\underline{x} \in \partial\Omega : z = 0 \text{ or } z = 10\}, \\ \underline{t} &= \underline{n} && \text{elsewhere.} \end{aligned}$$

The files with and without the *u* at the end of the basename differ by the way the boundary conditions are described. In the version without the *u* the data of the conditions are defined in the file, whereas in the version with *u* the functions from *bsp.f* are called.

There is a third pair of files in this family: *etest1.std* and *etest1u.std*, which differs from the first two pairs by the Neumann condition

$$\begin{aligned} \underline{u} &= \begin{pmatrix} x \\ y \\ z \end{pmatrix} && \text{on } \{\underline{x} \in \partial\Omega : z = 0 \text{ or } z = 10\}, \\ \underline{t} &= \underline{0} && \text{elsewhere.} \end{aligned}$$

In this case the exact solution is not known.

4.2.5 *lame22d.std* with *bsp.lame*

This is a test with a known solution which has the typical behaviour near an edge. The domain and the meshes are the same as in 4.1.7, the exact solution is

$$\underline{u} = \begin{pmatrix} r^{5/9}[\sqrt{3}(-\cos\frac{5}{9}\varphi + \cos\frac{13}{9}\varphi) - (5\sin\frac{5}{9}\varphi + \sin\frac{13}{9}\varphi)] \\ r^{5/9}[\sqrt{3}(-3\sin\frac{5}{9}\varphi + \sin\frac{13}{9}\varphi) + (-\cos\frac{5}{9}\varphi + \cos\frac{13}{9}\varphi)] \\ r^{2/3}\sin\frac{2}{3}\varphi \end{pmatrix}.$$

and with $\nu = \frac{17}{36}$ we get $\underline{f} = 0$ in Ω and $\underline{u} = 0$ on the faces forming the edge. In *lame22d.std* there are Dirichlet boundary conditions defined on the whole boundary, the values are taken from *bsp.f* = *bsp.lame*. Unfortunately the system is very badly conditioned because ν is close to $\frac{1}{2}$: $\frac{1}{2} - \nu = \frac{1}{36}$, which results in very high iteration numbers.

Examples of the error behaviour for different values for `verf` are given in the file *mesh3/lame22d.txt*. The numbers are not really promising, may be there is still an error anywhere. Hints are welcome.

Appendix A

Mesh generation and related programs

Our research group has been developing several programs for the automatic generation of meshes (2D/3D, sequential/parallel) and their visualization. Due to historical reasons, the pre-, main-, and postprocessing tools use input and output files with different data structure. Therefore a few little programs for converting the files from one structure into the other have been made available. This is useful for reusing meshes in other programs for example for benchmark tests. A survey of the programs and tools is given in Figure A.1, stressing their connection with respect to the data structure. A detailed description of the programs is beyond the scope of this manual, we restrict ourselves to the following list. Note that `$AFS` stands for */afs/tu-chemnitz.de/project/sfb393/FEM/bin*; such programs can be accessed from all computers with AFS installed, the drive letter f: for DOS files stands for *riemann_home2:\public\numwork*.

Graphical editors

GRAFED: (*f:\femtools\grafedv2.exe*) Graphical editor for describing geometrical data in 2D at PC, storing them as `file`.*inp* (M. Fritz), see also [8].

NETS: (*k:\util\nets\net.exe*) as *GRAFED*, but storing data as `file`.*net* (M. Seibt, M. Pester).

Automatic mesh generation

PARMESH3D: (`$AFS`*/parix/parmesh3d.px*, `$AFS`*/ppc/parmesh3d.px*) Automatic parallel 2D/3D mesh generation [10, 11], output files have structure `file`.*out* or *2D*-`file`.*wqf* (G. Globisch).

PREMESH: (`$AFS`*/SUN4/premeshg, f:\femtools\premesh.exe*) Sequential 2D grid generation in a UNIX and DOS version [22] (M. Goppold).

Converting data structures

GRAFEDSUN: (`$AFS`*/SUN4/grafedsun*) Converting `file`.*inp* (see *GRAFED*) into `file`.*bsp* (see *FEM⊕BEM*) and vice versa (G. Haase).

GUNDOLFSUN: (`$AFS`*/SUN4/gundolfsun*) Converting `file`.*net* (see *NETS*) into `file`.*bsp* (see *FEM⊕BEM*) and vice versa (G. Haase).

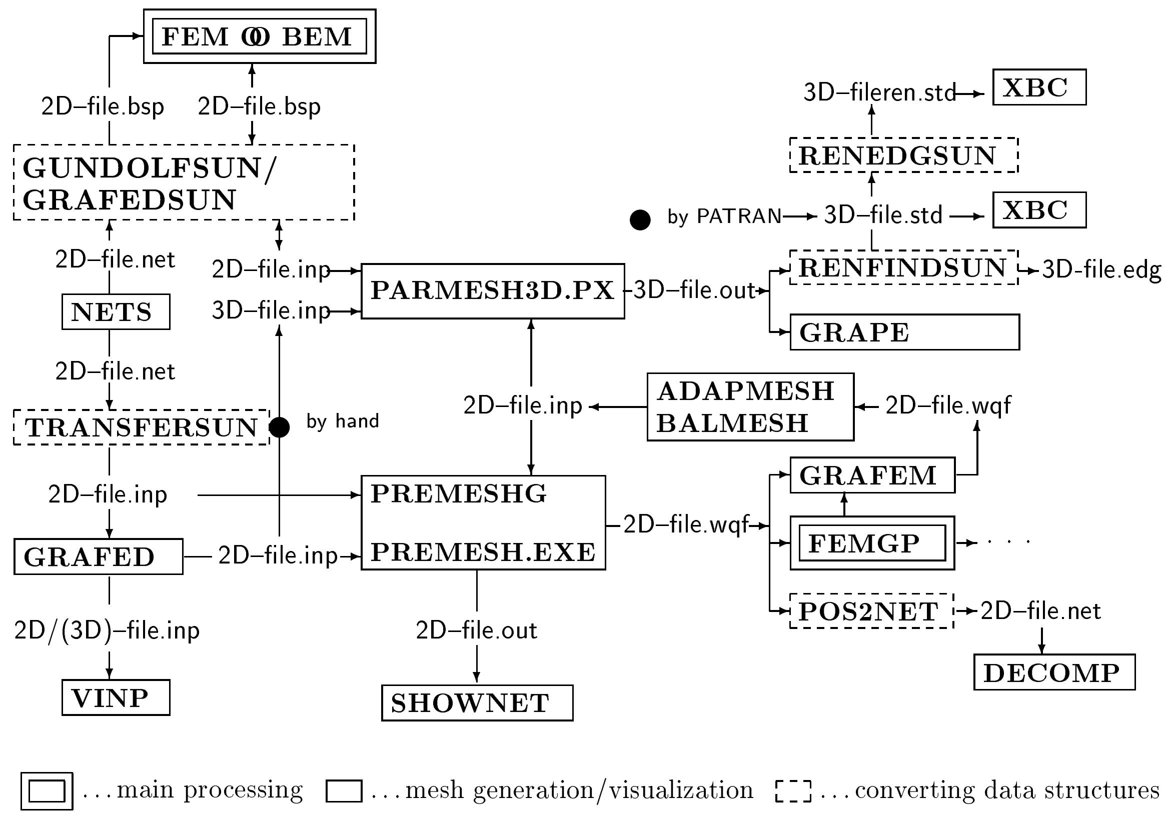


Figure A.1: Connection of the tools corresponding to the data structure.

POS2NET: ($AFS/*SUN4/pos2net*) Converting `file`.*wqf* into `file`.*net*, involving a renumbering of the nodes and the setting of boundary conditions (G. Globisch).

RENEDGSUN: ($AFS/*SUN4/renedgsun*) Renumbering the nodes to minimize the matrix profile, input and output are files of **.std* structure (G. Globisch).

RENFINDSUN: ($AFS/*SUN4/renfindsun*) Conversion of `file`.*out* (3D; node related, see [10, 22]) into `file`.*std* (3D; edge related, see 3.2), involving a renumbering of the nodes to minimize the matrix profile and the interactive setting of boundary conditions (G. Globisch), see also 3.3, 3.4.2.

TRANSFERSUN: ($AFS/*SUN4/transfersun*) Converting `file`.*net* (see *NETS*) into `file`.*inp* (see *GRAFED*) (G. Globisch).

NET4STD: ($AFS/*SUN4/net4std*, $AFS/*HPPA/net4std*, $AFS/*LINUX/net4std*) Takes **.net* files as surface and extends them into the third dimension (M. Pester).

Visualization[1]

F3: ($AFS/*SUN4/f3_sun* (dynamically linked), $AFS/*SUN4/f3grape.sun* (statically linked), $AFS/*SGI5/f3_sgi* (dynamically linked), $AFS/*SGI5/f3grape.sgi* (statically linked)) Visualization of 3D data received via socket connection [19]; based on GRAPE (M. Meyer).

[1]see also **Graphical editors** above

GRAFEM: (*f:\femtools\grafem.exe*) Visualization of 2D FEM data including the solution, data type `file`.*wql* [22] (G. Haase).

GRAPE: (`$AFS`*/SUN4/grape.sun*) Visualization of 3D data files `file`.*out* (node related structure) based on *GRAPE* [25] (Th. Hommel).

SHOWNET: (`$AFS`*/SUN4/shownet*) Visualization of 2D FEM data including the solution (isolines), data type `file`.*out*, possible output as ps-file (F. Bräuer).

VINP: (*f:\femtools\vinp.exe*, `$AFS`*/SUN4/vinp*) Visualization of 2D data files `file`.*inp* [22] (M. Goppold).

XBC: (`$AFS`*/SUN4/xbc*, `$AFS`*/SUN4/xbc*) Visualization of 3D data files `file`.*std* (edge related) and modification of boundary conditions (D. Lohse), see also Section 3.6.

Other preprocessing

DECOMP: (`$AFS`*/SUN4/decomp*) Spectral graph partitioning of finite element meshes for parallel computations (M. Goppold).

CHACO: (`$AFS`*/SUN4/chaco*, `$AFS`*/HPPA/chaco*, `$AFS`*/LINUX/chaco*) Partitioning of finite element meshes for loadbalacing of parallel computations, see[23] and the citations therein.

STD2GRAPH: (`$AFS`*/SUN4/std2graph*, `$AFS`*/HPPA/std2graph*) Generating of input files for *chaco* from **.std* files (U. Reichel).

Q2T: (`$AFS`*/SUN4/q2t*, `$AFS`*/HPPA/q2t*) Converting hexahedral **.std* files into tedrahedral **.std* files (U. Reichel).

Main processing

FEMGP: Package for solving 2D-boundary value problems on sequential computers, see [22], based on files `file`.*wqf*, and partially `file`.*out* (M. Jung, T. Steidten, W. Queck, and others).

FEM©BEM: Package for solving 2D-boundary value problems using a coupled FEM-BEM-strategy on parallel computers, based on files `file`.*bsp*, see [12] (G. Haase, M. Jung, and others).

FEMPS3D: Package for solving the Poisson equation over 3D domains on sequential computers, see [3], based on internal mesh generation and on another file structure **.ada* (Th. Apel, F. Milde).

SPC-PM CFD: (*/afs/tucz/home/urz/p/pester/workcfd/pmhi.ppc.px*) Parallel simulation of fluid dynamics in 2D (St. Meinel, A. Meyer).

SPC-PM EL 2D: (*/afs/tucz/home/urz/p/pester/workel/pmhi.ppc.px*) Parallel simulation of elasticity in 2D (A. Meyer).

SPC-PM Po 2D: (*/afs/tucz/home/urz/p/pester/worksy/pmhi.ppc.px*) Parallel simulation of potential problems in 2D (A. Meyer).

Bibliography

[1] Th. Apel, G. Haase, A. Meyer, and M. Pester. Numerical comparison of two communication algorithms. Documentation, TU Chemnitz-Zwickau, 1995.

[2] Th. Apel, G. Haase, A. Meyer, and M. Pester. Parallel solution of finite element equation systems: efficient inter-processor communication. Preprint SPC95_5, TU Chemnitz-Zwickau, 1995.

[3] Th. Apel and F. Milde. Comparison of several mesh refinement strategies near edges. *Comm. Numer. Methods Engrg.*, 12:373–381, 1996.

[4] Th. Apel, F. Milde, and M. Theß. *SPC-PMPo 3D* — Programmer's Manual. Preprint SPC95_34, TU Chemnitz-Zwickau, 1995.

[5] Th. Apel, R. Mücke, and J. R. Whiteman. An adaptive finite element technique with a-priori mesh grading. Technical Report 9, BICOM Institute of Computational Mathematics, 1993.

[6] Th. Apel and S. Nicaise. Elliptic problems in domains with edges: anisotropic regularity and anisotropic finite element meshes. Preprint SPC94_16, TU Chemnitz-Zwickau, 1994.

[7] Th. Apel, A.-M. Sändig, and J. R. Whiteman. Graded mesh refinement and error estimates for finite element solutions of elliptic boundary value problems in non-smooth domains. *Math. Methods Appl. Sci.*, 19:63–85, 1996.

[8] M. Fritz. Grafischer Editor für die Netzmanipulation. Diplomarbeit, TU Chemnitz, Sektion Mathematik, 1991.

[9] G. Globisch. Der Algorithmus HYBRID zur Knotenumordnung. Praktikumsarbeit, TH Karl–Marx–Stadt, Sektion Mathematik, 1985.

[10] G. Globisch. On an automatically parallel generation technique for tetrahedral meshes. Preprint SPC94_6, TU Chemnitz–Zwickau, 1994.

[11] G. Globisch. PARMESH – a parallel mesh generator. *Parallel Computing*, 21:509–524, 1995.

[12] G. Haase, B. Heise, M. Jung, and M. Kuhn. FEM⚭BEM – A parallel solver for linear and nonlinear coupled FE/BE–equations. Report Nr. 96–16, DFG–Schwerpunkt Randelementmethoden, 1994.

[13] G. Haase, Th. Hommel, A. Meyer, and M. Pester. Bibliotheken zur Entwicklung paralleler Algorithmen. Preprint SPC95_20, TU Chemnitz–Zwickau, 1995. Updated version of SPC94_4 and SPC93_1.

[14] G. Kunert. Error estimation for anisotropic tetrahedral and triangular finite element meshes. Preprint SFB393/97-17, TU Chemnitz, 1997.

[15] D. Lohse. Datenschnittstelle I – Standardfile. Preprint SPC95_Z, TU Chemnitz-Zwickau, 1995. In preparation.

[16] D. Lohse. Ein Standard-File für 3D-Gebietsbeschreibungen – Definition des Fileformats V 2.1 –. Preprint SFB393/98-11, TU Chemnitz, April 1998.

[17] M. Meisel and A. Meyer. Implementierung eines parallelen vorkonditionierten Schur-Komplement CG-Verfahrens in das Programmpaket FEAP. Preprint SPC95_2, TU Chemnitz–Zwickau, 1995.

[18] A. Meyer and M. Pester. Verarbeitung von Sparse-Matrizen in Kompaktspeicherform (KLZ/KZU). Preprint SPC94_12, TU Chemnitz–Zwickau, 1994.

[19] M. Meyer. Grafik-Ausgabe vom Parallelrechner für 3D-Gebiete. Preprint SPC95_4, TU Chemnitz–Zwickau, 1995.

[20] M. Pester. Grafik-Ausgabe vom Parallelrechner für 2D-Gebiete. Preprint SPC94_24, TU Chemnitz-Zwickau, 1994.

[21] M. Pester. Behandlung gekrümmter Oberflächen in einem 3D-FEM-Programm für Parallelrechner. Preprint SFB393/97-10, TU Chemnitz, April 1997.

[22] W. Queck. FEMGP – Finite-Element-Multi-Grid-Package – Programmdokumentation und Nutzerinformation. Report, TU Chemnitz-Zwickau, Fachbereich Mathematik, 1993.

[23] U. Reichel. Partitionierung von Finite-Elemente-Netzen. Preprint SFB393/96-18, TU Chemnitz–Zwickau, 1996.

[24] C. Walshaw and M. Berzins. Dynamic load balancing for PDE solvers on adaptive unstructured meshes. Research Report 92.32, University of Leeds, School of Computer Studies, 1992.

[25] A. Wierse and M. Rumpf. GRAPE – Eine objektorientierte Visualisierungs- und Numerikplattform. *Informatik Forsch. Entw.*, 7:145–151, 1992.

Index

The italic numbers denote the pages where the corresponding entry is described, numbers underlined point to the definition, all others indicate the places where it is used.