NETGEN - 4.0

Joachim Schöberl

January 16, 2001

Contents

Chapter 1

Getting Started

1.1 What is NETGEN

NETGEN is an automatic mesh generation tool for two and three dimensions. It comes as stand alone programme with graphical user interface, or as C++ library to be linked into an other application. Netgen is available for Unix/Linux and Windows 98/NT. Netgen generates triangular or quadrilateral meshes in 2D, and tetrahedral meshes in 3D. The input for 2D is described by spline curves, and the input for 3D problems is either defined by constructive solid geometries (CSG), see Chapter 2, or by the standard STL file format. NETGEN contains modules for mesh optimization and hierarchical mesh refinement.

1.2 The history of NETGEN

The NETGEN project was started 1994 in the master's programme of Joachim Schöberl, under supervision of Prof. Ulrich Langer, at the Department of Computational Mathematics and Optimization, University Linz, Austria. Its further development was supported by the Austrian science Fund "Fonds zur Förderung der wissenschaftlichen Forschung" (http://www.fwf.ac.at) under projects P 10643-TEC and SFB 1306. Since 1997 Wolfram Mühlhuber and Hannes Gerstmayr are involved into the development of NETGEN. Netgen was first released 1996. The current major release, NG4, was released 2000.

1.3 How to receive NETGEN

NETGEN is available from the WEB at

http://www.sfb013.uni-linz.ac.at/~joachim/netgen

You have to sign the license agreement found there and send it to

Joachim Schöberl
SFB "Numerical and Symbolc Scientific Computing"
Johannes Kepler Universität Linz
Freistädterstraße 313
4040 Linz
AUSTRIA

Then you will get the password for downloading NETGEN from the site given above.

1.4 Installing NETGEN 4.0

1.4.1 Installing NETGEN for Unix/Linux

To install NETGEN on Unix/Linux you will download the source code and compile it yourself. You need the following libraries:

- The 3D visualization library **OpenGL**. It comes with most systems with hardware graphics. The free software version mesagl is available from http://www.mesa3d.org.
- The graphical toolkit **TclTk** developed by John Ousterhout and the extension **Tix**. Both are available from http://www.scriptics.com/. Netgen has been tested with version TclTk 8.0 and Tix 4.6.

You can also download these packages from the Netgen site.

To install NETGEN 4 please move to the directory ng4. You set the Unix-variable MACHINE according to your machine/operating system, e.g.

```
setenv MACHINE LINUX
```

(in bash shell you type `export MACHINE=LINUX`). The Makefile includes the makefile-include

```
libsrc/makefile.mach.$(MACHINE)
```

Please create/modify the according file, (e.g. copy makefile.mach.LINUX to makefile.mach.SUN). Then you enter `make` to build the executable.

To make NETGEN globally available you just copy the binary "ng" to the global bin - directory. In difference to earlier versions, it needs no additional files.

1.4.2 Installing NETGEN for Windows 98/NT

NETGEN is available now for Windows in binary form. You download the zip - archive `ng4win.zip`. After unpacking it with winzip, you can start the executable "ng4.exe".

1.4.3 Testing Netgen

Please start Netgen by entering "ng" or clicking the "ng.exe" icon.

A black window with menu items should appear. Please load a geometry file by selecting "File `->` Load Geometry", choose e.g. examples/cube.geo. Then press the button "Generate Mesh". By keeping pressed the left, middle or right button of your mouse you can rotate, move or zoom the object. With "File `->` Export Mesh" you can save the mesh file.

Chapter 2

Constructive Solid Geometry (CSG)

The CSG input format is a useful geometry format for small and medium size geometries. One defines the geometry by writing an ASCII file in a text editor.

The geometry is defined by the Eulerian operations (union, intersection and complement) from primitives. A complete list of availabe primitives is given in Section 2.1.

The following input describes a cube:

```
# A cube
algebraic3d
solid cube = orthobrick (0, 0, 0; 1, 1, 1);
tlo cube;
```

Lines starting with . are comments. Every CSG file must contain the keyword `algebraic3d` before any non-comment line. The keyword `solid` defines a named solid, here the solid *cube* is defined. A solid is defined by the Eulerian operations applied to primitives. Here, the solid is just the primitve defined by `orthobrick`. This is a brick parallel to the axis, specified by the minimal x, y, and z coordinates, and the maximal x, y, and z coordinates. The present definition gives the cube $[0,1]^3$. Finally, the definition `tlo cube` declares the solid *cube* as a top-level-object, what is necessary for meshing.

Please start netgen with the geometry file above by entering

ng cube.geo

Instead, you can also load the geometry from the file menu. You will see a blue cube, which you can rotate by keeping the left mouse button pressed. Pressing the big **generate mesh** button will result in a (very coarse) mesh of that cube.

Instead of using the primitive `orthobrick`, one can also specify a cube by intersecting six halfspaces (called planes). Each primitive `plane` is given by an arbitrary point in the plane, and a outward vector, not necessarily a unit vector. The six halfspaces are intersected by the keyword `and`. The following input gives an equivalent result:

```
# A cube
algebraic3d
solid cube = plane (0, 0, 0; 0, 0, -1)
         and plane (0, 0, 0; 0, -1, 0)
         and plane (0, 0, 0; -1, 0, 0)
         and plane (1, 1, 1; 0, 0, 1)
         and plane (1, 1, 1; 0, 1, 0)
         and plane (1, 1, 1; 1, 0, 0);
tlo cube;
```

To drill a hole though the cube, we will intersect the cube and the complement of a cylinder. A cylinder is defined by two points on the central axis, and the radius. Please note, a cylinder is understood as an infinitely long cylinder (although the visualization may suggest a finite cylinder):

```
# cube with hole
algebraic3d
solid cubehole = orthobrick (0, 0, 0; 1, 1, 1)
         and not cylinder (0.5, 0.5, 0; 0.5, 0.5, 1; 0.1);
tlo cubehole;
```

Like `and` denotes the intersection, `or` denotes the union:

```
solid cubeball = orthobrick (0, 0, 0; 1, 1, 1)
        or sphere (0, 0, 0; 0.5) -maxh = 0.2;
```

The flag `-maxh=0.2` assignes the maximal mesh size of 0.2 to the solid. The current version, NG4.1, uses the mesh size assigned to the main solid of the top-level-object for the domain. Future version will contain more possibilities to define mesh-sizes for parts of a domain.

It is possible to define geometries with several sub-domains, simply by declaring several tlos:

```
algebraic3d
solid cube = orthobrick (0, 0, 0; 1, 1, 1);
solid cyl = cylinder (0.5, 0.5, 0; 0.5, 0.5, 1; 0.1);
solid dom1 = cube and not cyl;
solid dom2 = cube and cyl;

tlo dom1 -col=[0,0,1] -transparent;
tlo dom2 -col=[1,0,0];
```

This example show also solid trees involving previously defined named solids. Top-level-objects can be assigned a color specified by the amount of red, green and blue (RGB) values. The flag `-transparent` makes the solid appear transparent.

It is possible to specify bounday condition numbers for individual surfaces of a solid. The flag `-bc` assignes the bc to all surfaces of that solid-tree. If several flags are given the one closest to the leaves of the tree dominates. The following file defines a cube, with $bc = 1$ at the bottom, $bc = 2$ at the top, and $bc = 3$ for all other surfaces:

```
algebraic3d

solid bottom = plane (0, 0, 0; 0, 0, -1) -bc=1;
solid top    = plane (1, 1, 1; 0, 0,  1) -bc=2;
solid cube = bottm and top
         and plane (0, 0, 0; 0, -1, 0)
         and plane (0, 0, 0; -1, 0, 0)
         and plane (1, 1, 1; 0, 1, 0)
         and plane (1, 1, 1; 1, 0, 0) -bc=3;

tlo cube;
```

2.1 Available Primitives

Netgen 4.1 supports the following primitives:

1. A halfspace, i.e., a plane and everything on one side of it, given by an arbitrary point $p = (p_x, p_y, p_z)$ in the plane and an outside normal vector $n = (n_x, n_y, n_z)$, not necessarily a unit vector:

   `plane` (p_x , p_y , p_z ; n_x , n_y , n_z)

2. A cylinder of infinite length, given by two points $a = (a_x, a_y, a_z)$ and $b = (b_x, b_y, b_z)$ on the central axis and the radius r:

   `cylinder` (a_x , a_y , a_z ; b_x , b_y , b_z ; r)

3. A sphere, given by the center $c = (c_x, c_y, c_z)$ and the radius r:

   `sphere` (c_x , c_y , c_z ; r)

4. A cone is given by two points on the central axis and the two corresponding radii. It is not possible to mesh the top of the cone yet, it must be cut off.

   `cone` (a_x , a_y , a_z ; r_a ; b_x , b_y , b_z ; r_b)

5. A orthobrick is a brick parallel to the coordinate axis. It is specified by two opposite corner points $a = (a_x, a_y, a_z)$ and $b = (b_x, b_y, b_z)$:

   orthobrick (a_x , a_y , a_z ; b_x , b_y , b_z)

6. A polyhedron is defined by a set of triangles forming a closed polyhedron. First, a set of points is defined, then the triangles are given by point indices. The triangles must be oriented counter-clockwise when looking onto the object. The following polyhedron describes a tetrahedron:

```
algebraic3d
solid poly = polyhedron (0,0,0; 1,0,0; 0,1,0; 0,0,1 ;;
                         1,3,2 ; 1,4,3; 1,2,4 ; 2,3,4 );
tlo poly;
```

2.2 Known problems and work-arounds

2.2.1 Interfaces

A airdomain with two connected interior parts may be described by

```
algebraic3d

solid cube = orthobrick (0, 0, 0; 1, 1, 1);
solid part1 = orthobrick (0.2, 0.2, 0.2; 0.5, 0.8, 0.8);
solid part2 = orthobrick (0.5, 0.2, 0.2; 0.8, 0.8, 0.8);
solid air = cube and not part1 and not part2;

tlo air;
tlo part1;
tlo part2;
```

The problem is, that a domain is an open domain. Thus, the domain *air* is not only the outer part, but also the interface between *part1* and *part2*. The result is unspecified. To fix this problem, one can define the *air*-domain by cutting out one big brick:

```
solid air = cube and not othrobrick  (0.2, 0.2, 0.2; 0.8, 0.8, 0.8);
```

2.2.2 Degenerated edges

Degenerated edges are found sometimes, but some still cause troubles. A sphere on top of a cylinder my be described by:

```
solid cyl = cylinder (0, 0, 0; 1, 0, 0; 0.5)
        and plane (0, 0, 0; -1, 0, 0)
        and plane (1, 0, 0;  1, 0, 0);
solid main = cyl or sphere (1, 0, 0; 0.5);
tlo main;
```

The edge is a degenerated one. A work-around is to split the domain (artificially) into two non-degenerated parts:

```
solid cyl = cylinder (0, 0, 0; 1, 0, 0; 0.5)
        and plane (0, 0, 0; -1, 0, 0)
        and plane (1, 0, 0;  1, 0, 0);
solid hemisphere = sphere (1, 0, 0; 0.5) and not plane (1, 0, 0; -1, 0, 0);
tlo cyl;
tlo hemisphere;
```

Chapter 3

Other Geometry Formats

3.1 Using STL Geometries

STL is a standardized file format to describe (approximate) geometies by triangulated surfaces. It is useful to describe complicated parts which are modeled with some CAD programmes. Also, some users have written their own (C) programmes to define STL geometries, where was not so easy to use the CSG format. The syntac of STL files is as follos

(not available yet. please figure out the syntax from the examples)

We found that many STL geometries have some difficulties. Some of them can be corrected (removed) by the **STL - Doctor**. Please see the corresponding manual pages.

3.2 2D Spline Geometry

The extension for 2D spline geometry is ".in2d".

The boundary is given in terms of straight lines and of quadratic rational spline patches. A line is given by the two endpoints, a spline patch by 3 d'Boor points. The patch is an elliptic arc from point 1 to point 3, such that the lines 1-2 and 2-3 are tangents.

It is possible to use different subdomains with this format.

This file format also supports a priori mesh grading. To the spline point i one adds a local refinement factor $\texttt{rp}_i$. Close to this point the mesh-size $h(x)$ is $\texttt{h}_{Glob}$ / $\texttt{rp}_i$. The global parameter `grading` describes how fast the mesh-size decreases. The gradient of the local mesh-size function $h(x)$ is bounded by $|\nabla_x h(x)| \leq \text{grading}^{-1}$ Also a refinement by a factor $\texttt{rs}_i$ ¿ 1 along the whole segment i is possible. The file looks like:

```
splinecurves2d
grading
np
x_1 y_1 rp_1
...
x_np y_np rp_np
ns
dil_1 dir_1 [ 2 | 3 ] pi_{1,1} pi_{1,2} [ pi_{1,3} ] rs_1
...
dil_nl dir_nl [ 2 | 3 ] pi_{nl,1} pi_{nl,2} [ pi_{nl,3} ] rs_nl
```

`np` is the number of points, `ns` the number of spline segments. Every segment starts with the domain numbers at the left and at the right sides of the segment. Domain number 0 is reserved for the exterior. Then the number of points and two or three point indices follow. Finally, the refinement factor along the line follows.

Chapter 4

Mesh and Solution Formats

You can export meshes to a couple of file formats. Some are self-defined, some other are standard formats. The self-defined are the followings:

4.1 Neutral Format

The neutral volume mesh format contains the following sections:

1. nodes
   After the number of nodes there follows a list of x, y, and z-coordinates of the mesh-nodes.

2. volume elements
   After the number of volume elements there follows the list of tetrahedra. Each element is specified by the sub-domain number, and 4 node indices. The node indices start with 1.

3. surface elements
   After the number of surface elements there follows the list of triangles. Each element is specified by the boundary condition number, and 3 node indices. The node indices start with 1.

4.2 Fepp Format 2D

The Fepp 2D format contains the following sections:

1. boundary segmetns
   After the number of boundary segments there follows a list of segments. Each segment is specified by the spline - patch number, and the two node indices. Counting starts with 1

2. domain elements
   After the number of domain elements there follows the list of elements. Each element is specified by the sub-domain number, the number of nodes (3 or 4) and the node indices. Counting starts with 1

3. nodes
   After the number of nodes there follows a list of x and y -coordinates of the mesh-nodes.

4. geometric information
   After the number of spline patches there follows a list of spline specifications. Each spline patch is given by the 6 coefficients of the descibing quadratic polynomial equation

$$c_1x^2 + c_2y^2 + c_3xy + c_4x + c_5y + c_6 = 0$$

4.3 Surface triangulaton file

One can export to and import from a surface mesh file. Its structure is as follows:

1. `surfacemesh`
   starts with that keyword

2. number of points
   point coordinates (x, y, z).

3. number of surface triangles,
   surface triangles oriented counter-clock wise when looking at the object, index starts from 1.

4.4 Solution File Format

By means of the solution file, one may load functions into netgen and visualize them in various manners. A solution file contains one or several blocks defining gridfunctions. Each block has the folowing structure:

1. `solution` *function-name* flags

   `solution` is the keyword, *function-name* is a string to refer to that functions. The supported flags are

   (a) -size=s
       number of entries (default: number of mesh-points)

   (b) -components=c
       number of components (default: 1). E.g., elasticity has 3

(c) -type=[nodal,element,surfaceelement]
the grid-funciton has nodal values, or one value per volume elemnet or one value per surface element

2. block of $size \times components$ values

Chapter 5

Netgen operations

5.1 Command line arguments

5.2 Meshing Options

5.3 Visualization Options

Chapter 6

The Algorithms of Netgen

Netgen follows a top down strategy. It starts from computing the corner points (CSG only). Then, the edges are defined and meshed into segments (CSG and STL). Next, the faces are meshed by an advancing front surface mesh generator. After meshing, the faces meshes are optimized. Finally, the individual sub-domains are filled with tets. Therefore, a fast Delaunay algorithm generates most of the elements (about 98 percent). But often it fails for mesh the whole domain, then the slower back-tracking rule-base algorithm takes over. Finally, the volume is optimized by the usual node - movement, element swapping and splitting algorithms.

Chapter 7

Programming Interfaces

7.1 The nginterface

By means of the nginterface one's own simulation code can be included into the netgen environment. This is particular useful for FEM (FVM,BEM) code developers, since they may profit from the netgen preprocessing and postprocessing possibilities.

Please download the example Netgen-add-on module *demoapp* and follow the instructions therein

7.2 The nglib

7.2.1 Introduction

The NETGEN mesh generation library *nglib* is available in C++ source code and can be compiled for Unix/Linux as well as Win95/98/NT and linked to one library file. The interface to the application programme is by the C language header file *nglib.h*.

The functionality of nglib is volume mesh generation by a domain given by a surface triangulation, and surface mesh generation from a domain described by an STL file (standard file format for geometries defined by triangle approximation). It can do mesh optimization as well as mesh refinement. It can generate 4 node tetrahedra and 10 node tetrahedrons (with quadratic shape functions). The local mesh size can be defined automatically by geometric features and/or by user specification.

7.2.2 The Header File

The interface file contains the following type definitions and function calls. All Netgen types and functions start with `Ng`. Types and functions have capital initial letters, constants are in capital letters.

7.2.3 Types and Constants

```
/// Data type for NETGEN mesh
typedef void * Ng_Mesh;

/// Data type for NETGEN STL geomty
typedef void * Ng_STL_Geometry;


// max number of nodes per element
#define NG_VOLUME_ELEMENT_MAXPOINTS 10

// implemented element types:
enum Ng_Volume_Element_Type { NG_TET = 1, NG_PYRAMID = 2, NG_PRISM = 3,
                              NG_TET10 = 4 };

// max number of nodes per surface element
#define NG_SURFACE_ELEMENT_MAXPOINTS 6

// implemented element types:
enum Ng_Surface_Element_Type { NG_TRIG = 1, NG_QUAD = 2,
                                NG_TRIG6 = 3 };


struct Ng_Meshing_Parameters
{
  double maxh;
  double fineness;   // 0 .. coarse, 1 .. fine
  int secondorder;
};

enum Ng_Result { NG_OK = 0,
                 NG_SURFACE_INPUT_ERROR = 1,
                 NG_VOLUME_FAILURE = 2,
                 NG_STL_INPUT_ERROR = 3,
                 NG_SURFACE_FAILURE = 4 };
```

`Ng_Mesh` is a data type representing a Netgen mesh. `Ng_STL_Geometry` represents an STL geometry. One can operate on these data structures by the functions defined below. Netgen can (now and/or in future) work with various element types defined by generic constants. Several parameters can be specified in the `Ng_Meshing_Parameters` structure for volume and/or surface mesh generation. The result of Netgen functions is of type `Ng_Result`.

7.2.4 Initialization

Please call these functions before using netgen functions and after using netgen functions, respectively:

```
// initialize, deconstruct Netgen library:
void Ng_Init ();
void Ng_Exit ();
```

7.2.5 Mesh access

Netgen meshes can be processed by the means of the following functions. A mesh contains nodes, surface elements and volume elements. Counting starts from 1.

```
// Generates new mesh structure
Ng_Mesh * Ng_NewMesh ();
void Ng_DeleteMesh (Ng_Mesh * mesh);

// feeds points, surface elements and volume elements to the mesh
void Ng_AddPoint (Ng_Mesh * mesh, double * x);
void Ng_AddSurfaceElement (Ng_Mesh * mesh, Ng_Surface_Element_Type et,
                           int * pi);
void Ng_AddVolumeElement (Ng_Mesh * mesh, Ng_Volume_Element_Type et,
                          int * pi);

// ask for number of points, surface and volume elements
int Ng_GetNP (Ng_Mesh * mesh);
int Ng_GetNSE (Ng_Mesh * mesh);
int Ng_GetNE (Ng_Mesh * mesh);


//  return point coordinates
void Ng_GetPoint (Ng_Mesh * mesh, int num, double * x);

// return surface and volume element in pi
Ng_Surface_Element_Type
Ng_GetSurfaceElement (Ng_Mesh * mesh, int num, int * pi);

Ng_Volume_Element_Type
Ng_GetVolumeElement (Ng_Mesh * mesh, int num, int * pi);
```

Mesh Generation

The user can specify the mesh size function by the global parameter maximal mesh size, and can additionally restrict the mesh size in points or cubes. The function Ng_GenerateVolumeMesh generates the volume mesh starting from the surface.

```
// Defines MeshSize Functions
void Ng_RestrictMeshSizeGlobal (Ng_Mesh * mesh, double h);
void Ng_RestrictMeshSizePoint (Ng_Mesh * mesh, double * p, double h);
void Ng_RestrictMeshSizeBox (Ng_Mesh * mesh, double * pmin, double * pmax, double h);

// generates volume mesh from surface mesh
Ng_Result Ng_GenerateVolumeMesh (Ng_Mesh * mesh, Ng_Meshing_Parameters * mp);
```

7.2.6 STL Geometry

A STL geometry can be read from a STL file (ASCII or binary), or can be assembled by providing triangle by triangle. Either, the user can specify the edges of the geometry, or netgen can define edges by Ng_STL_MakeEdges by using an angle criterium.

```
// loads geometry from STL file
Ng_STL_Geometry * Ng_STL_LoadGeometry (char * filename, int binary = 0);

// generate new STL Geometry
Ng_STL_Geometry * Ng_STL_NewGeometry ();

// fills STL Geometry
// positive orientation
// normal vector may be null-pointer
void Ng_STL_AddTriangle (Ng_STL_Geometry * geom,
                         double * p1, double * p2, double * p3, double * nv);

// add (optional) edges:
void Ng_STL_AddEdge (Ng_STL_Geometry * geom,
                     double * p1, double * p2);


// after adding triangles (and edges) initialize
Ng_Result Ng_STL_InitSTLGeometry (Ng_STL_Geometry * geom);

// automatically generates edges:
void Ng_STL_MakeEdges (Ng_STL_Geometry * geom);
```

```
  // generates mesh, empty mesh be already created.
Ng_Result Ng_STL_GenerateSurfaceMesh (Ng_STL_Geometry * geom,
                                      Ng_Mesh * mesh,
                                      Ng_Meshing_Parameters * mp);
```

7.2.7 Programming Example

The File *ngcore.cc*, see Appendix A, is a simple application using the netgen volume mesh generator. First, the surface mesh is read from a file containing point coordinates and surface triangles (see e.g. file *cube.surf*). The volume mesh generate is called, and the volume mesh is written to the standard output, see file *cube.vol*.