# Ani3D

# Advanced Numerical Instruments 3D

The package Ani3D is designated for **generating** unstructured tetrahedral meshes, **adapting** them isotropically and anisotropically, **discretizing** systems of PDEs, **solving** linear and nonlinear systems, **visualizing** meshes and associated solutions. It is a set of independent libraries with different tasks. All libraries may be combined in a single program. Extensive tutorials represent powerful capabilities of the package.

The package Ani3D was developed by a team of researchers headed by the two principle invistigators:

- Konstantin Lipnikov[1]
- Yuri Vassilevski[2].

Ideas and technologies, as well as packages Ani3D-MBA , Ani3D-LMR , Ani3D-FEM , and Ani3D-VIEW have been developed by the principal investigators.

The package Ani3D-AFT was developed by former students of the Moscow State University:

- Alexander Danilov[2] (main contribution)
- Kirill Nikitin[2]
- Anatoly Vershinin
- Andrey Plenkin

under the supervision of the principal investigators.

The packages Ani3D-RCB and Ani3D-PRJ were developed by Vadim Chugunov[2] and Yuri Vassilevski[2].

The package Ani3D-ILU was developed by

- Sergei Goreinov[2]
- Vadim Chugunov[2]
- Yuri Vassilevski[2].

The package Ani3D-INB has been developed by student of the Moscow State University

- Alexey Chernyshenko

under the supervision of the principal investigators.

Besides the original software, the package Ani3D incorporates a number of public libraries such as BLAS, LAPACK, UMFPACK, AMD, GLUT, Open CASCADE, and CGM.

The authors are grateful to Rao Garimella for mentoring Alexander Danilov and helping him with the CGM and Open CASCADE packages.

[1]Los Alamos National Laboratory Theoretical Division, MS-284 Los Alamos, NM 87545, USA.
[2]Institute of Numerical Mathematics RAS 8 Gubkina St, 119333 Moscow, RUSSIA.

## Copyright and Usage Restrictions

## Structure of our packages

After package installation, the user will get the following subdirectories

```
bin/  data/  doc/  lib/  src/  cmake/  include/
```

The executable files are always placed in `bin/`. Examples of simple meshes are located in `data/`. A PDF documentation for the package is in `doc/`. The source code is located in `src/` and the usage of the libraries is demonstrated in `src/Tutorials`. Examples of short cmake scripts are in directory `cmake/` to compile the packages with support of `GLUT` and `OpenCASCADE`. The result of installation is a set of libraries placed in `lib/` and executables placed in `bin/`.

The directory `src/` contains libraries and tutorials:

```
aniXXX/  lapack/  blas/  cgm/  Tutorials/
```

The libraries are located in various directories `aniXXX`. An incomplete versions of `LAPACK`[3] and `BLAS`[4] libraries are located in directories with the same names.

The tutorials are split into two parts

```
PackageXXX/   MultiPackage/
```

The first set of directories contains simple examples of using our libraries individually. The directory `MultiPackage` contains more complex examples using multiple packages. The main focus in `MultiPackage` is the solution of linear and nonlinear PDEs. Most of the directories are equipped with `READMEs` to help the user to navigate through the code.

[3] http://www.netlib.org/lapack
[4] http://www.netlib.org/blas

## Two alternative installation methods

We provide two methods for package installation. The first method uses `CMake`. It requires to execute the following commands:

```
$ mkdir build
$ cd build
$ cmake ../  (or one of the cmake scripts in directory cmake/)
$ make install
$ cd ..
$ ./DEMO
```

The packages will be compiled with a local copies of `LAPACK` and `BLAS`. To use system libraries, provide the following option to `CMake`:

```
$ cmake -DENABLE_SYSTEM_LAPACK:BOOL=TRUE ../
```

The second method is based on a set of simple `Makefiles`. Support of this installation method will be significantly reduced with time. In order to compile the code, the user has to set up the compilers names in `src/Rules.make` and then to execute the following commands:

```
$ make libs
$ ./DEMO
```

WARNING. Linking of the package within the second method assumes that `mergelibs` program is available. Under Ubuntu, `mergelibs` is a part of `xutils-dev` package.

WARNING. Problems with automatic linking of C and FORTRAN libraries can be found on some OS and for some compilers. The simplest solution is to skip three tests in installation of our packages by adding the following option to `CMake` command:

```
$ cd build
$ cmake -DDISABLE_C2F_TESTS:BOOL=TRUE [other options] ../
```

# Contents


**1 MESHING PACKAGEs** **9**

**Package Ani3D-AFT** **11**
1.1 Introduction . . . 12
1.2 Copyright and Usage Restrictions . . . 12
1.3 Description of Ani3D-AFT . . . 12
1.4 Generation of the initial front . . . 13
1.5 Generation of a tetrahedral mesh . . . 18
1.6 Output . . . 19

**Package Ani3D-RCB** **20**
2.1 Basic features of the library . . . 21
2.2 Initialization . . . 21
2.3 Refinement . . . 21
2.4 Coarsening . . . 23

**Package Ani3D-MBA** **25**
3.1 Introduction . . . 26
3.2 Copyright and Usage Restrictions . . . 26
3.3 Description of Ani3D-MBA . . . 26
3.4 Getting started . . . 29
3.5 Useful features of Ani3D-MBA . . . 31
3.6 How to use library libmba3D-3.1 . . . 32
3.7 Useful routines . . . 34
3.8 FAQ . . . 36

**2 DISCRETIZATION PACKAGEs** **39**

**Package Ani3D-FEM** **40**
4.1 Introduction . . . 41
4.2 Copyright Notice . . . 41
4.3 Description of Ani3D-FEM . . . 41
4.4 Examples . . . 49

**3 SOLUTION PACKAGEs** **53**

**Package Ani3D-ILU** **54**
5.1 Basic features of the library . . . 55
5.2 Iterative solution . . . 55
5.3 ILU0 preconditioner . . . 57
5.4 ILU2 preconditioner . . . 59

**Package Ani3D-INB** **63**
6.1 Basic features of the library . . . 64
6.2 Iterative solution . . . 64

**Package Ani3D-LU** **67**
7.1 Overview . . . 67

**4 SERVICE PACKAGEs** **69**

**Package Ani3D-LMR** **70**
8.1 Introduction . . . 71
8.2 Copyright and Usage Restrictions . . . 71
8.3 Description of Ani3D-LMR . . . 71
8.4 Examples . . . 74

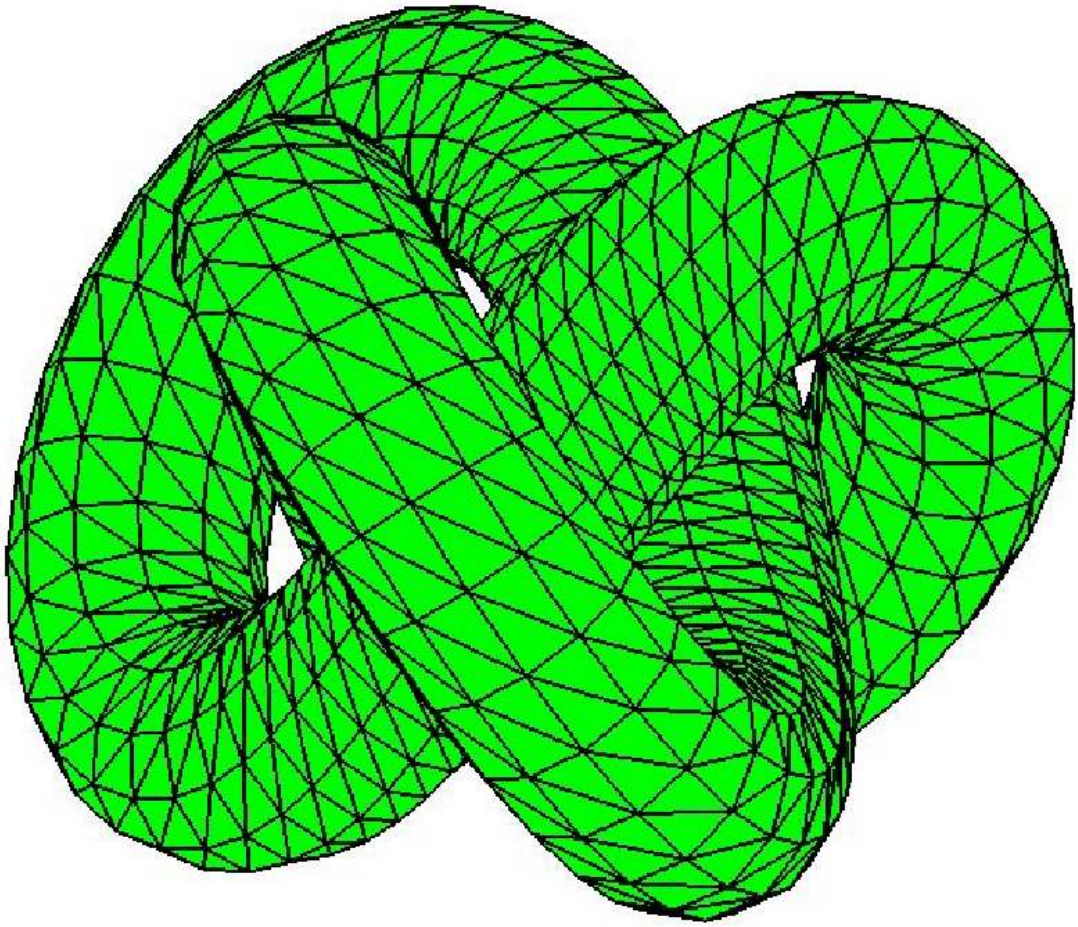
**Package Ani3D-PRJ** **75**
9.1 Basic features of the library . . . 76
9.2 Usage of the library `libprj3D-3.1` . . . 76

**Package Ani3D-VIEW** **78**
10.1 Overview . . . 78

**Package Ani3D-C2F** **79**
11.1 Requirements . . . 80
11.2 Ani3D compilation in MinGW . . . 80
11.3 Microsoft Visual C++ and Intel Visual Fortran . . . 81

# Chapter 1

# MESHING PACKAGEs



# MESH STRUCTURE

All packages use the same definition of a mesh. Hereafter, the `mesh` means either all arrays shown below or their various subsets.

```
Points:
  nv    - the actual number of mesh points
  nvmax - the maximal allowed number of mesh points
  nvfix - the number of fixed points

  vrt(3, nvmax) - the Cartesian coordinates of mesh points
  labelV(nvmax) - point identificator, a non-negative number
  fixedV(nvmax) - a list of fixed points


Faces:
  nb    - the actual number of boundary and interface faces
  nbmax - the maximal number of boundary and interface faces
  nbfix - the number of fixed faces

  bnd(3, nbmax) - connectivity list of boundary and interface faces
  labelB(nbmax) - boundary face identificator, a positive number
  fixedB(nbmax) - a list of fixed faces


Tetrahedra:
  nt    - the actual number of tetrahedra
  ntmax - the maximal number of tetrahedra
  ntfix - the number of fixed tetrahedra

  tet(4, ntmax) - connectivity list of tetrahedra
  labelT(ntmax) - tetrahedron identificator, a positive number
  fixedT(ntmax) - a list of fixed tetrahedra
```

# Ani3D-AFT version 3.1 "*Forget-me-not*"

# Flexible Tetrahedral Mesh Generator Using Advanced Front Technique

## User's Guide for libaft3D-3.1.a, libfrtprm-3.1.a, libfrtmdf-3.1.a, libfrtscg-3.1.a, libfrtcad-3.1.a

## 1.1 Introduction

The C package Ani3D-AFT is an independent part of the package Ani3D . Ani3D-AFT was developed by Alexander Danilov, Kirill Nikitin, Anatoly Vershinin and Andrey Plenkin under the supervision of Yuri Vassilevski and Konstantin Lipnikov. It is designated for generating tetrahedral meshes in arbitrary 3D domains. The libraries of the package Ani3D-AFT are split into two subsets by their objectiveis: to generate an initial front as a boundary discretization *(libfrtprm-3.1.a, libfrtmdf-3.1.a, libfrtscg-3.1.a, libfrtcad-3.1.a)* and to generate a tetrahedrization with the given boundary trace *(libaft3D-3.1.a)*.

The libraries are designed to be incorporated easily into other packages.

This document describes the structure of the package, input data, and user-supplied routines. It presents a few examples illustrating details of the package.

## 1.2 Copyright and Usage Restrictions



## 1.3 Description of Ani3D-AFT

The basic features of package Ani3D-AFT are listed below.

**Domain type** : single-/multi-component and simply-/ multi-connected bounded domains

**Boundary type** : piecewise smooth or polyhedral (piecewise linear)

**Domain data input** : set of 2D patches representing the boundary, or the front (triangulation of the boundary)

**Number of mesh elements** : non-limited

**Data format** : double precision and integer arrays. Enumeration starts with 1 (default).

The library *libaft3D-3.1.a* uses the Advanced Front Technique for 3D meshing. The technique presumes that an initial front defining the boundary is given. The initial front is a surface shape-regular triangulation with a prescribed orientation of normals for all triangles. The orientation defines the direction of the front advancing. The initial front may be composed of multiple components. The local mesh size may be controlled by the user defined function `fsize`.

The libraries *libfrtprm-3.1.a, libfrtmdf-3.1.a, libfrtscg-3.1.a, libfrtcad-3.1.a* use different technologies for the generation of the initial front. These technologies are listed below:

1. Representation of the boundary as a union of smooth parameterized patches with explicit parameterizations of their boundaries (if patches are flat polygons then parameterization of their boundaries is not needed). An automated triangulation of the boundary yields the initial front. The algorithms are in the library *libfrtprm-3.1.a*.

2. Representation of the initial front as unions and intersections of prescribed surface meshes of the given primitives. The primitives may be defined by the user via templated routines. The surface meshes are produced by the automated surface triangulation using algorithms from library *libfrtscg-3.1.a*.

3. Representation of the boundary as a conformal triangulation approximating the actual boundary with a given accuracy (a CAD mesh). The triangles (facets) may be anisotropic or severely distorted. An automated remeshing of the CAD mesh such that the new mesh nodes belong to the CAD surface, yields the initial front. The algorithms are in the library *libfrtmdf-3.1.a*.

4. Representation of the boundary using a public CAD system (Open CASCADE) and the public interface library CGM. The algorithms in library *libfrtcad-3.1.a* provides the interface to technology 1.

The package is designed so that no graphical interface is needed for the front and 3D mesh generation. The mesh and the front may be saved in files with the GMV-format (`write_mesh_gmv`) and then visualized and analyzed with the GMV package.

However, for user convenience, one graphical user interface (GUIs) is added for the second tool (`src/aniAFT/src/aniFRT/GUI/SCG`). It visualizes operations with primitives using the *freeglut* package. An example of using the GUI is given in `src/aniAFT/src/aniFRT/GUI/SCG/gui_glut.cpp`. A description of GUI parameters is given in `src/aniAFT/src/aniFRT/GUI/SCG/README`.

If the user installed Open CASCADE library, he can use its GUI executable `DRAWEXE`. A description of GUI parameters is given in `src/aniAFT/src/aniFRT/GUI/CAD/README`.

## 1.4 Generation of the initial front

### 1.4.1 Boundary as a union of smooth parameterized patches (library libfrtprm-3.1.a)

The library contains three routines which may be used for the initial front generation:

```
set_param_functions(surface_param, v_u_param);
set_surface_size_function(fsize);
i = surface_boundary_( ... );
```

Routine `set_param_functions` adopts the user given parameterizations of the patches `surface_param()`, and their boundaries `v_u_param()`. Routine `set_surface_size_function` adopts the user given mesh size function `fsize()`. Routine `surface_boundary_` produces

a shape regular triangulation of the boundary with a mesh size controlled by `fsize()`. An example using these routines is given in file `src/Tutorials/PackageAFT/main_prm.c`. The initial boundary patches and the constructed surface mesh are shown in Fig 1.1.

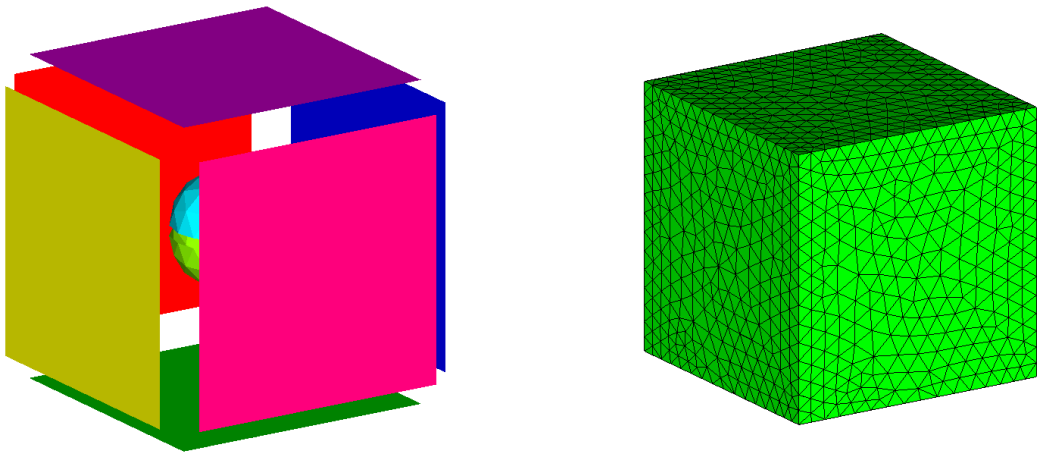


Figure 1.1: The initial boundary patches (left) and the final surface mesh (right).

If the boundary is a union of flat polygons, the user is advised to use a wrapper `polyhedron_make_front` to `surface_boundary_` which uses the polygons data. An example of this technique is given in file `src/Tutorials/PackageAFT/main_prm_flat.c`. The user is advised to replace `main_prm.c` with `main_prm_flat.c` and recompile it.

### 1.4.2 Boundary as a combination of given front primitives (library libfrtscg-3.1.a)

The library contains routines which operate with initial fronts. They generate surface meshes for primitives (parallelepiped, sphere or cylinder), move, rotate and scale them, and generate a surface mesh for the intersection/union/difference of the primitives.

```
/* Surface mesh generation for the sphere with radius 1.45;
   mesh step is 0.3 */
scg_make_sphere (&nV_in1, &nF_in1, Vertex_in1, Index_in1, 1.45, 0.3);


/* Surface mesh generation for the cylinder with radius 0.5,
   height 5; mesh step is 0.3 */
scg_make_cylinder (&nV_in2, &nF_in2, Vertex_in2, Index_in2, 0.5, 5, 0.3);


/* Translation of an object along vector (0.1, 0.5, -2.3) */
scg_translate (nV_in2, Vertex_in2, 0.15, 0.5, -2.3);
```

```
/* Rotation of an object by angles pitch=0.2, yaw=0.1 and roll=0 */
scg_rotate (nV_in2, Vertex_in2, 0.2, 0.1, 0);


/* Intersection of meshes for two objects: the last integer
   parameter is an operation:
     0 - union (in1 + in2)
     1 - difference (in1 - in2)
     2 - another difference (in2 - in1)
     3 - intersection (in1 * in2) */
scg_intersect_mesh (nV_in1, nF_in1, Vertex_in1, Index_in1,
                    nV_in2, nF_in2, Vertex_in2, Index_in2,
                    &nV_out, &nF_out, Vertex_out, Index_out, 2);
```

An example of using these routines is in file `src/Tutorials/PackageAFT/main_scg.cpp`. Different types of boolean operations with primitives are shown in Fig 1.2.

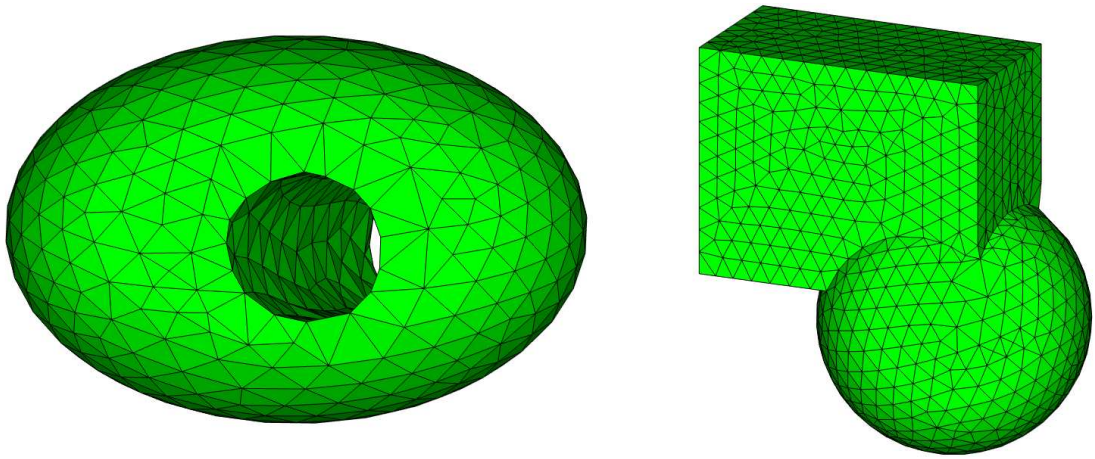


Figure 1.2: The difference of a scaled sphere and a cylinder (left) and the union of a box and a sphere (right).

The user can visualize and control the process of meshing. To this end, a Graphical User Interface is provided in `src/aniAFT/src/aniFRT/GUI/SCG`. It is based on library *libfrtscg-3.1.a* and the public library *freeglut*. The user can operate with the primitives using keyboard and input string and load and save the mesh front. An example of use of the GUI is given in file `src/aniAFT/src/aniFRT/GUI/SCG/gui_glut.cpp`. In order to create a 0.3-mesh for sphere with radius 1.6, centered at (0, 0, 1) and rotated by angles 0, 0, 0 and a 0.3-mesh for cylinder with radius 0.6, height 5, centered at (0, 0, -1) and rotated by angles 0.3, 0 and 0.5, type

```
bin/gui_glut.exe +s 1.6 0 0 1 0 0 0 0.3 +c 0.6 5 0 0 -1 0.3 0 0.5 0.3
```

Then use the keyboard to generate the surface mesh of a combination of the two primitives.

### 1.4.3 Boundary as a CAD mesh with triangular facets (library libfrtmdf-3.1.a)

The library contains three routines which convert the input CAD mesh to a shape-regular mesh:

```
check_surface_topology_fix_(&nV, vertex, &nF, face);
surface_refine_setup_poly(0.10, 0.10, 1e-6, 1e-6);
surface_refine_setup_cf(1.2);
surface_refine_(&nV, vertex, &nF, face, facematerial, &nnV, &nnF);
```

Routine `check_surface_topology_fix_` checks the topology of the CAD mesh and merges possible duplicate vertices. Routine `surface_refine_setup_poly` sets control parameters for `surface_refine`. Their meaning is described in `src/Tutorials/PackageAFT/main_mdf.c`. Routine `surface_refine_setup_cf` sets coarsening factor for new surface mesh. Default value for surface coarsening factor is 1.5. Routine `surface_refine_` refines the CAD mesh by generating a shape regular triangulation which approximates the boundary with the same accuracy as the CAD mesh. The nodes of the shape-regular triangulation are on the CAD mesh.

An example using these routines is in file `src/Tutorials/PackageAFT/main_mdf.c`. The initial and final surface meshes are shown in Fig 1.3.

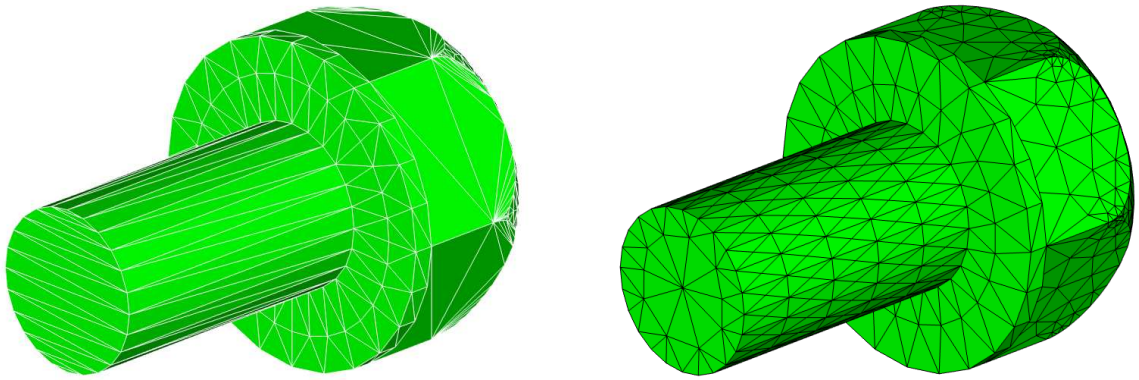


Figure 1.3: The initial surface mesh (left) and the final surface mesh (right).

### 1.4.4 Boundary representation through a CAD system (library libfrtcad-3.1.a)

The library libfrtcad-3.1.a generates the initial front using the public CAD system Open CASCADE (http://www.opencascade.org) through the interface library CGM (Common Geometry Module). The library may be used only if Open CASCADE is properly installed

before the compilation of Ani3D-AFT. The interface between AFT kernel and CAD system is based on CGM (http://sigma.mcs.anl.gov/cgm-library/). Ani3D-AFT uses a modified version of CGM-15.1 with addition of CMakeLists files for Ani3D compatibility. This version is located in `src/cgm`. It requires at least Open CASCADE version 6.5. We recommend Open CASCADE versions 6.9.x and 6.7.x. We noticed some stability regressions with Open CASCADE versions 6.5.x and 6.8.0. Note, that Open CASCADE versions 7.x are not yet supported by CGM version 15.1.

In order to compile CGM and use Open CASCADE support in Ani3D-AFT , environment variable `$CASROOT` should be defined. It should point to the directory where the Open CASCADE has been installed. For CGM installation, it is assumed that the target of `$CASROOT` is the directory which includes subdirectories either inc, lib, or ros/inc, ros/lib. If the installed version of Open CASCADE does not have such a directory or does not define `$CASROOT` (it is the case of installation via package manager in Ubuntu Linux), we recommend to define it as follows: mkdir opencascade (elsewhere, e.g. in home directory of Ani3D )

```
mkdir -p opencascade/ros
ln -s /usr/lib opencascade/ros/lib
ln -s /usr/include/opencascade opencascade/ros/inc
ln -s /usr/share/opencascade/6.5.0/src opencascade/ros/src  (optional)
export CASROOT=PATH_TO_opencascade/ros or
define CASROOT in your .bashrc ("export CASROOT=PATH_TO_opencascade/ros")
```

If `$CASROOT` variable is unset, CGM will not be compiled, and Open CASCADE support will be disabled. If `$CASROOT` variable is set, CGM will be compiled, and library `libfrtcad-3.1.a` (source code is in `src/aniAFT/src/aniFRT/CAD`) will be built.

Routine `CGM_Init` initializes the CGM interface. Routine `CGM_LoadModelFromFile` loads the CAD input data (*.brep) and results in Open CASCADE object 'model'. Routine `surface_CGM_model_(model,..)` produces a shape regular triangulation of the boundary of 'model'.

```
// initialize CGM
CGM_Init();

// load CGM model from file
model = CGM_LoadModelFromFile(argv[1]);

// allocate memory
vertex        = (double*)malloc(sizeof(double) * 3 * nnV);
face          = (int*)   malloc(sizeof(int)    * 3 * nnF);
facedup       = (int*)   malloc(sizeof(int)    * 3 * nnF);
facematerial  = (int*)   malloc(sizeof(int)        * nnF);
facematdup    = (int*)   malloc(sizeof(int)        * nnF);
tetra         = (int*)   malloc(sizeof(int)    * 4 * nnT);
tetramaterial = (int*)   malloc(sizeof(int)        * nnT);
```

```
// this is the relative mesh size for the front
surfacesizeratio = 0.1;

// create initial front
surface_CGM_model_(model, &nV, vertex, &nF, face, facematerial, &nnV, &nnF);
```

An example is given in file `src/Tutorials/PackageAFT/main_cad.c`. The initial model and the final surface mesh are shown in Fig 1.4.

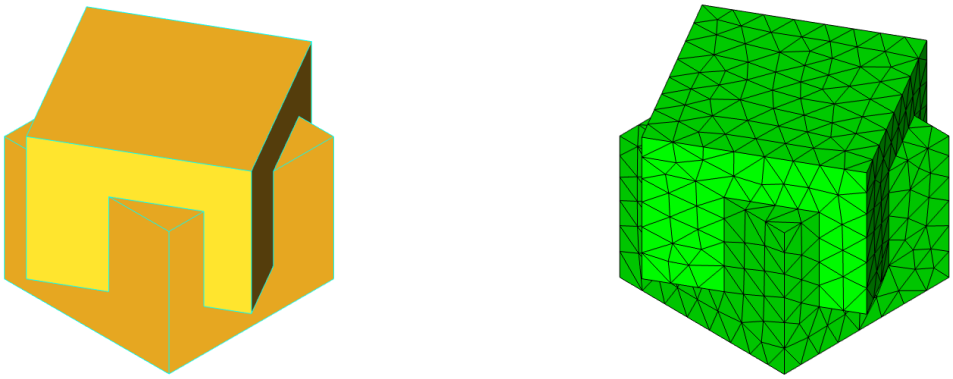


Figure 1.4: The initial model in Open CASCADE (left) and the constructed surface mesh (right).

There exists the GUI for the Open CASCADE library. The executable `DRAWEXE` allows the user to visualize the process of production of the CAD input data (*.brep). A brief tutorial for this GUI may be found in `src/aniAFT/src/aniFRT/GUI/CAD/README`. The executable assumes that shared source directory of Open CASCADE is available at `$CASROOT/src`. In the case of Open CASCADE installation via package manager in Ubuntu Linux `$CASROOT/src` should be symlinked to `/usr/share/opencascade/6.3/src`.

## 1.5 Generation of a tetrahedral mesh

After an initial front is defined, the user may call one of two 3D meshing routines

```
i = mesh_3d_aft_func_( ..., fsize );
i = mesh_3d_aft_cf_( ..., &cf );
```

Routine `mesh_3d_aft_func_` generates a shape regular tetrahedrization with a local mesh size controlled by the user defined function `fsize`. Routine `mesh_3d_aft_cf_` generates a shape regular tetrahedrization which is coarsened towards the domain interior with the geometric factor `cf`. The second routine is useful when the local mesh size is unknown,

(e.g., if the input is a CAD mesh). Both routines return the error code: zero value means the mesh is generated successfully, other values imply generation failure.

Examples using these routines are given in files `src/Tutorials/PackageAFT/main_aft.c`, `src/Tutorials/PackageAFT/main_prm.c`, `src/Tutorials/PackageAFT/main_mdf.c`, and `src/Tutorials/PackageAFT/main_scg.cpp`.

## 1.6 Output

Two library routines allow to save mesh:

```
write_mesh    ("mesh.out", ... );
write_mesh_gmv("mesh.gmv", ... );
```

Routine `write_mesh_gmv` generates the GMV-file `mesh.gmv`. The General Mesh Viewer (GMV) is a public visualizing tool available at *www-xdiv.lanl.gov/XCM/gmv/GMVHome.html.*

Routine `write_mesh` saves the mesh in file `mesh.out` using a simple format:

```
nV
x_1 y_1 z_1
...
x_nV y_nV z_nV
nT
i1_1 i2_1 i3_1 i4_1 tetramaterial_1
...
i1_nT i2_nT i3_nT i4_nT tetramaterial_nT
nF
j1_1 j2_1 j3_1 facematerial_1
...
j1_nF j2_nF j3_nF facematerial_nF
```

The output mesh is not necessarily shape regular. To get rid of bad shape elements, one can apply a postprocessing tool `mbaFixShape` provided by the library Ani3D-MBA -3.1. For details refer to section *Useful routines* of the Ani3D-MBA -3.1 section of this user-guide document. In particular, the user can use `src/Tutorials/PackageMBA/mainFixAftMesh.f` program in order to apply mesh cosmetics to the output mesh file `mesh.out`.

# Ani3D-RCB version 3.1 "*Windflower*"

# Flexible Mesh Refining/Coarsening Tool Using Marked Edge Bisection

# User's Guide for librcb3D-3.1.a

## 2.1 Basic features of the library

The FORTRAN77 package Ani3D-RCB is a part of the package Ani3D . Ani3D-RCB was developed by Vadim Chugunov and Yuri Vassilevski. It is designated for hierarchical refining and coarsening of arbitrary tetrahedral meshes. Basic restriction: prior coarsening the mesh must be refined; no coarsening is applied to an unrefined mesh.

The library contains an initialization tool, a refinement tool, and a coarsening tool. An example of using the package is in file `src/Tutorials/PackageRCB/main.f`.

### Mesh data

The mesh output is produced in place of the mesh input. A mesh is represented by the following data. The number of mesh nodes is `nv`, their Cartesian coordinates are stored in the array `vrt(3,*)`. The number of mesh tetrahedra is `nt`, the connectivity list of tetrahedra is stored in the array `tet(4,*)`, the tetrahedron materials (labels) are `material(*)`. The number of mesh boundary faces is `nb`. The columns of `bnd(3,*)` are node indexes of the boundary faces. Face material/label is stored in `labelF(*)`.

## 2.2 Initialization

The initialization tool (`aux.f`) prepares auxiliary structure which defines how to bisect the tetrahedra. In `InitializationRCB` all input tetrahedra are marked for bisection according to specific rule. Also, auxiliary data structure is generated in this routine. In actual refinement, the user is free to mark for refinement any subset of tetrahedra.

```
      iERR = 0
      call InitializeRCB(nt, ntmax, nv, nvmax, vrt, tet,
     &                   MaxWi, iW, listtet, tetpmax, iERR)
      If(iERR.GT.0) Call errMes(iERR, 'main', 'error in InitializeRCB')
```

The size of work memory `iW` for `InitializeRCB, LocalRefine, LocalCoarse` should be at least `15*ntmax+nvmax+41`. The size of data array `listtet` is `(tetpmax+1)*ntmax` where `tetpmax=50`.

## 2.3 Refinement

The refinement tool `LocalRefine` (`refine.f`) refines the input tetrahedrization according to the user defined routine `RefineRule`. The name of the latter routine is the input parameter of `LocalRefine`. The output tetrahedrization is in place of the input tetrahedrization. The by-product of `LocalRefine` is the logical data array `history(4,maxlevel,ntmax)`. It will be used in later coarsening. The input current index of the refinement level `ilevel` is passed to `RefineRule`.

```
c ... user defined procedures
      external  RefineRule
```

```
...
nlevel = 7
Do ilevel = 1, nlevel
    call LocalRefine (
&          nv, nvmax, nb, nbmax, nt, ntmax,
&          vrt, tet, bnd, material,labelF,
&          RefineRule, ilevel,
&          maxlevel, history,
&          listtet, tetpmax, RefineRuleData,
&          MaxWi, iW,
&          iPrint, iERR)
     If(iERR.GT.0) Call errMes(iERR, 'main',
&                 'error in LocalRefine')
End do
```

The key control of the refinement process is the user defined routine `RefineRule`. Here, the user defines which tetrahedra have to be refined and how they must be refined, depending on each tetrahedra data and the current level of refinement. The array `RefineRuleData` is served for passing external data to the routine. The control for refinement is the marker `verf(i)`, where `i` runs from 1 to `nt`. If the marker is 0, then there is no need to refine tetrahedron `i`; if the marker is 1, then the user wants to refine tetrahedron by single bisection; if the marker is 2, then the user wants to refine tetrahedron by two levels of bisection; if the marker is 3, then the user wants to refine tetrahedron by three levels of bisection into eight similar subtetrahedra.

```
Subroutine RefineRule (nt, tet, vrt, verf, ilevel, RefineRuleData)

...
If (ilevel .le. 3) Then
   Do i = 1, nt
      verf(i) = 1  ! one level of bisection
   End do
Else  ! refine towards the plane y=0
   Do i = 1, nt
    xy = vrt(3,tet(1,i))* vrt(3,tet(2,i))*
&   vrt(3,tet(3,i))* vrt(3,tet(4,i))

      If (xy .eq. 0) Then
         verf(i) =  3  ! three levels of bisection (keep the shape)
      Else
         verf(i) =  0  ! no need to refine
      End if
   End do
End if
End
```

The example of application of the above procedure is shown in Fig.2.5

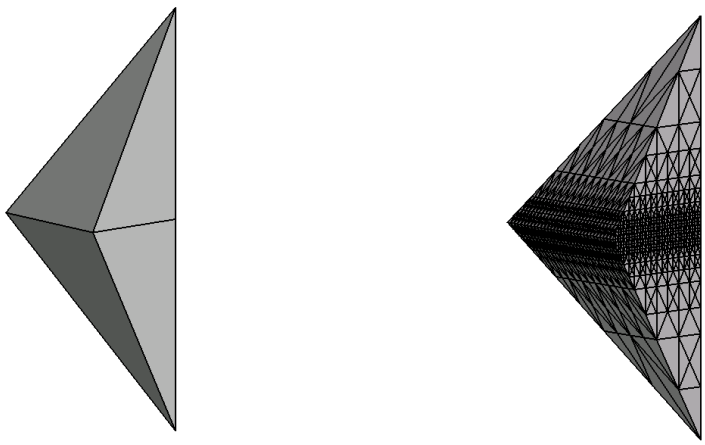


Figure 2.5: Initial mesh and locally refined mesh.

## 2.4 Coarsening

The coarsening tool `LocalCoarse` (`coarse.f`) coarses the input tetrahedrization according to the user defined routine `CoarseRule`. The name of the latter routine is the input parameter of `LocalCoarse`. The output tetrahedrization is in place of the input tetrahedrization. The array `CoarseRuleData` is served for passing external data to the routine. The by-product of `LocalCoarse` is the logical data array `history(4,maxlevel,ntmax)`. It will be used in later coarsening/refinement. The input current index of the refinement level `ilevel` is passed to `CoarseRule`.

```
external  CoarseRule
...
nlevel = 5
Do ilevel = nlevel, 1, -1
    Call LocalCoarse (
&        nv, nvmax, nb, nbmax, nt, ntmax,
&        vrt, tet, bnd, material,labelF,
&        CoarseRule, ilevel,
&        maxlevel, history,
&        listtet, tetpmax, CoarseRuleData,
&        MaxWi, iW,
&        iPrint, iERR)
     If(iERR.GT.0) Call errMes(iERR, 'main', 'error in LocalCoarse')
End do
```

The key control of the coarsening process is the user defined routine `CoarseRule`. Here, the user defines which tetrahedra have to be merged and how they must be merged, depending on each tetrahedron data and the current level of coarsening. The control for coarsening is the marker `verf(i)`, where `i` runs from 1 to `nt`. If the marker is 0, then there is no need to coarse tetrahedron `i`; if the marker is 1, then the user wants to merge tetrahedron with its neighbor; if the marker is 2, then the user wants to merge tetrahedron with its neighbor and then merge the result one more time; if the marker is 3, then the user wants to merge tetrahedron with its neighbor and then merge the result to more times so that the result be similar to tetrahedron `i`.

```
      Subroutine CoarseRule (nt, tet, vrt, verf, ilevel, CoarseRuleData)
      ...
c Coarsening conjugate to refinement rule
      If (ilevel .le. 3) Then
        Do i = 1, nt
          verf(i) =  1 ! one level  of merging
        End do
      Else ! coarse tets touching plane y=0
        Do i = 1, nt
          xy = vrt(3,tet(1,i))* vrt(3,tet(2,i))*
     &    vrt(3,tet(3,i))* vrt(3,tet(4,i))

          If (xy .eq. 0) Then
            verf(i) =  3 ! three levels of merging (keep the shape)
           Else
            verf(i) =  0 ! no need to coarse
          End if
        End do
       End if
       End
```

# Ani3D-MBA version 3.1 "*Stone Flower*"

# Flexible Mesh Generator Using Metric Based Adaptation

# User's Guide for libmba3D-3.1.a

## 3.1 Introduction

The Fortran package Ani3D-MBA (3D metric based adaptation) is a part of the package Ani3D . It has been developed by Konstantin Lipnikov and Yuri Vassilevski. Package Ani3D-MBA generates conformal tetrahedral meshes which are quasi-uniform in a given metric. The metric may be defined either explicitly, via an analytical formula, or implicitly, via a discrete Hessian or an edge-based error estimate recovered from a user-supplied mesh function (usually, a mesh solution). In the first case, the user may generate a mesh with desirable properties. In the second case, the resulting mesh is adapted to the given mesh function enabling a better resolution of sharp changes in the solution.

The library *libmba3D-3.1.a* can be incorporated into other packages.

The input data for our generator is an initial conformal tetrahedrization. It may be a very coarse mesh consisting of a few tetrahedra (for example, made by hands), or a very fine mesh produced by another mesh generator. Ani3D-MBA *changes* the initial mesh through a sequence of local modifications. This approach provides a stable algorithm for generating strongly anisotropic grids.

This document describes the structure of the package, input data, and user-supplied (optional) routines. It explains how the user can control the mesh generation process.

## 3.2 Copyright and Usage Restrictions

This software is released under the GNU LGPL Licence. You may copy and use this software without any charge, provided that the `COPYRIGHT` and `LICENSE` files are attached to all copies.

This software is available "as is" without any assurance that it will work for your purposes. The developers are not responsible for any damage caused by using this software.

## 3.3 Description of Ani3D-MBA

The main objective of Ani3D-MBA is to produce a mesh with a prescribed number of tetrahedra which is as much quasi-uniform in a given tensor metric as possible. For example, when the metric is isotropic and constant, Ani3D-MBA may generate a mesh consisting of shape-regular tetrahedra if the model geometry allows this. A measure of quasi-uniformity is a positive number between 0 and 1 which is called the *mesh quality*. The mesh with a prescribed number of equilateral tetrahedra of the same size (measured in the given metric) has quality 1.

### 3.3.1 Structure of the package

The main Fortran 77 subroutines of Ani3D-MBA are *mbaAnalytic* and *mbaNodal* located in files `mba_analytic.f` and `mba_nodal.f`, respectively. The depending subroutines are contained in the other files in directory `src/aniMBA`. The examples using these subrotuines are in directory `src/Tutorials/PackageMBA`. The files

`src/Tutorials/PackageMBA/main*.f` `src/aniMBA/time.c`

may be modified by the user.

The files

```
main_analytic.f    main_nodal.f
```

show how to call two main routines of the package: *mbaAnalytic* and *mbaNodal*. The program in file `main_analytic.f` generates a mesh using an analytic (isotropic) metric. The program in file `main_nodal.f` uses a user-supplied solution defined at mesh nodes to generate an adapted mesh. File `time.c` is a wrapper for the system call *times* that returns the CPU time. Generally speaking, this routine depends on the operational system.

For user convenience, package Ani3D-MBA is equipped with auxiliary files

```
loadM.f        saveM.f
```

Their purpose is to facilitate loading and saving meshes. For visualization purposes, a simple service library *libview3D-3.1.a* was created. Routine *drawMesh* from *libview3D.a* draws five cuts of the mesh by $XY$-planes. This provides a quick overview of the mesh. For more details, we recommend to use routine *drawGMV* which saves the mesh in the GMV format. To view the `*.gmv` file, the user has to install the General Mesh Viewer from www-xdiv.lanl.gov/XCM/gmv/GMVHome.html. The program in file

```
main_read_aft_mesh.f
```

loads a mesh (data/aft.out) generated by the alibaft3D library (using the AFT format), and saves it in two files: save.ani (using the MBA format) and save.gmv (using the GMV format).

Other two main programs in directory `src/Tutorials/PackageMBA` show additional functionality of the package. The program in file

```
main_refine.f
```

demonstrates how the user can refine heirarchically an input conformal mesh (uniformly in whole domain, and locally, in connected subdomains).

The program in file

```
main_fixshape.f
```

shows how the user can apply mesh "cosmetics" procedure `mbaFixShape` in order to improve shape quality of the input mesh. This program loads a mesh (`mesh.out`) generated by the alibaft3D library (using the AFT format), fixes its quality and saves back in AFT format (`mesh.out`) and in GMV format (`mesh.gmv`) If file `mesh.out` is not present in the current directory, the program downloads `../data/aft.out` and saves the fixed mesh to `save.out`.

The retired (but still working) file

```
Makefile
```

builds the package under Linux. The executable programs are put in directory `bin`. The names for compilers are defined in `src/Rules.make`. A few examples of input meshes may be found in directory `data`. This document and other documentation related to the package Ani3D are located in directory `doc`.

### 3.3.2 Basic things the user should know

The package provides two major methods for mesh generation. The first method is based on a piecewise linear interpolant of the user-defined metric (see Sec. 3.6). The second method is based on a piecewise linear metric recovered automatically from the user-supplied mesh function. This mesh function is defined at mesh nodes. If such a mesh function is not available, the package contains a few routines for accurate interpolation of mesh functions defined on edges or tetrahedra to nodes (see Sec. 3.7).

The package is encapsulated in the two basic routines *mbaAnalytic* and *mbaNodal* corresponding to the above methods. The comments in file `mba_nodal.f` are worth to read! After understanding what are input and output data for each of the methods, the user may find more details in files `main_analytic.f` and `main_nodal.f` located in directory `src/Tutorials/PackageMBA`.

### 3.3.3 Input data

The input data may be split into three types: data files, user input routines (optional) and control parameters.

- The *data files* are the files containing coordinates of mesh nodes, connectivity tables for tetrahedra and *boundary* faces, and lists of fixed nodes, faces and cells. The lists of fixed nodes, faces and elements may be empty. The list of boundary faces may be also empty. In this case, the boundary faces are recovered by package routines. A good example illustrating format of the data file is `data/cube.ani`. A data file can be accessed via routine *loadMani*.

  The mesh loader *loadMani* understands the format of input data files `*.ani` located in directory `data`. For other formats, a new mesh loader has to be written by the user.

  Depending of the mesh generation method, in addition to mesh data, nodal values of a mesh function have to be provided. It can be done by the function loader *loadS* located in file `src/aniMBA/loadM.f`.

- The *user input routines* are the Fortran 77 routines describing the analytical space metric. Example of such a routine is in the file `src/aniMBA/forlibmba.f` and in the end of file `src/Tutorials/PackageMBA/main_analytic.f`. Note that the analytical metric is required only for routine *mbaAnalytic*. For more detail, we refer to comments in the above files.

- The *control parameters* are the input parameters that control the mesh generation. They are defined in files `main_analytic.f` and `main_nodal.f`. These files are in directory `src/Tutorials/PackageMBA`. The input control parameters are the following variables:

```
nEstar   - [integer] the desired number of tetrahedra
Lp       - [real*8]  the norm in that the error is minimized
MaxQItr  - [integer] the maximal number of local grid modifications
Quality  - [real*8]  the desired quality for the final grid
```

```
                              (a positive number between 0 and 1)
    MaxSkipE - [integer] the maximal number of skipped tetrahedra
```

The mesh generation is an iterative process every step of which is a local modification of the current mesh. The stopping criteria for the iterative process is either the final mesh quality (`Quality`) or the allowed number of local modifications (`MaxQItr`). We recommend to set `Quality` to a value between 0.3 and 0.5 and to choose `MaxQItr` to be several times bigger than `nEstar`. We also recommend to set `MaxSkipE` (an interior parameter for the iterative process) to the default value which is about 500.

**Mesh format**

Understaning details of the mesh format is one of the first steps in discovering capabilities of package Ani3D-MBA . The mesh presentation includes:

```
nP - [integer] the number of points
nF - [integer] the number of boundary and interface faces
nE - [integer] the number of tetrahedra

XYP(3, *) - [real*8]  the Cartesian coordinates of mesh points
IPE(4, *) - [integer] connectivity list of tetrahedra
lbE(*)    - [integer] material indentificator (a positive number)

IPF(3, *) - [integer] connectivity list of boundary and interior faces
lbF(*)    - [integer] boundary identificator (a positive number)

nPv - [integer] the number of fixed points
nFv - [integer] the number of fixed faces
nEv - [integer] the number of fixed tetrahedra

IPV(*) - [integer] list of fixed points
IFV(*) - [integer] list of fixed faces
IEV(*) - [integer] list of fixed tetrahedra
```

Since some of the mesh data may be empty lists, the minimal mesh representation may contain only `nP`, `nE`, `XYP`, `IPE` and `lbE`.

For historical reasons, the notations used in this and a few other packages differ from that introduced on page 10. We are working gradually to mitigate this inconsistency.

## 3.4 Getting started

After package installation, the user will get the following subdirectories

```
bin/  data/  doc/  include/  lib/  src/
```

By default, the executable files are stored in `bin/`. A few example of input files are located in `data/`. A documentation for the package may be found in `doc/`. The source code is

stored in `src/aniMBA/`. The examples are in directory `src/Tutorials/PackageMBA/`. In order to compile the code, the user has to set up the compilers names in `scr/Rules.make` and then to execute the following commands:

```
$make libs
$cd src/Tutorials/PackageMBA
$make help exe
```

The user may change the names and options for compilers in file `src/Rules.make`. After the successful compilation, the user may run one of the executables in `bin/`. The same task can be accomplished with `make run-ana` or `make run-nod`. The output may look like:

```
$ cd bin; ./mbaAnalytic.exe

The mesh has   1536 tetrahedra
Saving GMV image prism.gmv

MBA: STONE FLOWER! (1997-2017), version 3.1
     Target: Quality=0.40 (nEStar:   2000, SkipE:  300, maxITR:   50000)

Avg Quality is 0.3596E-01,  Maximum R/r = 0.3687E+01,  status.fd:    9
ITRs:    118 Q=0.1789E-02  #V#F#T:    523     512     1890  tm=   0.0s
Avg Quality is 0.3596E-01,  Maximum R/r = 0.3687E+01,  status.fd:    9
ITRs:  24783 Q=0.2306E+00  #V#F#T:    836     948     3242  tm=   5.0s
Avg Quality is 0.5462E+00,  Maximum R/r = 0.2369E+02,  status.fd:    9
Total: 24783 Q=0.2306E+00  #V#F#E:    836     948     3242  tm=   5.0s

Saving GMV image save.gmv
```

The initial and final meshes for a triangular prism are shown in Fig 3.6.
First, some of the control parameters and average mesh quality are printed. Then, the quality of the current mesh and the numbers of points, faces and tetrahedra are printed. Additional output goes into GMV files `prism.gmv` and `save.gmv` containing initial and final meshes, respectively. The files are located in directory `bin`. One way to check the contents of these files is to execute `gmv -i prism.gmv`.

The program creates a coarse prismatic mesh with 3 cells and refines it hierarchically 3 times. To run a different example, the user may use the following mesh loader:

```
      Call loadMani(
     &      MaxP, MaxF, MaxE, nP, nF, nE,
     &      XYP, IPF, IPE, lbF, lbE,
     &      nPv, nFv, nEv, IPV, IFV, IEV,
     &      iW, iW, "../data/ramp.ani")
```

The user may try to change the input control parameters in file `main_analytic.f` and/or the analytical metric in function `MetricFunction_user` located in the same Fortran file. For instance, changing the metric the user will learn how to control the shape of tetrahedra.

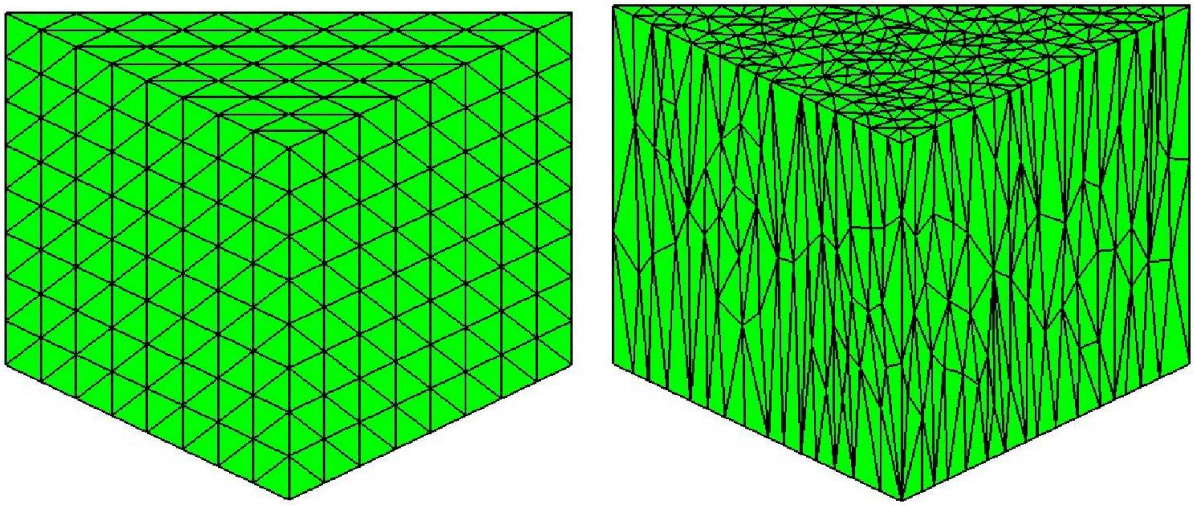


Figure 3.6: The initial structured mesh (left) and the final anisotropic mesh (right).

## 3.5 Useful features of Ani3D-MBA

We continuously improve robustness and efficiency of the code, make it more user friendly and add new features in each release. The most important features are listed below:

1. It is possible to produce meshes minimizing different $L^p$-norms of the interpolation. The input non-negative parameter `Lp` defines this norm. In the special case `Lp = 0`, the maximum norm is used.

2. The complete list of available features is in file `src/aniMBA/status.fd`. Here are the most important features:

   - The user may freeze boundary points. This allows to preserve important geometric features for both isotropic and anisotropic metrics.
   - The user may freeze boundary faces and/or mesh elements. This allows to preserve mesh structure in important regions (e.g., in boundary layers).
   - The interfaces between materials with different labels (lbE) are recovered and preserved automatically.

3. The library *libmba3D-3.1.a* contains routine *P02P1* which maps a discontinuous piece-wise constant function defined on elements to a continuous piece-wise linear function defined at mesh points (see `src/aniMBA/ZZ.f` for more detail).

   The same library contains a few routines *listX2Y* which create connectivity lists $X \to Y$ for mesh objects $X$ and $Y$ such as elements, faces, boundary faces, edges, and points (see `src/aniMBA/utils.f` for more detail).

## 3.6 How to use library libmba3D-3.1

Here we describe one of the main subroutines, *mbaNodal*, from the library `libmba3D-3.1.a`. Both subroutines have the same number of parameters and differ in only one parameter (`MetricFunction` vs `Metric`).

```
      Call mbaAnalytic(
     &      nP, MaxP, nF, MaxF, nE, MaxE,
     &      XYP, IPF, IPE, lbF, lbE,
     &      nEStar,
     &      nPv, nFv, nEv, IPV, IFV, IEV,
     &      flagAuto, status,
     &      MaxSkipE, MaxQItr,
     &      MetricFunction, Quality, rQuality,
     &      MaxWr, MaxWi, rW, iW,
     &      iPrint, iERR)

      Call mbaNodal(
     &      nP, MaxP, nF, MaxF, nE, MaxE,
     &      XYP, IPF, IPE, lbF, lbE,
     &      nEStar,
     &      nPv, nFv, nEv, IPV, IFV, IEV,
     &      flagAuto, status,
     &      MaxSkipE, MaxQItr,
     &      Metric, Quality, rQuality,
     &      MaxWr, MaxWi, rW, iW,
     &      iPrint, iERR)
```

Some of the parameters were described in Section 3 (see file `src/aniMBA/mba_nodal.f` for more detail). The details on the other input parameters are below:

```
I        MaxP - [integer] maximal number of points
N        MaxF - [integer] maximal number of boundary and interface faces
P        MaxE - [integer] maximal number of tetrahedra
U
T        nEstar - [integer] the desired number of tetrahedra

         nFv - [integer] the number of fixed faces
P        nEv - [integer] the number of fixed tetrahedra
A
R        IFV(nFv) - [integer] list of fixed faces
A        IEV(nEv) - [integer] list of fixed tetrahedra
M
E        flagAuto - [logical] flag controling mesh generation:
T                             TRUE  - recover missing mesh elements
E                             FALSE - check that input data are complete
R
```

```
s       MaxSkipE - [integer] maximal number of skipped tetrahedra
        MaxQItr  - [integer] maximal number of mesh modifications

        Metric(6, nP) - metric defined at mesh nodes. The metric is a
                        3x3 symmetric matrix M_ij. Each column of this
                        array stores the upper diagonal entries in the
                        following order: M_11, M_22, M_33, M_12, M_23, M_13.

        Metricfunction - [integer] function created by the user:

                         Integer Function MetricFunction(x,y,z, Metric)

                         It provides a 3x3 symmetric metric at the given
                         point (x,y,z). Only the upper triangular part of
                         array Metric must be defined.

        Quality  - [real*8] target quality for the final mesh

        Lp - [real*8] the norm in which the final mesh be optimal
                      Lp > 0  means the L_p norm
                      Lp = 0  means the maximum norm
                      Lp < 0  means the H_1 norm (not implemented)

        MaxWr - [integer] maximal memory allocation for rW
        MaxWi - [integer] maximal memory allocation for iW

        iPrint - [integer] level of output information (0 - nothing)
```

Here we collect parameters which are both input and output:

```
I       nP - [integer] the number of points
N       nF - [integer] the number of boundary and interface faces
P       nE - [integer] the number of tetrahedra
U
T       XYP(3, MaxP) - [integer] list of points
/       IPE(4, MaxE) - [integer] list of tetrahedra
O       lbE(MaxE)    - [integer] material indentificator
U
T       IPF(3, MaxF)    - [integer] list of boundary and interface faces
P       ParCrv(2, MaxF) - [real*8]  parametrizations of curved edges
U       iFnc(MaxF)      - [integer] list of corresponding functions
T
        nPv      - [integer] the number of fixed points
        IPV(nPv) - [integer] list of fixed points
P
A       rQuality - [real*8] quality of the final mesh
```

```
R
A       status   - [integer] sum of positive numbers corresponding
M                            to desired mesh properties (see status.fd)
E
T       rW(MaxWr) - [real*8]  working array
E       iW(MaxWi) - [integer] another working array
R
s       iERR   - error code:
                 0 - the correct program termination
```

## 3.7 Useful routines

The library *libmba3D-3.1.a* has a few subroutines that can be useful in other projects. Most of the input parameters in these routines are explained above.

- Mesh postprocessing tool which removes elements with bad shape. The mesh is modified so that to contain approximately the same number of elements which are all shape regular in a user supplied metric. In other words, the tool fixes bad shape elements.

```
      Subroutine mbaFixShape(
     &      nP, MaxP, nF, MaxF, nE, MaxE,
     &      XYP, IPF, IPE, lbF, lbE,
     &      nPv, nFv, nEv, IPV, IFV, IEV,
     &      flagAuto, status,
     &      MaxSkipE, MaxQItr,
     &      MetricFunction, Quality, rQuality,
     &      MaxWr, MaxWi, rW, iW,
     &      iPrint, iERR)
```

  The meanings of parameters are identical to that for subroutine *mbaAnalytic(...)* except for another definition of mesh quality `Quality`. The mesh quality measures only the shape-reguarity of mesh element. An example of using the tool is `src/Tutorials/PackageMBA/main_fixshape.f`.

- Uniform and local mesh refinement (partitioning of a tetrahedron into 8 subtetrahedra). The mesh shape quality is not degradating during this refinements. Local refinement is robust if refinement regions are not fractal. Before refinement, initialization step has to be performed that fills partially arrays *MapMtr(3,3,MaxE)* and *Ref2MapMtr(MaxE)*.

```
      Subroutine initializeRefinement(
     &      nP, nE, XYP, IPE,
     &      MapMtr, Ref2MapMtr)
```

Then, uniform refinement subroutine can be called (the size of working integer array `iW` is at least `14 nE + nP` where `nP,nE` are the input values)

```
 Subroutine uniformRefinement(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      XYP, IPF, IPE, lbF, lbE,
&      MapMtr, Ref2MapMtr,
&      iW, MaxWi)
```

A local refinement subroutine refines elements marked `.TRUE.` in the logical array `SplitFlag(nE)`. The size of working integer array `iW` is at least `14 nE + nP + 3 nR + 4 min(MaxF,4 nF)` where `nP,nE,nF,MaxF` are the input values and `nR` is the estimated number of edges in the input mesh:

```
 Subroutine localRefinement(
&      nP, MaxP, nF, MaxF, nE, MaxE,
&      XYP, IPF, IPE, lbF, lbE,
&      MapMtr, Ref2MapMtr, SplitFlag,
&      iW, MaxWi)
```

An example of using the tools is `src/Tutorials/PackageMBA/mainRefineMesh.f`.

- `P02P1` maps a discontinuous piece-wise constant function defined on tetrahedra to a continuous piece-wise linear function defined at points. We use the ZZ method for the interpolation. We assume that each boundary node can be connected with an interior node by at most two mesh edges. The size of working integer array `iW` is `4 nE + 3 nP`. The size of working double precision array `rW` is `nP`.

```
 Subroutine P02P1(
&           nP, nF, nE, XYP, IPF, IPE,
&           fP0, fP1,
&           MaxWr, MaxWi, rW, iW)
```

- `listE2R` creates a connectivity list `IRE` for mesh edges. The routine counts mesh edges. For an element `E`, `IRE([1:6], E)` give indexes of six edges in the following order: 12, 13, 14, 23, 24, and 34. For example, the second edge is `[IPE(1,E), IPE(3,E)]`. The working integer arrays are `nEP(nP)` and `IEP(4 nE)` (see `src/aniMBA/utils.f` for more detail):

```
 Subroutine listE2R(
&           nP, nR, nE, IPE, IRE, nEP, IEP)
```

- `listR2R` creates connectivity lists `nRR` and `IRR` for mesh edges. The routine counts the number of mesh edges, `nR`. Then, `nRR(i) - nRR(i-1)` (`nRR(1)` when `i=1`) gives the total number of edges in tetrahedra sharing the edge `i`. The corresponding edge numbers are saved in array `IRR` in positions `nRR(i-1) + 1` to `nRR(i)`. The size of working integer array `iW` is `12 nE + nR` (see `src/aniMBA/utils.f` for more detail):

```
      Subroutine listR2R(
     &           nP, nR, nE, MaxL, IPE, nRR, IRR, iW, iERR)
```

- File `src/aniMBA/utils.f` contains more routines for creating other connectivity lists such as edges to points, points to points, elements to boundary edges, etc. The routine *listConv* colvolutes two given connectivity lists. The routines *backReferences* and *reverseMap* create reverse connetivity lists for a given structured and unstructured connectivity list, respectively. For instance, *backReferences* takes the structured connectivity list `IPE` from elements to points and creates the unstructured connectivity lists `nEP` and `IEP` from points to elements.

## 3.8 FAQ

- Q. The mesh generator does not refine the input mesh.
  A. There are two cases when the code may do nothing. First, the number of mesh elements whose quality is limited by model geometry (e.g. thin layers) is bigger then the control parameter `MaxSkipE`. The remedy is to increase this parameter. Second, a severe anisotropic input metric does not allow to insert new mesh points in a very coarse mesh. The simple remedy is to refine mesh using an isotropic metric and then switch to the anisotropic metric.

- Q. The mesh generator uses the same input data but produces different grids on different computers.
  A. The output of the mesh generator may depend on a computer arithmetics. The order of local mesh modifications depends on round-off errors and may be computer-dependent.

- Q. The final mesh quality is very small.
  A. The mesh quality equals to quality of the worst tetrahedron in the mesh. In some cases, the shape of near-boundary tetrahedra is driven mainly by the geometry. A possible remedy is either to increase the number `nEStar` of desired tetrahedra or to fix a possible contradiction between the boundary and the metric. Another reason for the low mesh quality is strong jumps in the metric. If the metric is isotropic, the optimal tetrahedra are equilateral ones. This size is changed strongly across lines of metric discontinuity.

- Q. The number of tetrahedra in the final mesh is never equal to `nEStar`.
  A. `nEStar` is achieved exactly if and only if `Quality = 1` and the computational domain may be covered by equilateral (in the user given metric) tetrahedra. Apparently, it is possible only in very special cases.

- Q. Why *mbaNodal* and *mbaAnalytic* have so many input parameters?
  A. The package has routines *mbaAnalyticShort* and *mbaNodalShort* with functionality close to that of main subroutines *mbaAnalytic* and *mbaNodal*, respectively, but with smaller number of input parameters. For example, lists of fix points and boundary tetrahedra are omitted.

- Q. Is it possible to use *libmba3D-3.1.a* in an adaptive loop?
  A. Yes. Use `make libs` to generate the library *libmba3D-3.1.a* which may be linked with other codes. Depending on the user goals, he or she may call either *mbaAnalytic* or *mbaNodal*. The package contains a few examples of solving partial differential equations on adaptive grids (see `src/Tutorials/MultiPackage` for more detail).

- Q. I do not understand why *libmba3D-3.1.a* fails to generate a mesh.
  A. The developers are interested in any feedback from the users. To report a problem, please email to either `lipnikov@gmail.com` or `yuri.vassilevski@gmail.com`. To help us to fix the problem, please attach file `main_analytic.f` or `main_nodal.f` and files containing the input mesh.

# References

1. Yu.Vassilevski and K.Lipnikov, An adaptive algorithm for quasioptimal mesh generation, *Computational Mathematics and Mathematical Physics* (1999) **39**, No.9, 1468–1486.

2. A.Agouzal, K.Lipnikov, Yu.Vassilevski, Adaptive Generation of Quasi-optimal Tetrahedral Meshes, *East-West Journal* (1999) **7**, No.4, 223–244.

3. K.Lipnikov, Y.Vassilevski, Parallel adaptive solution of 3D boundary value problems by Hessian recovery, *Comput. Methods Appl. Mech. Engrg.* (2003) **192**, 1495–1513.

4. K.Lipnikov, Yu. Vassilevski, Optimal triangulations: existence, approximation and double differentiation of $P_1$ finite element functions, *Computational Mathematics and Mathematical Physics* (2003) **43**, No.6, 827–835.

5. K.Lipnikov, Yu.Vassilevski, On a parallel algorithm for controlled Hessian-based mesh adaptation. Proceedings of 3rd Conf. Appl. Geometry, Mesh Generation and High Performance Computing, Moscow, June 28 - July 1, Comp. Center RAS, V.1, 2004, 154–166.

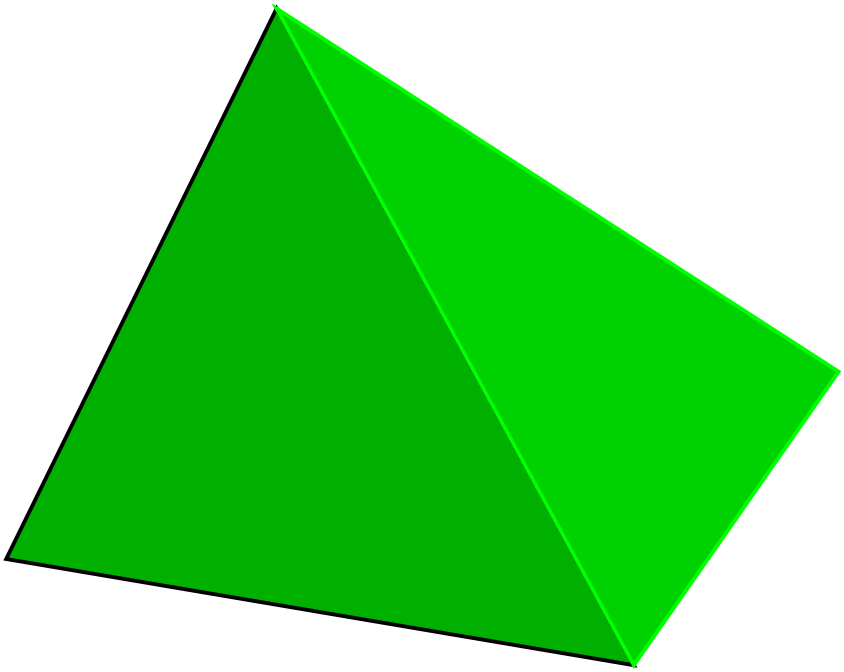
6. K.Lipnikov, Yu.Vassilevski, On control of adaptation in parallel mesh generation. *Engrg. Computers* (2004) **20**, 193–201.

7. K.Lipnikov, Yu.Vassilevski, Error bounds for controllable adaptive algorithms based on a Hessian recovery. *Computational Mathematics and Mathematical Physics* (2005) **45**, 1374–1384.

8. K.Lipnikov, Yu.Vassilevski, Analysis of Hessian recovery methods for generating adaptive meshes. *Proceedings of 15th International Meshing Roundtable*, P.Pebay (Editor), Springer, Berlin, Heidelberg, New York, 2006, pp.163–171.

# Chapter 2

# DISCRETIZATION PACKAGEs



# Ani3D-FEM version 3.1 "*Sunflower*"

# Flexible Generator of Finite Elements Systems on Tetrahedral Meshes

# User's Guide for libfem3D-3.1.a

## 4.1 Introduction

The Fortran package Ani3D-FEM is developed by Konstantin Lipnikov and Yuri Vassilevski. It generates finite element matrices on tetrahedral meshes. The package allows to build elemental matrices for variety of finite elements, modify these matrices, assemble them, and impose boundary conditions.

The package Ani3D-FEM differs from other similar packages by providing a very flexible interface for incorporating problem coefficients in elemental matrices. In addition, the elemental matrices are understood in a very broad sense. They may involve different types of finite elements. For instance, the elemental matrix for the Stokes problem is a saddle point matrix involving both the pressure and velocity unknowns.

The library *libfem3D-3.1.a* can be incorporated into other packages.

This document describes structure of the package, input data, and the user-supplied routines. It also presents a few examples illustrating details of the package.

## 4.2 Copyright Notice



## 4.3 Description of Ani3D-FEM

### 4.3.1 Elemental finite element matrix

The core of the package is routine *fem3Dtet* which computes elemental matrix corresponding to the bilinear form

$$\int_T \mathbf{D}\, OP_A(u^h) \cdot OP_B(v^h)\, \mathrm{d}x \tag{4.1}$$

where $\mathbf{D}$ is a tensor, $OP_A$ and $OP_B$ are linear first-order or zero-order differential operators, and $u^h$ and $v^h$ are finite element functions. Table 4.1 shows the list of implemented finite elements (see file `fem3Dtet.f` for more detail). Table 4.2 gives the list of available operators. The package understands a few types of tensor $\mathbf{D}$. Table 4.3 gives the list of supported tensors.

The following compatability rule must held. If we consider the tensor and operators as matrices, the product of these matrices in (4.1) must exist. For instance, the gradient of a vector linear finite element, GRAD(`FEM_P1vector`), is a $9 \times 12$ matrix. Thus, the tensor $\mathbf{D}$ can be either a constant or a matrix with 9 columns. If the bilinear form is symmetric, the tensor is a square matrix and the resulting stiffness matrix is $12 \times 12$.

| Finite Element | Description |
|---|---|
| FEM_P0 | piecewise constant, $P_0$ |
| FEM_P1 | continuous piecewise linear, $P_1$ |
| FEM_P2 | continuous piecewise quadratic, $P_2$ |
| FEM_P3 | continuous cubic quadratic, $P_3$ |
| FEM_P1vector | vector continuous piecewise linear, $P_1 \times P_1 \times P_1$. The unknowns are ordered first by vertices and then by the space directions (x and y) |
| FEM_P2vector | vector continuous piecewise quadratic, $P_2 \times P_2 \times P_2$. The unknown are ordered first by vertices, then by edges, and then by the space directions (x and y) |
| FEM_ND0 | the lowest order Nedelec (edge) finite element |
| FEM_RT0 | the lowest order Raviart-Thomas (face) finite element |
| FEM_CR1 | the Crouzeix-Raviart finite element |

Table 4.1: Available finite elements.

| Operator Name | Description |
|---|---|
| IDEN | identity operator, IDEN$(v^h) = v^h$ |
| GRAD | gradient operator, GRAD$(v^h) = \nabla v^h$ |
| DIV | divergence operator, DIV$(v^h) = \mathrm{div} v^h$ |
| CURL | rotor operator, CURL$(v^h) = \nabla \times v^h$ |
| DUDX | partial derivative in x-direction DUDX$(v^h) = \partial v^h / \partial x$ |

Table 4.2: Available operators

| Tensor Type | Description |
|---|---|
| TENSOR_NULL | identity tensor |
| TENSOR_SCALAR | scalar tensor |
| TENSOR_SYMMETRIC | symmetric tensor |
| TENSOR_GENERAL | general tensor |

Table 4.3: Available tensors

The package uses several quadrature rules:

| | |
|---|---|
| order = 1 | quadrature formula with one center point |
| order = 2 | quadrature formula with 4 points inside tetrahedron |
| order = 3 | quadrature formula with 8 points inside tetrahedron |
| order = 5 | quadrature formula with 15 points inside tetrahedron |
| order = 6 | quadrature formula with 24 points inside tetrahedron |

The quadrature rules have positive weights.

A solution of non-linear problems is usually based on a Newton-type iterative method. In this case, the tensor **D** may depend on a discrete function (e.g. solution approximation from the previous iterative step). If so, evaluation of **D** may be a complex procedure and may require additional data. We provide the flexible machinery for incorporating additional data into the user written function `Dcoef` that calculates **D**. This function has the following format:

```
      Integer Function Dcoef(x, y, z, label, DDATA, IDATA, iSYS, Coef)

C     The function returns type of the tensor Coef (see Table 3 above).
C
C     (x, y, z) - [input] Real*8 Cartesian coordinates of a 3D
C                 point where tensor Coef should be evaluated
C
C     label     - [input] Integer label of a either element or face
C
C     DDATA     - [input] Real*8 user given data array
C
C     IDATA     - [input] Integer user given data array
C
C     iSYS(21)  - [input/output] integer buffer for information exchange:
C                  iSYS(1) = iD [output] number of rows in Coef
C                  iSYS(2) = jD [output] number of columns in Coef
C                  iSYS(3)      [input]  tetrahedron number
C                  iSYS(4:7)    [input]  numbers of vertices
C                  iSYS(8:13)   [input]  numbers of edges
C                  iSYS(14:17)  [input]  numbers of faces
C                  iSYS(18)     [input]  the number of points
C                  iSYS(19)     [input]  the number of edges (if available)
C                  iSYS(20)     [input]  the number of faces
C                  iSYS(21)     [input]  the number of tetrahedra
C
C     Coef(9,jD) - [output] Real*8 matrix with the leading dimension 9!
```

To compute entries of the tensor `Coef`, the user may use the tetrahedron number `iSYS(3)` and arrays `DDATA, IDATA`. Here are a few examples.

- isotropic diffusion coefficient. The user has to set iD = jD = 1, Dcoef = TENSOR_SCALAR and to return the diffusion value Coef(1,1) at the point ($x$, $y$, $z$).

- anisotropic diffusion coefficient. The user has to set iD = jD = 3, Dcoef = TENSOR_SYMMETRIC, and to return diffusion tensor (3x3 matrix with entries Coef(i,j), i,j=1,2,3) at the point ($x$, $y$, $z$).
- convection coefficient. The user has to set iD = 1, jD = 3, Dcoef = TENSOR_GENERAL, and to return the velocity vector values Coef(1,1), Coef(1,2), Coef(1,3) at the point ($x$, $y$, $z$).

Now we are ready to call routine *fem3Dtet* which computes an elemental matrix A:

```
      Call FEM3Dtet(
     &     XY1, XY2, XY3, XY4,
     &     OpA, FemA, OpB, FemB,
     &     label, Dcoef, DDATA, IDATA, iSYS, order,
     &     LDA, A, nRow, nCol)

C     XYi(3)     - [input] Real*8 Cartesian coordinates of i-th vertex
C     OpA, OpB   - [input] operators in (4.1), integers
C     FemA, FemB - [input] type of finite elements from (4.1), integers
C
C     Dcoef      - [input] external integer function described above
C     order      - [input] order of the numeric quadrature, integer
C
C     LDA        - [input] leading dimension of matrix A(LDA, LDA)
C     A(LDA,LDA) - [output] Real*8 finite element matrix A
C     nRow       - [output] the number of rows of A
C     nCol       - [output] the number of columns of A
```

The following rules are applied for numbering unknowns within the elemental matrix:

- First, basis functions associated with vertices (if any) are numerated in the same order as the vertices $r_i$, $i = 1, 2, 3, 4$ (input parameters XY1, XY2, XY3, XY4).
- Second, basis functions associated with edges (if any) are numerated in the order as edges $r_{12}$, $r_{13}$, $r_{14}$, $r_{23}$, $r_{24}$ and $r_{34}$, where $r_{ij}$ is the edge with end-points $r_i$ and $r_j$.
- Third, basis function associated with faces are numerated in the same order as faces $r_{123}$, $r_{234}$, $r_{341}$ and $r_{412}$.
- Fourth, basis functions associated with element (if any) are numerated.
- The vector basis functions with 3 degrees of freedom per a mesh object (vertex, edge, face) are numerated first by the corresponding mesh objects and then by the space coordinates, first $x$, then $y$, and finally $z$.

In order to compute a linear form representing an elemental right hand side, we can use the following trick:

$$f(v^h) = \int_T \mathbf{D}_{rhs}\, \mathrm{IDEN}(\texttt{FEM\_P0})\ \mathrm{IDEN}(v^h)\, \mathrm{d}x \tag{4.2}$$

where $\mathbf{D}_{rhs}$ represents the right hand side function $f$:

```
      Call FEM3Dtet(
     &     XY1, XY2, XY3, XY4,
     &     IDEN, FEM_P0, IDEN, FemB,
     &     label, Drhs, DDATA, IDATA, iSYS, order,
     &     LDA, F, nRow, nCol)
```

### 4.3.2 Boundary integrals

To implement boundary conditions, we need to compute an elemental matrix corresponding to the bilinear form

$$\int_f \mathbf{D}\, OP_A(u^h) \cdot OP_B(v^h)\, \mathrm{d}x \tag{4.3}$$

or

$$\int_f \Big(\mathbf{D}\, OP_A(u^h) \cdot \mathbf{n}_f\Big) \cdot OP_B(v^h)\, \mathrm{d}x \tag{4.4}$$

on manifold $f$ (more precesily, on triangle $f$) in 3D. Here $\mathbf{D}$ is a tensor, $OP_A$ and $OP_B$ are linear first-order or zero-order differential operators, $u^h$ and $v^h$ are finite element functions, and $\mathbf{n}_f$ is (exterior) normal for the manifold. The calculation of the elemental matrix is done with routine *FEM3Dface* that has all parameters of routine *FEM3Dtet* plus two additional parameter related to face $f$:

```
      Call FEM3Dface(
     &     XY1, XY2, XY3, XY4, idface, dot_with_normal,
     &     operatorA, FEMtypeA, operatorB, FEMtypeB,
     &     label, D, DDATA, IDATA, iSYS, order,
     &     LDA, A, nRow, nCol)

C     XYi(3)     - [input] Real*8 Cartesian coordinates of i-th vertex
C     idface     - [input] local face id in the tet (1, 2, 3, or 4 for
C                          faces 123, 234, 341, and 412
C     dot_with_normal - [input] integer switch between formula (4.3)
C                          and (4.4). If zero, the 1st formula is used.
C
C     OpA, OpB   - [input] operators in (4.3), integers
C     FemA, FemB - [input] type of finite elements from (4.3), integers
C
C     Dcoef      - [input] external integer function described above
C     order      - [input] order of the numeric quadrature, integer
C
C     LDA        - [input] leading dimension of matrix A(LDA, LDA)
C     A(LDA,LDA) - [output] Real*8 finite element matrix A
C     nRow       - [output] the number of rows of A
C     nCol       - [output] the number of columns of A
```

Here is the deprecated routine for calculating boundary conditions with limited capabilities. It projects face $f$ on the $XY$-plane and uses two-dimensional algorithms.

```
      Call fem3Dtri(  ! deprecated
     &     XY1, XY2, XY3,
     &     operatorA, FEMtypeA, operatorB, FEMtypeB,
     &     label, Dcoef, DDATA, IDATA, iSYS, order,
     &     LDA, A, nRow, nCol)

C     XYi(3)     - [input] Real*8 Cartesian coordinates of i-th vertex
C     OpA, OpB   - [input] operators in (4.3), integers
C     FemA, FemB - [input] type of finite elements from (4.3), integers
C
C     Dcoef      - [input] external integer function described above
C     order      - [input] order of the numeric quadrature, integer
C
C     LDA        - [input] leading dimension of matrix A(LDA, LDA)
C     A(LDA,LDA) - [output] Real*8 finite element matrix A
C     nRow       - [output] the number of rows of A
C     nCol       - [output] the number of columns of A
```

The integer function `Dcoef` calculating the tensor $\mathbf{D}$ has been described above. Note that the tensor returned by it must be consistent with the two-dimensional operators $OP_A$ and $OP_B$. For instance, the leading dimension of the full tensor should be 4. The following rules are applied for numbering unknowns within the elemental matrix:

- First, basis functions associated with vertices (if any) are numerated in the same order as the vertices $r_i$, $i = 1, 2, 3$ (input parameters XY1, XY2, XY3).
- Second, basis functions associated with edges (if any) are numerated in the order as edges $r_{12}$, $r_{23}$, and $r_{13}$, where $r_{ij}$ is the edge with end-points $r_i$ and $r_j$.
- Third, basis function associated with the triangle $f$ are numerated.
- The vector basis functions with 3 degrees of freedom per a mesh object (vertex, edge, face) are numerated first by all mesh objects (as described above) and then by the space coordinates, first $x$, then $y$, and finally $z$.

### 4.3.3 Extended elemental finite element matrix

Now we describe an alternative way to create elemental matrices. Each elemental matrix may be a combination of a few *fem3Dtet* calls reflecting the fact that the bilinear forms (4.1), (4.3), (4.4) may consist of a few simple bilinear forms. One of the examples is the Stokes problem. Degrees of freedom in the extended elemental matrix are characterized by arrays templateR and templateC:

```
     Subroutine FEM3Dext(
    &           XY1, XY2, XY3, XY4,
    &           lbE, lbF, lbR, lbP, DDATA, IDATA, iSYS,
    &           LDA, A, F, nRow, nCol,
    &           templateR, templateC)

C     XYi(3)  - [input] Real*8 Cartesian coordinates of the i-th point
C
C     lbE     - [input] Integer label of the tetrahedron (material label)
C     lbF(4)  - [input] Integer labels of its faces (boundary labels)
C                       lbF(i) = 0 for internal faces
C     lbR(6)  - [input] Integer labels of its edges (copied from global lbR)
C     lbP(3)  - [input] Integer labels of its vertices (copied from global lbP)
C
C     DDATA(*)- [input] Real*8 user given data
C     IDATA(*)- [input] Integer user given data
C     iSYS(21)- [input] Integer array providing tetrahedral information
C                       related to the mesh. See description above.
C
C     LDA       - [input] leading dimension of matrix A
C     A(LDA, *) - [output] Real*8 elemental matrix, degrees of freedom
C                 are ordered according to templateR and templateC
C     F(nRow)   - [input] Real*8 vector of the right-hand side
C
C     nRow      - [output] the number of rows in A
C     nCol      - [output] the number of columns in A
C
C     templateR(nRow) - [output] Integer array of degrees of freedom for
C                       rows. We recommend to group them as follows: four
                        for points, six for edges, etc.
C     templateC(nCol) - [output] Integer array of degrees of freedom for
C                       columns.
```

In general, different order of unknowns is allowed. However, in the assembled matrix, they will be grouped according to their geometric location. For instance, the first four unknowns associated with points will go to the first group of point-based unknowns. Next four point-based unknowns will go to the second group. After counting all point-based unknowns, we group the face-based unknowns, then the edge-based unknows, and finally the element-based unknowns.

Admissible values for arrays `templateR` and `templateC` are defined in file *fem3Dtet.fd*. Including this header file, the user may indicate a vertex degree of freedom as follows:

```
templateR(i) = Vdof
```

The degrees of freedom on faces are indicated either by `Fdof` or `FdofOrient`. The former corresponds to a scalar unknown (e.g. a Lagrange multiplier in a hybrid mixed finite element) that has no orientation. The latter correponds to a vector unknown (e.g. the

Raviart-Thomas finite element basis function) that has orientation. Similarly, degrees of freedom on edges are indicated either by `Rdof` or `RdofOrient`. Finally, degree of freedom associated with a mesh element is indicated by `Edof`.

Here are a few examples where this routine may be useful.

- For a diffusion-reaction equation, we sum elemental matrices corresponding to the diffusion and reaction terms.
- For a diffusion equation written in a mixed form, we use hybridization algorithm inside `FEM3Dext` and return the elemental matrix for Lagrange multipliers.
- We may also incorporate boundary conditions in the elemental matrix `A`.

### 4.3.4 Assembling routines

The package provides a few routines for assembling elemental matrices and right hand sides. The assemble routine returns a sparse matrix in the CSR (compressed sparse row) format with diagonal entries at the beginning of each row. Other formats will be supported in the nearest future or by request (see converters in file `algebra.f`). Here is the header of the assembling routine that must be used for the extended elemental matrix described in Section 4.3.3. We describe only the new parameters.

```
      Subroutine BilinearFormTemplate(
     &           nP, nF, nE, XYP, lbP, IPF, lbF, IPE, lbE,
     &           FEM3Dext, DDATA, IDATA, assembleStatus,
     &           MaxIA, MaxA, IA, JA, A, F, nRow, nCol,
     &           MaxWi, iW, iPrint)

C     nP  - [input] the number of points (P)
C     nF  - [input] the number of edges (F)
C     nE  - [input] the number of elements (E)
C
C     XYP(3, nP) - [input] Real*8 Cartesian coordinates of mesh points
C     lbP(nP)    - [input] Integer labels of mesh points
C
C     IPF(3, nF) - [input] connectivity list of boundary faces (see
C                  documentation for package Ani3D-MBA)
C     IPE(4, nE) - [input/output] connectivity list of elements.
C                  On output, each column is ordered by increasing.
C
C     lbF(nF)    - [input] boundary labels
C     lbE(nE)    - [input] element (material) labels
C
C     assembleStatus - [input] a priori information about matrix A.
C              The logical sum of constants defined in assemble.fd.
C                   MATRIX_SYMMETRIC - symmetric matrix
C                   MATRIX_GENERAL   - general matrix
```

```
C
C                   FORMAT_AMG - format used in the AMG solver (CSR with
C                                rows starting with the diagonal entry)
C                   FORMAT_ROW - diagonal of A is saved only in DA
C
C     MaxIA  - [input] the maximal number of equations plus one
C     MaxA   - [input] the maximal number of nonzero entries in A
C     IA,JA,DA,A - [output] sparsity structure of matrix A:
C
C          IA(nRow+1)- number IA(k + 1) equals to the number of
C                      nonzero entries in first k rows  plus 1
C          JA(M)     - column indexes of non-zero entries ordered
C                      by rows, M = IA(nRow + 1) - 1
C          A(M)      - non-zero entries ordered as in JA
C
C     nRow - [output] the number of rows in A
C     nCol - [output] the number of columns in A
C
C     MaxWi - [input] size of the working integer array
C     iW(MaxWi) - [input] integer working array
C
C     iPrint - [input] verbosity level (0 means no output)
```

The matrix `A` is in one of the sparse row formats. The unknowns are ordered in groups as explained in Section 4.3.3.

## 4.4 Examples

### 4.4.1 Diffusion problem

The program `src/Tutorials/PackageFEM/mainSimple.f` demonstrates the approximate solution of the boundary value problem with continuous piecewise linear finite elements $P_1$:

$$\begin{aligned} -\mathrm{div}(D\,\mathrm{grad}\,u) &= 1 \quad \text{in} \quad \Omega, \\ u &= 0 \quad \text{on} \quad \partial\Omega_D, \\ \frac{\partial u}{\partial n} &= 0 \quad \text{on} \quad \partial\Omega_N, \end{aligned}$$

where $\Omega = T(x,y) \times (0,1)$ is a prism with base $T$. This base is the triangle with vertices $(0,0)$, $(0,1)$ and $(1,0)$. The Neumann boundary condition is imposed on one side of the prism only, $\partial\Omega_N = \{(x,y,z) \in \partial\Omega : y = 0\}$. The Dirichlet boundary condition is imposed on the rest of the boundary $\partial\Omega_D = \partial\Omega \setminus \partial\Omega_N$. The diffusion coefficient $D$ is a scalar function:

$$D(x,y,z) = 1 + x^2.$$

First, the program refines an initial coarse mesh consisting of three tetrahedra. This is accomplished with routines *initializeRefinement* and *uniformRefinement* from the library

*libmba3D-3.1.a*. Second, the program generates the finite element system using subroutines *BilinearFormVolume, LinearFormVolume* and *BoundaryConditions* from the library *libfem3D-3.1.a*.

### 4.4.2 Stokes problem

The program `src/Tutorials/PackageFEM/mainTemplate.f` demonstrates the approximate solution of the Stokes problem with $P_2 \times P_1$ pair of finite elements:

$$\begin{aligned} -\operatorname{div}\operatorname{grad}\mathbf{u} + \nabla p &= 0 \quad \text{in} \quad \Omega, \\ -\operatorname{div}\mathbf{u} &= 0 \quad \text{in} \quad \Omega, \\ \mathbf{u} &= \mathbf{u}_0 \ \text{on} \quad \partial\Omega_1, \\ \mathbf{u} &= 0 \quad \text{on} \quad \partial\Omega_2, \\ \frac{\partial \mathbf{u}}{\partial n} - p &= 0 \quad \text{on} \quad \partial\Omega_3, \end{aligned}$$

where $\Omega = T(x,y) \times (0,1)$ is a prism with base $T$. This base is the triangle with vertices $(0,0)$, $(0,1)$ and $(1,0)$. The boundaries $\partial\Omega_i$ are defined as follows:

$$\begin{aligned} \partial\Omega_1 &= \{(x,y,z) \in \partial\Omega : z = 0\}, \\ \partial\Omega_3 &= \{(x,y,z) \in \partial\Omega : z = 1\}, \\ \partial\Omega_2 &= \partial\Omega \setminus (\partial\Omega_1 \cup \partial\Omega_3)\,. \end{aligned}$$

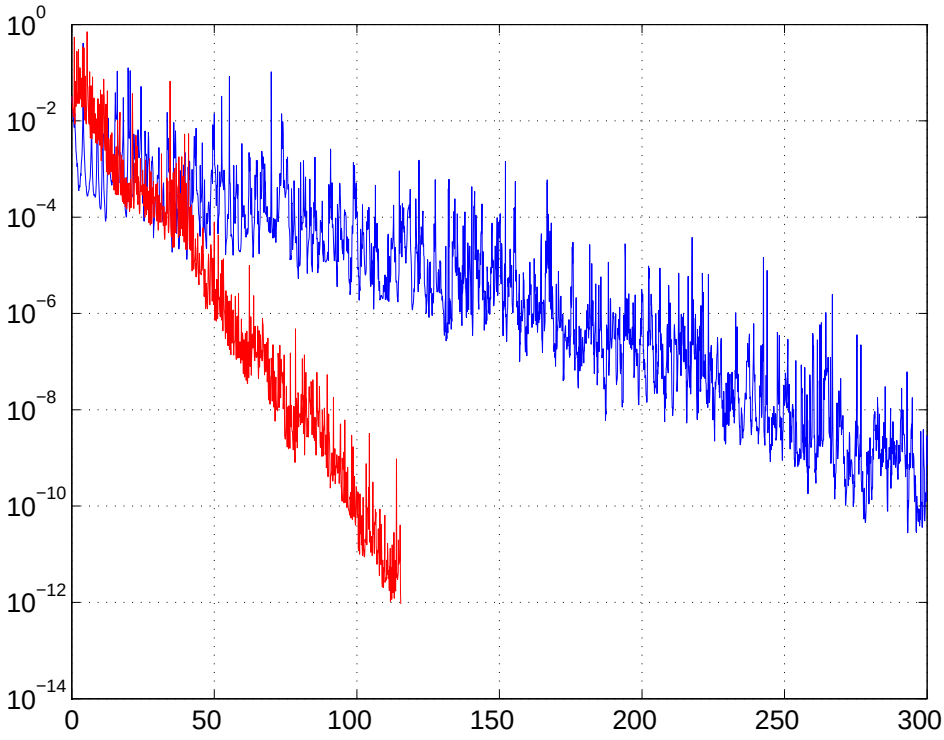
The non-homogeneous boundary condition is

$$\mathbf{u}_0 = (0,\, 0,\, 4y(1-y))^T.$$

First, the program refines an initial coarse mesh consisting of three tetrahedra and creates a quasi-uniform mesh with `460` elements. This is accomplished with subroutine *mbaMetric* from the library *libmba3D-3.1.a*. Second, the program generates the finite element system using subroutine *BilinearFormTemplate* from the library *libfem3D-3.1.a*. The elemental matcies are generated in subroutine *FEM3Dext* which is in file `src/Tutorials/PackageFEM/mainTemplate.f`.

# Chapter 3

# SOLUTION PACKAGEs



# Ani3D-ILU Version 3.1 "*Bellflower*"

# Flexible Iterative Solver Using Incomplete LU Factorization

# User's Guide for libilu-3.1.a

## 5.1 Basic features of the library

The FORTRAN package Ani3D-ILU is an independent part of the package Ani3D . Ani3D-ILU was developed by Yuri Vassilevski, Sergey Goreinov and Vadim Chugunov. It is designated for the iterative solution of sparse linear systems. Ani3D-ILU may be easily incorporated into any other software, for instance, package Ani3D .

The basic features of library *libilu-3.1.a* are listed below.

**Iterative method** : BiConjugate Gradient Stabilized, BiCGStab, and Conjugate Gradient, PCG

**Preconditioners** : ILU0 and ILU2, the second order accurate ILU

**Matrix storage format** : Compressed Sparse Row-wise, CSR

**Data format** : double precision or integer arrays. Enumeration starts from 1.

**Typical memory requests** : for systems with $N$ equations and $NZ$ non-zero matrix elements, BiCGStab (resp., PCG) needs 8 (resp., 4) work vectors of dimension $N$, right-hand side and solution vectors. ILU0 requires the same storage as the CSR matrix representation. ILU2 requires upto 2-5-fold memory for the CSR matrix representation.

## 5.2 Iterative solution

The default iterative solver is BiConjugate Gradient Stabilized method (BiCGStab). This is the Krylov subspace method applicable to non-singular non-symmetric matrices. Therefore, it requires two procedures: matrix-vector multiplication and precondition-vector evaluation. If the user is not confident that the matrix and the preconditioner are symmetric positive definite, he or she is advised to choose the default method. The call of the method is

```
 call slpbcgs(
&  prevec, IPREVEC, iW,rW,
&  matvec, IMATVEC, ia,ja,a,
&  WORK, MW, NW,
&  N, RHS, SOL,
&  ITER, RESID,
&  INFO, NUNIT )
```

- `prevec` is the name of a precondition-vector multiply routine and `IPREVEC` is an integer array of user's data which may be passed to `prevec` and used there. In the presented examples of precondtioners `IPREVEC` contains single entry equal to the system order. The format of `prevec` is:

```
      Subroutine prevec(IPREVEC, ICHANGE, X, Y, iW, rW)
c Input
      Integer IPREVEC(*), ICHANGE, iW(*)
      Real*8  X(*), rW(*)
c Output
      Real*8  Y(*)
```

This routine solves the system $(LU)\,Y = X$ with $L$ low triangular and $U$ upper triangular factors stored in arrays `iW,rW`. `ICHANGE` is the flag controlling the change of the preconditioner (useful when convergence stagnation occurs). The user is given two examples of `prevec` corresponding to two preconditioners, `prevec0` (ilu0.f) and `prevec2` (iluoo.f).

- `iW, rW` are Integer and Real*8 arrays which store the preconditioner.
- `matvec` is the name of generalized matrix-vector multiply routine and `IMATVEC` is an integer array of user's data which may be passed to `matvec` and used there. In the presented example `IMATVEC` contains single entry equal to the system order. The format of `matvec` is:

```
      Subroutine matvec(IMATVEC, ALPHA, X, BETA, Y, ia, ja, a)
c Input
      Integer IMATVEC(*), ia(*), ja(*)
      Real*8  X(*), Y(*), a(*), ALPHA, BETA
c Output
      Real*8  Y(*)
```

This routine calculates matrix-vector product $AX$ and adds the vector $\beta Y$:

$$Y := \alpha AX + \beta Y.$$

For example, for $\alpha = 1, \beta = 0$ `matvec` returns $Y = AX$. The example of a `matvec` routine is in file `bcg.f`. It uses the compressed sparse row (CSR) representation of matrix $A$ stored in arrays `ia, ja, a`.

- `ia,ja,a` are two Integer and one Real*8 arrays containing matrix in the CSR format.
- `WORK(MW,NW)` is Real*8 working two-dimensional array which stores at least 8 Krylov vectors.
- `MW*NW` the total length of `WORK` which must be not less than `8N`.
- `N` is order of system and length of vectors.
- `RHS` is the right hand side vector (Real*8).
- `SOL` is the initial guess and the iterated solution (Real*8).
- `ITER` is the maximal number of iterations on input and the actual number of iterations on output.

- `RESID` is the convergence criterion on input and norm of the final residual on output.
- `INFO` is the performance information, 0 - converged, 1 - did not converge, etc.
- `NUNIT` is the channel number for output (0 - no output).

If the user is confident that the matrix and the preconditioner are symmetric and positive definite, he or she can save 4 work vectors and probably 10-30% of the CPU time by calling the Preconditioned Conjugate Gradient method (PCG):

```
 call slpcg(
&  prevec, IPREVEC, iW,rW,
&  matvec, IMATVEC, ia,ja,a,
&  WORK, MW, NW,
&  N, RHS, SOL,
&  ITER, RESID,
&  INFO, NUNIT )
```

The parameters of this routine are the same, except that `MW*NW` must be not less than $4N$.

## 5.3 ILU0 preconditioner

ILU0 preconditioner is the simplest and the most popular ILU preconditioner. It is characterized by very fast and economical factorization. The drawbacks of the method are slow convergence and danger to get zero pivot. Nevertheless, for simple non-stiff problems it works well. The application of the preconditioner has two stages: initialization and evaluation. The evaluation must be performed at each step of the iterative method. It is provided by the routine `prevec0` accompanying the initialization routine `ilu0`. The user should only put the name `prevec0` as the input parameter in the iterative solver:

```
   external prevec0
   ....

   call slpbcgs(
&  prevec0, IPREVEC, iW,rW,
&  matvec, IMATVEC, ia,ja,a,
&  WORK, MW, NW,
&  N, RHS, SOL,
&  ITER, RESID,
&  INFO, NUNIT )
```

The initialization routine has the following parameters

```
call ilu0(n, a, ja, ia, alu, jlu, ju, iw, ierr)
```

where

- `n` is matrix order,
- `ja,ia,a` are two Integer and one Real*8 arrays containing the matrix in the CSR format,
- `alu,jlu,ju` are one Real*8 and two Integer arrays containing the L and U factors together,
- `ierr` is the integer error code (0 - successful factorization, $k$ - zero pivot at step $k$),
- `iw` is the integer working array of length $n$.

Below we present the basic blocks of a program solving a system with matrix `a, ia, ja` and a right hand side vector `f` by the BiCGstab method with the ILU0 preconditioner.

First we define all necessary arrays and variables:

```
C Arrays for matrix  kept in CSR format
      Integer ia(maxn+1), ja(maxnz)
      Real*8   a(maxnz), f(maxn), u(maxn)

C Work arrays keeping ILU factors and 8 BCG vectors
      Integer   MaxWr,MaxWi
      Parameter(MaxWr=maxnz+8*maxn, MaxWi=maxnz+2*maxn+1)
      Real*8  rW(MaxWr)
      Integer iW(MaxWi)

C BiCGStab data
      External  matvec, prevec0
      Integer   ITER, INFO, NUNIT
      Real*8    RESID

C ILU0 data
      Integer   ierr, ipaLU, ipjLU, ipjU, ipiw

C Local variables
      Integer   ipBCG
```

When the matrix is stored in the CSR format, we initialize the preconditioner by computing $L$ and $U$ factors and saving them in `rW, iW`:

```
      ipaLU=1
      ipBCG=ipaLU+nz
      ipjU =1
```

```
      ipjLU=ipjU+n+1
      ipiw =ipjLU+nz !work array of length n

      call ilu0(n,a,ja,ia, rW(ipaLU),iW(ipjLU),iW(ipjU),iW(ipiw),ierr)

      if (ierr.ne.0) then
         write(*,*)'initialization of  ilu0 failed, zero pivot=',ierr
         stop
      end if

c Keep data in rW and iW up to rW(nz) and iW(nz+n+1) !
```

Once the preconditioner is initialized, we can call the iterative solution:

```
      ITER = 1000             ! max number of iterations
      RESID = 1d-8            ! threshold for \|RESID\|
      INFO  = 0               ! no troubles on input
      NUNIT = 6               ! output channel


      call slpbcgs(
     >  prevec0, n, iW,rW,
     >  matvec, n, ia,ja,a,
     >  rW(ipBCG), n, 8,
     >  n, f, u,
     >  ITER, RESID,
     >  INFO, NUNIT )

      if (INFO.ne.0) stop 'BiCGStab failed'
```

An example of calling program is in file `src/Tutorials/PackageILU/main_ilu0.f`.

## 5.4 ILU2 preconditioner

The ILU2 preconditioner is an ILU factorization with two thresholds proposed by I.Kaporin in 1998. For symmetric positive definite stiff systems it is shown to be robust and to give better convergence rates compared to other factorizations. It can be applied for non-symmetric matrices as well. The factorization of the input matrix $A$ satisfies the formula

$$A = LU + TU + LR - S$$

where $L, U$ are the first order factors, $T, R$ are the second order factors (kept and used in calculation, neglected after calculation), $S$ is the residual matrix (neglected during the calculation). The method seems to be a very flexible and powerful tool to construct efficient preconditioners for stiff matrices. The application of the preconditioner has two stages: initialization and evaluation. The initialization includes: (1) balance scaling, i.e. search for

diagonal matrices $D_l$, $D_r$ such that matrix $D_lAD_r$ has approximately balanced rows and columns; and (2) incomplete factorization of balanced matrix $D_lAD_r$. The evaluation must be performed at each step of an iterative method. It is provided by the routine `prevec2` accompanying the initialization routine `iluoo`. The user should only put the name `prevec2` as an input parameter in the iterative solver:

```
      external prevec2
      ....

      call slpbcgs(
&     prevec2, IPREVEC, iW,rW,
&     matvec, IMATVEC, ia,ja,a,
&     WORK, MW, NW,
&     N, RHS, SOL,
&     ITER, RESID,
&     INFO, NUNIT )
```

The initialization routine has the following parameters

```
      call iluoo (n, ia, ja, a, tau1, tau2, verb,
&      work, iwork, lendwork, leniwork,
&      partlur, partlurout,
&      lendworkout, leniworkout, ierr)
```

- `n` is the order of the square matrix $A$;
- `ia,ja,a` are two Integer and one Real*8 arrays containing matrix in the CSR format;
- `tau1` is the absolute threshold for entries of $L$ and $U$ (elements of $L$ and $U$ greater than $\tau_1$ will enter $L$ and $U$; recommended values lie in the interval [0.01; 0.1]);
- `tau2` is the absolute threshold for entries of $T$ and $R$ (elements not included in $L$ and $U$ but greater than $\tau_2$ will enter $T$ and $R$; recommended value lie in the interval $\tau_1^2$ or $5\tau_1^2 - 0.1\tau_1$);
- `verb` sets up the verbocity level: 0 means no output, positive means verbose output;
- `work,iwork,lendwork,leniwork` are working Real*8 and Integer arrays and their sizes;
- `partlur` is user defined partition of the available memory `work,iwork`, $L, U$ occupy `(1-partlur)* lendwork` and $R$ occupies `partlur*lendwork`);
- `partlurout` is the optimal partition computed during factorization (may be useful for the next factorization);

- `lendworkout,leniworkout` are the minimalist (round-off errors may cause a tiny underestimate) memory demands provided that the optimal partition LU/R of the available memory is used (may be useful for the next factorization);
- `ierr` is the integer error code (0 - successful factorization).

Below we present the basic blocks of a program solving a system with matrix `a, ia, ja` and a right hand side vector `f` by the BiCGstab method with the ILU2 preconditioner. First we define all necessary arrays and variables:

```
C Arrays for matrix  kept in CSR format
      Integer ia(maxn+1), ja(maxnz)
      Real*8   a(maxnz), f(maxn), u(maxn)

C Work arrays
      Integer   MaxWr,MaxWi
      Parameter(MaxWr=5*maxnz, MaxWi=6*maxnz)
      Real*8  rW(MaxWr)
      Integer iW(MaxWi)

C BiCGStab data
      External  matvec, prevec2
      Integer   ITER, INFO, NUNIT
      Real*8    RESID

C ILU data
      Real*8   tau1,tau2,partlur,partlurout
      Integer  verb, ierr, UsedWr, UsedWi

C Local variables
      Integer   ipBCG, ipIFREE
```

When the matrix is stored in the CSR format, we initialize the preconditioner by computing $L$ and $U$ factors and saving them in `rW, iW`:

```
      verb    = 0      ! verbose no
      tau1    = 1d-2
      tau2    = 1d-3
      partlur = 0.5
      ierr    = 0

      call iluoo (n, ia, ja, a, tau1, tau2, verb,
     &   rW, iW, MaxWr, MaxWi, partlur, partlurout,
     &   UsedWr, UsedWi, ierr)

      if (ierr.ne.0) then
         write(*,*)'initialization of  iluoo failed, ierr=',ierr
```

```
      stop
   end if

   if (UsedWr+8*n.gt.MaxWr) then
       write(*,*) 'Increase MaxWr to ',UsedWr+8*n
       stop
   end if
   ipBCG  = UsedWr + 1
```

Once the preconditioner is initialized, we can call the iterative solution:

```
   ITER = 1000             ! max number of iterations
   RESID = 1d-8            ! threshold for \|RESID\|
   INFO  = 0               ! no troubles on imput
   NUNIT = 6               ! output channel

   call slpbcgs(
  >  prevec2, n, iW,rW,
  >  matvec, n, ia,ja,a,
  >  rW(ipBCG), n, 8,
  >  n, f, u,
  >  ITER, RESID,
  >  INFO, NUNIT )

   if (INFO.ne.0) stop 'BiCGStab failed'
```

An example is given in file `src/Tutorials/PackageILU/main_ilu2.f`.

# Ani3D-INB Version 3.1 "*Starflower*"

# Flexible Iterative Solver Using Inexact Newton-Krylov Backtracking

## User's Guide for libinb-3.1.a

## 6.1 Basic features of the library

The FORTRAN package Ani3D-INB is an independent part of the package Ani3D . Ani3D-INB was developed by Alexey Chernyshenko under the supervision of Yuri Vassilevski. It is designated for the iterative solution of nonlinear systems. Ani3D-INB may be easily incorporated into any other software. The package interfaces the ILU precondtioners provided by the Ani3D-ILU package. The package Ani3D-INB is a deeply processed and essentially simplified version of the NITSOL package by Homer F. Walker.

The basic features of library *libinb-3.1.a* are listed below.

**Iterative method** : Inexact Newton-Krylov Backtracking (INB), with BiConjugate Gradient Stabilized (BiCGStab) iteration as the interior Krylov subspace solver

**Preconditioners** : Common interface with ILU0 and ILU2, the second order accurate ILU (provided by the Ani3D-ILU package).

**Problem setting** : User defined routine computing the nonlinear residual.

**Data format** : double precision or integer arrays. Enumeration starts from 1.

**Typical memory requests** : for systems with $N$ equations INB needs 11 work vectors of dimension $N$, one solution vector and a room for preconditioner data. If the preconditioner is built by the Ani3D-ILU package, ILU0 requires the same storage as the CSR representation of the jacobian, ILU2 requires 2-5-fold storage.

## 6.2 Iterative solution

The iterative solver is Inexact Newton-Krylov Backtracking (INB) method with inner linear solve BiConjugate Gradient Stabilized method (BiCGStab). This is the Newton type method applicable to non-singular nonlinear systems. It requires two procedures: evaluation of the nonlinear residual function and (optional) precondition-vector evaluation. The preconditioner should be an approximation of the inverse jacobian matrix. The jacobian matrix is not required explicitly due to the finite-difference evaluation of the jacobian-vector product. The call of the method is

```
      external prevec, funvec
      ....

      call slInexactNewton(
     &     prevec, IPREVEC, iWprevec, rWprevec,
     &     funvec, rpar, ipar,
     &     N, SOL,
     &     RESID, STPTOL,
     &     rWORK, LenrWORK,
     &     INFO)
```

- `prevec` is the name of a precondition-vector multiplication routine. `IPREVEC, iWprevec, rWprevec` are arrays (Integer, Integer, Real*8, respectively) of user's data which may be passed to `prevec` and used there. Arrays `iWprevec, rWprevec` are recommended to keep the preconditioner bulk data (triangular factors, for instance). Array `IPREVEC` may contain control parameters or basic user data such as the system order and useful pointers. In the presented example, `IPREVEC` contains a single entry equal to the system order. The format of `prevec` coincides with that from the package Ani3D-ILU :

```
      Subroutine prevec(IPREVEC, ICHANGE, X, Y, iW, rW)
c Input
      Integer IPREVEC(*), ICHANGE, iW(*)
      Real*8  X(*), rW(*)
c Output
      Real*8  Y(*)
```

  where $X$ is the input vector and $Y$ is the output vector. `ICHANGE` is the flag controlling the change of the preconditioner. It is useful when convergence stagnation occurs. `iW, rW` are Integer and Real*8 arrays, respectively, which store the preconditioner data. The package Ani3D-ILU provides two examples of routine `prevec` corresponding to two ILU preconditioners, `prevec0` (ilu0.f) and `prevec2` (iluoo.f). The details may be found in the user guide for Ani3D-ILU .

- `funvec` is the name of the user routine computing the nonlinear residual vector function and `ipar, rpar` are Integer and Real*8 arrays of user's data which may be passed to `funvec` and used there. The format of `funvec` is as follows:

```
      Subroutine funvec(n, xcur, fcur, rpar, ipar, itrmf)
c INPUT:
      Integer  n       ! dimension of vectors
      Real*8   xcur(*) ! current vector
      Real*8   rpar(*) ! double precision user-supplied parameters
      Integer  ipar(*) ! integer user-supplied parameters
c OUTPUT:
      Integer  fcur(*) ! nonlinear residual vector (zero for the solution)
      Integer  itrmf   ! flag for successful termination of the routine
```

  This routine calculates the nonlinear residual $F(X)$.

- `N` is order of system and length of vectors.

- `SOL` is the initial guess and the iterated solution (Real*8).

- `RESID` is the convergence criterion for the nonlinear residual.

- `STPTOL` is the stopping tolerance on the Newton's steplength.

- `rWORK(LenrWORK)` is Real*8 working array which stores at least 11 vectors of size $N$.

- `LenrWORK` is the total length of `rWORK` which must be not less than `11 N`.
- `INFO` is the array of control parameters. On input: `INFO(1)` sets initial value for successful termination flag, `INFO(2)` sets the maximal number of linear iterations per Newton step, `INFO(3)` sets the maximal number of nonlinear iterations, `INFO(4)` sets the maximal number of backtracks, `INFO(5)` sets the printing level (0 none, 1 nonlinear residuals, 2 nonlinear and linear residuals). On output: `INFO(1)` is the value of the termination flag (successful termination corresponds to 0), `INFO(2)` is the number of performed linear iterations, `INFO(3)` is the number of performed nonlinear iterations, `INFO(4)` is the number of actual backtracks, `INFO(5)` is the number of performed function evaluations.

Examples of using the package are in files `src/Tutorials/PackageINB/main_simple.f`, `src/Tutorials/PackageINB/main_bratu.f`, and `src/Tutorials/MultiPackage/StokesNavier/main.f`.

# Ani3D-LU "*Twinflower*"

# LU Factorization Solver for Sparse Systems

# User's Guide for liblu-3.1.a

## 7.1 Overview

The C package Ani3D-LU is a simplified version of UMFPACK-4.1 and AMD packages developed by Timothy A. Davis, Patrick R. Amestoy, and Iain S. Duff. It is designated for the direct solution of sparse linear systems. Ani3D-LU is an independent part of the package Ani3D .

Examples of using Ani3D-LU in FORTRAN programs are given in files `src/Tutorials/PackageLU/main.f`, `src/Tutorials/MultiPackage/Stokes/main.f`, `src/Tutorials/MultiPackage/StokesNavier/main.f`. For detailed documentation, see `doc/lu_guide.pdf`.

# Chapter 4

# SERVICE PACKAGEs

# Ani3D-LMR version 3.1 "*Cornflower*"

# Local Metric Recovery

# User's Guide for liblmr3D-3.1.a

## 8.1 Introduction

The FORTRAN package Ani3D-LMR is developed by Konstantin Lipnikov and Yuri Vassilevski. It generates continuous tensor metrics. The tensor components are piecewise linear functions defined on nodes of a given tetrahedral mesh. The generated metric may be used in the Metric Based Adaptation package Ani3D-MBA .

The input data for metric generation is either a discrete solution defined at mesh nodes, or cell-based or edge-based errors estimates, or edge-based error estimates.

The library *liblmr3D-3.1.a* can be incorporated into other packages.

This document describes the structure of the package, input data, and user-supplied routines. It presents a few examples illustrating details of the package.

## 8.2 Copyright and Usage Restrictions



## 8.3 Description of Ani3D-LMR

### 8.3.1 General structure of package

The package Ani3D-LMR consists of a few FORTRAN files. The routines in these files implement one of the following basic tasks:

1. Recovery of a nodal metric from a discrete nodal function;
2. Recovery of a nodal metric from an edge-based error estimator;
3. Recovery of a nodal metric from a cell-based error estimator;
4. Modification of a metric for error minimization in the $L_p$ norm.

These tasks will be discussed in subsequent sections.

In addition to the library Ani3D-LMR , package Ani3D contains a tutorial directory discribed in the last section.

### 8.3.2 Local metric recovery from discrete function

A nodal tensor metric may be recovered from the discrete function defined at nodes of the mesh. The metric is the spectral module of the discrete Hessian of this mesh function. A mesh that is quasi-uniform in this metric minimizes the maximum norm of the approximation error of an underlying continuous function. Two methods for the Hessian recovery are implemented in files `Nodal2MetricVAR.f` and `Nodal2MetricZZ.f`.

```
      Subroutine Nodal2MetricVAR(u,
     &                           Vrt,Nvrt, Tri,Ntri, Bnd,Nbnd, measure,
     &                           Nrmem,rmem, Nimem,imem)

      Subroutine Nodal2MetricZZ(u,
     &                          Vrt,Nvrt, Tri,Ntri, Bnd,Nbnd, measure,
     &                          Nrmem,rmem, Nimem,imem)

C  Input: u    - function defined at mesh vertices
C         Nvrt - the number of vertices
C         Vrt  - coordinates of these vertices
C         Ntri - the number of triangles
C         Tri  - the connecticity table
C         Nbnd - the number of boundary edges
C         Bnd  - the list of boundary edges
C
C  Output: measure - tensor metric
C
C  Work arrays: rmem - real*8 array of length Nrmem
C               imem - integer array of length Nimem
```

For the first method, the input mesh has to satisfy the following "two arms" condition. Every boundary node can be connected to an interior node with at most *two* mesh edges.

### 8.3.3 Local metric recovery from edge-based error estimator

Nodal tensor metric may be recovered from edge-based error estimates $\eta_{e_k}$. The metric may be anisotropic in this case. Two methods of metric recovery are implemented. The first method (`EdgeEst2MetricLS.f`) based on the Least Squares solution of the local system

$$(M(a_i)e_k, e_k) = \eta_{e_k}.$$

Here $M(a_i)$ is the tensor metric to be recovered at a mesh node $a_i$, $e_k$ are mesh edges incident to $a_i$, and $\eta_{e_k}$ are error estimates prescribed to these edges.

```
       Subroutine EdgeEst2MetricLS(nP, nE, XYP, IPE,
     &                             Error, Metric,
     &                             MaxWr, MaxWi, rW, iW)
c
c Input:
c      Integer nP, nF, nE    ! numbers of nodes and elements
c      Real*8  XYP(3, nP)    ! coordinates of mesh nodes
c      Integer IPE(4, nE)    ! connectivity table of elements
c      Real*8  Error(6, nE)  ! error estimates prescibed to element edges
c
```

```
c Output:
c      Real*8  metric(6, nP) ! node-based tensor metric
c
c Working arrays:
c      Integer iW(MaxWi)
c      Real*8  rW(MaxWr)
```

The second method (`EdgeEst2MetricMAX.f`) is called the method of shifts. First, it recovers a cell-based (piecewise constant) tensor metric and then for each mesh node $a_i$ picks a metric with the maximum determinant among all metrics in elements sharing the node $a_i$. The input parameters are the same as above.

```
      Subroutine EdgeEst2MetricMAX(Error,
     &                             nP, nE, XYP, IPE,
     &                             Metric, MaxWr, rW)
```

The above routines recover a tensor metric that can be used to minimize maximum norm of the interpolation error. The following routine (`EdgeEst2GradMetricMAX.f`) can be used to minimize maximum norm of the gradient of the interpolation error.

```
      Subroutine EdgeEst2GradMetricMAX(Error,
     &                                 nP, nE, XYP, IPE,
     &                                 Metric, MaxWr, rW)
```

These routines may be used for any edge-based errors, including interpolation errors and a posteriori error estimates, see the last section.

If an analytical function is available, the interpolation error can be calculated and the tensor metric can be build using the following two routines.

```
      Subroutine Func2MetricMAX(Func,
     &                          nP, nE, XYP, IPE,
     &                          Metric, MaxWr, rW)

      Subroutine Func2GradMetricMAX(Func,
     &                              nP, nE, XYP, IPE,
     &                              Metric, MaxWr, rW)
c
c  Input:
c      Func  - Real*8 Function f(xy), where xy(3) is point
```

Both routines use the method of shifts to build the metric.

### 8.3.4 Local metric recovery from cell-based error estimator

Nodal tensor metric may be recovered from cell-based error estimates $\eta_{\Delta_k}$:

$$M(\Delta_k) = \eta_{\Delta_k}.$$

The metric will be isotropic (scalar tensor) in this case. The nodal metric is generated by applying the ZZ recovery algorithm to a scalar cell-based metric (`CellEst2MetricZZ.f`).

```
      Subroutine CellEst2MetricZZ(nP, nF, nE, XYP, IPE, IPF,
     &                            Error, Metric,
     &                            MaxWr, MaxWi, rW, iW)
```

This method is recommended for problems with isotropic solutions.

### 8.3.5 Metric modification for error minimization in $L_p$

The above routines build tensor metrics to minimize of the maximum ($L_\infty$) norm of error. If the user wants to minimize the $L_p$ norm, he or she should modify the metric using the following routine:

```
      Subroutine Lp_norm(nP, Lp, Metric)
c
c     Real*8  Lp - norm for which the metric is to be adjusted:
c             Lp > 0  means  L_p     norm
c             Lp = 0  means  maximum norm (L_infinity)
```

## 8.4 Examples

Examples of usage of the package Ani3D-LMR are located in `src/Tutorials/PackageLMR`.

The program `mainNodal2Metric.f` demonstrates the local metric recovery from a given discrete function defined at mesh nodes. The metric is recovered by evaluating the discrete Hessian of this mesh function. The metric is build to minimize the $L_p$ norm of interpolation error.

The program `mainFunc2GradMetric.f` demonstrates building the optimal metric for minimizing $L_p$ norm of the gradient of the $P_1$ interpolation error. The metric is recovered using analytic representation of the interpolated function, since it requires nodal and mid-edge values of this function.

The program `mainEst2Metric.f` builds a metric from either errors defined at centers of mesh elements or mesh edges. This program calculates the maximum norm of the interpolation error on cells or edges for the user-defined function `Func(xyz)`.

The errors may be replaced with a posteriori error estimates.

# Ani3D-PRJ version 3.1 "*Feather Flower*"

# Finite Element $L^2$ Projection

# User's Guide for libprj3D-3.1.a

## 9.1 Basic features of the library

The FORTRAN-77 package Ani3D-PRJ is a part of the package Ani3D . It is designated for remapping data between two unstructured meshes using the conventional finite element $L^2$ projection. Intersection of two meshes (a metamesh) is constructed during assembling of the right-hand side, the most crucial part of the projection algorithm.

The package is organized as a library `libprj3D-3.1`. An example of using the library is `Tutorials/PackagePRJ/main.f`.

## 9.2 Usage of the library `libprj3D-3.1`

Given a finite element solution $u_h^{(2)} \in V_{h,2}$ on mesh $\Omega_h^{(2)}$, this library finds its finite element projection $u_h^{(1)}$ on to mesh $\Omega_h^{(1)} \in V_{h,1}$. The finite element spaces $V_{h,1}$ and $V_{h,2}$ may be different. We assume that the meshes occupy the same domain; if the domains are different, the algorithms stops with warning message.

Mathematical formulation of the problem is as follows: Find $u_h^{(1)}$ such that

$$\int_\Omega u_h^{(1)} v_h^{(1)} \,\mathrm{d}x = \int_\Omega u_h^{(2)} v_h^{(1)} \,\mathrm{d}x, \qquad v_h^{(1)} \in V_{h,1}.$$

The algorithm consists of calculating a metamesh, the right-hand side vector, the mass matrix, and solution of a linear system. This library performs the first two steps. The last two steps are performed using the packages Ani3D-FEM and Ani3D-ILU or Ani3D-LU .

The metamesh is created by calling the following routine:

```
      Call MetaMesh(nv, vrt, nt, tet, nv2, vrt2, nt2, tet2,
     &              nv12, nvMetaMax, vrt12,
     &              nt12, ntMetaMax, tet12, parents,
     &              MaxWi, MaxWr, iW, rW, iERR)
C
C    nv1, vrt1(2,nv1), nt1, tet1(3,nt1) - the first mesh
C    nv2, vrt2(2,nv2), nt2, tet2(3,nt2) - the second mesh
C    nv12, vrt12(2,nv12), nt12, tet12(3,nt12) - intersection of two meshes
C    parents(2,nt12) - two parents of new tetrahedra
C
C    rW(MaxWr) - Real*8 working memory of size greater than 5*nv2
C    iW(MaxWi) - Integer working memory of size greater than 11*nv2 + 16*nt2
```

The right-hand side vector is calculated from elemental contributions of tetrahedra in the metamesh using the library `libfem3D-3.1.a` of the package Ani3D-FEM . The assembling routine below allows the user to perform finite element projection not only in $L^2$-norm and also in energy norms. We describe only new parameters:

```
      Call assemble_rhs(nv1, vrt1, nt1, tet1, nv2, vrt2, nt2, tet2,
     &                  nv12, vrt12, nt12, tet12, parents,
     &                  operatorA, FEMtypeA, operatorB, FEMtypeB,
     &                  RHS, U2, MaxWi, iW)
```

```
C
C    operatorA - differential operator in front of u_2, e.g. IDEN or GRAD
C                as described in the package AniFEM
C    FEMtypeA  - finite element space V_h1, e.g., FEM_P1
C
C    operatorB - differential operator in front of u_1
C    FEMtypeB  - finite element space V_h2
C
C    U2(*)  - a given finite element solution on the second mesh
C    RHS(*) - right-hand side on the first mesh
C
C    iW(MaxWi) - integer working memory of size
C                10*(nt1+nt2) + max(nv1,nv2) + 4*max(nt1,nt2)
```

Let `M11` be the mass matrix in space $V_{h,1}$. Then, a finite element vector `U1` corresponding to $u_h^{(1)}$ is calculated by solving the problem `M11 U1 = RHS`.

# Ani3D-VIEW version 3.1 "*Coneflower*"

# Visualization Toolkit

# User's Guide for libview3D-3.1.a

## 10.1 Overview

Ani3D-VIEW is a simple visualizing library producing GMV-files of a mesh and discrete solution.

Self-instructive examples of using Ani3D-VIEW are given in `src/Tutorials/PackageVIEW/main.f`.

# Ani3D-C2F version 3.1 "*Fleeceflower*"

# C-wrapper for FORTRAN Packages

# User's Guide for libc2f3D-3.1.a

The C package Ani3D-C2F is a simple C-wrapper to call mesh generation routines from package Ani3D-MBA in a C program. In future releases Ani3D-C2F will be extended by C-wrappers to Ani3D-RCB , Ani3D-FEM , Ani3D-ILU .

Examples of using Ani3D-C2F in C programs are given in files new
`src/Tutorials/PackageC2F/main_nodal.c` and
`src/Tutorials/PackageC2F/main_analytic.c`.

# Notes for Windows users

## 11.1 Requirements

In order to compile Ani3D under Windows, you'll need the following software:

1. MinGW with gcc, g77 (or gfortran) and mingw make
2. MSYS base system
3. GMV viewer

MinGW (Minimalist GNU for Windows) provides Windows port of GNU binutils and GNU compiler collection. Tested version is 5.1.4 (gcc 3.4.5). MinGW can be downloaded from
`http://sourceforge.net/project/showfiles.php?group_id=2435`.
MSYS (Minimal SYStem) adds UNIX terminal emulator to MinGW. MSYS can be downloaded from the same site as MinGW. Tested version is 1.0.11.
Ani3D-VIEW library outputs meshes and solutions as GMV-files. To view these files, you'll need GMV viewing software
`www-xdiv.lanl.gov/XCM/gmv/GMVHome.html`.

## 11.2 Ani3D compilation in MinGW

In general, there is no difference between compiling Ani3D under Windows and UNIX, so you may refer Ani3D documentation on compilation. It is recommended to execute all make commands under MSYS terminal. Before compiling provide path to GMV viewing program to Ani3D by editing `src/Rules.make`, `src/aniAFT/src/cflags.make` files in Ani3D distribution.
For example, if you are using GSview installed in Program Files directory, then Rules.make should look like this:

```
...
F77 = g77 # Fortran compiler
CC  = gcc # C compiler
CXX = g++ # C++ compiler
VIEWER  = "C:\Program Files\GMV\gmv.exe"
...
```

Please notice the presence of quotes in VIEWER variable assignment: you should use them when path to gmv.exe contains spaces.

If you are planing to use Ani3D with MinGW (i.e. you're using gcc and g77 compilers supplied with MinGW) then everything is ready for it. You can check Ani3D on any example supplied with it (examples are located in src/Tutorials directory).

## 11.3 Microsoft Visual C++ and Intel Visual Fortran

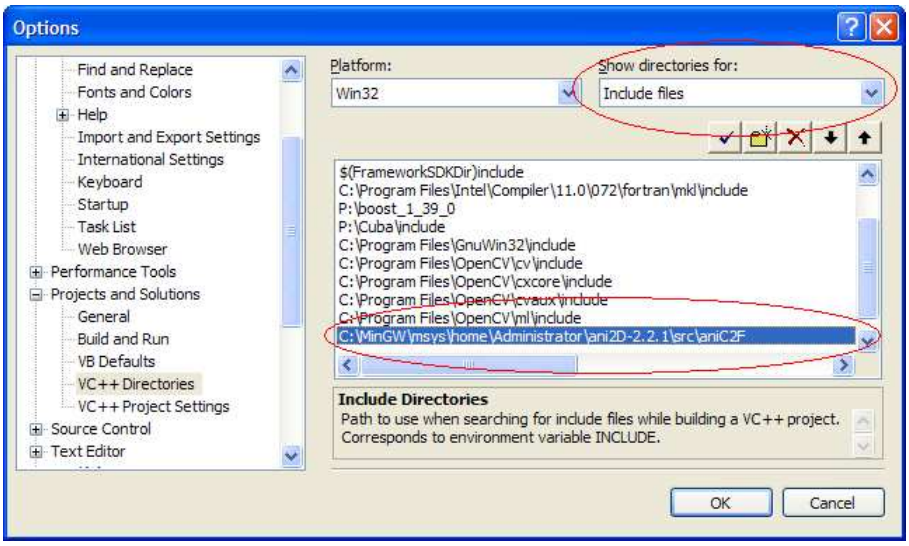
Under Windows Ani3D can be used with Microsoft Visual C++ and Intel Visual Fortran. We've tested Ani3D with Microsoft Visual C++ 2008 and Intel Visual Fortran 11. Ani3D should also be compatible with other versions of these workbenches.

### 11.3.1 Configuring Microsoft Visual C++ 2008

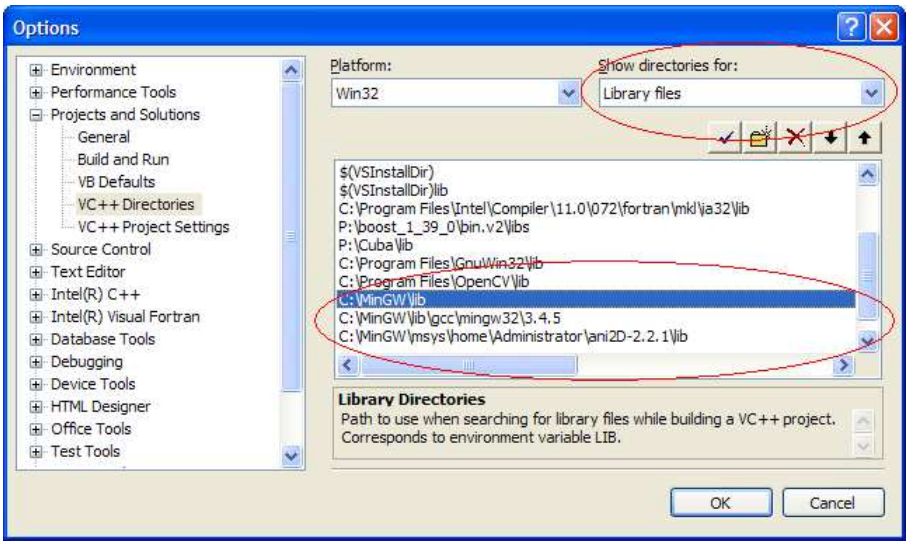
Before compiling programs, you should configure paths to Ani3D headers and libraries. Suppose that MinGW is installed in C:\MinGW and Ani3D is located in C:\MinGW\msys\home\user\ani3D-2.3.

In Visual Studio open *Tools* → *Options* → *Projects and Solutions* → *VC++ Directories* and add the following directories:

- Include files: C:\MinGW\msys\home\user\ani3D-2.3\src\aniC2F




- Library files: C:\MinGW\msys\home\user\ani3D-2.3\lib, C:\MinGW\lib, C:\MinGW\lib\gcc\mingw32\3.4.5




### 11.3.2 Linking in Microsoft Visual C++ 2008 projects

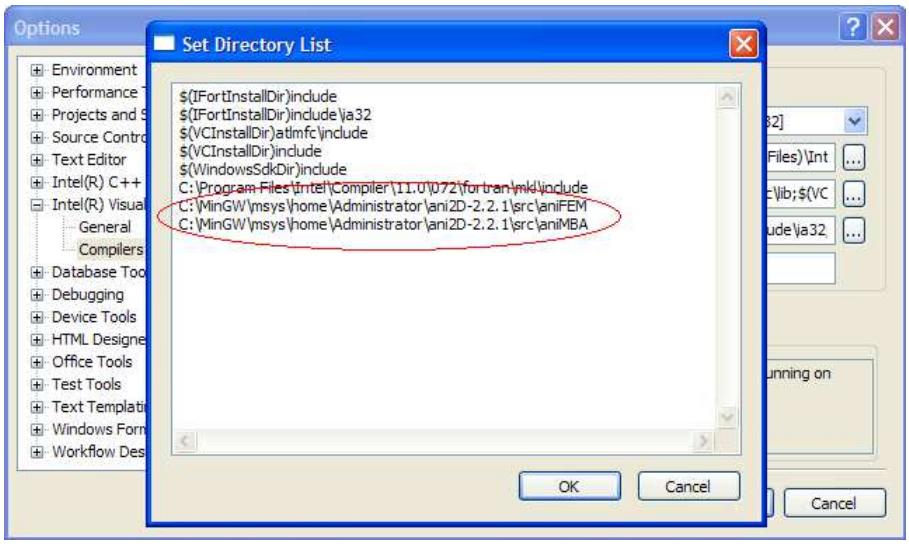
In addition to required Ani3D libraries, you need to link with libg2c.a, libgcc.a and libmingwex.a.

### 11.3.3 Configuring Intel Visual Fortran 11

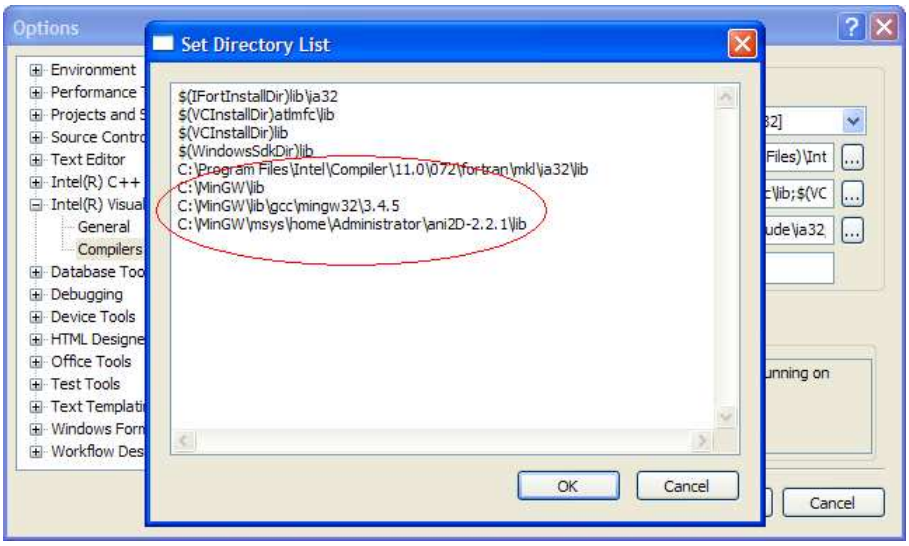
Visual Fortran also requires you to configure library directories.

In Visual Studio open *Tools* → *Options* → *Intel(R) Visual Fortran* → *Compilers* and add the following directories:

- Includes: C:\MinGW\msys\home\user\ani3D-2.3\src\aniFEM and C:\MinGW\msys\home\user\ani3D-2.3\src\aniMBA




- Libraries: C:\MinGW\msys\home\user\ani3D-2.3\lib, C:\MinGW\lib, C:\MinGW\lib\gcc\mingw32\3.4.5

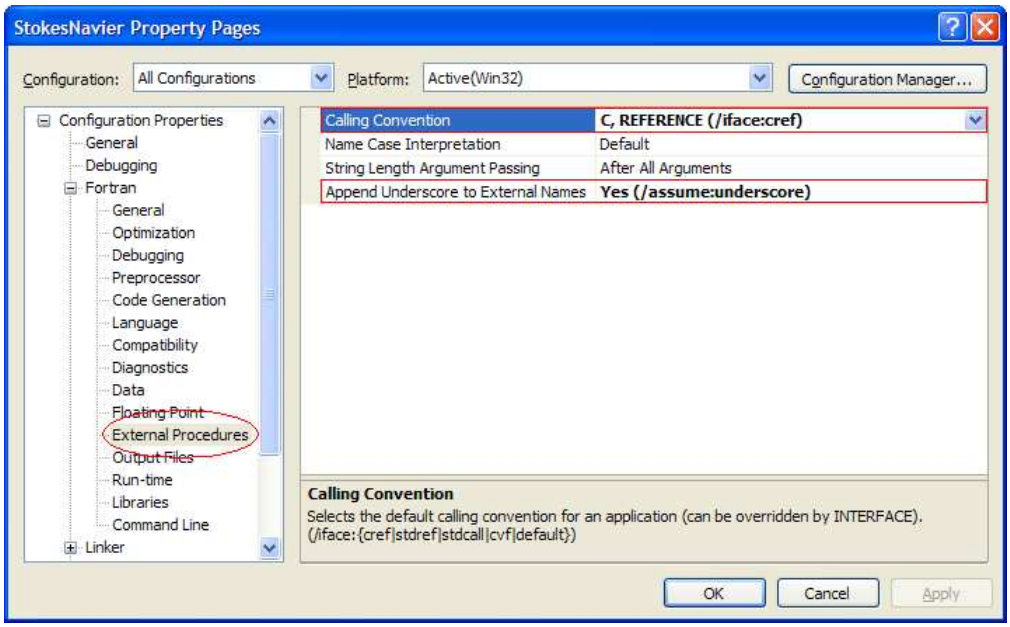



### 11.3.4 Linking in Intel Visual Fortran 11

In addition to required Ani3D libraries, you need to link with libg2c.a, libgcc.a and libmingwex.a.

### 11.3.5 Fortran naming conventions

Visual Fortran's naming conventions are not compatible with libraries generated by g77. To link successfully with Ani3D you'll need to specify correct naming conventions in your projects. This is done through *Project* → *Project Properties* dialog box.




This makes linking successful. But sometimes you'll still get "unresolved external" errors for some subroutines. For such subroutines just append underscore to its name, e.g. `Lp_norm` → `Lp_norm_`.

### 11.3.6 Visual C++ step-by-step tutorial: MeshAnalytic

Let's demonstrate usage of Ani3D in Visual C++.

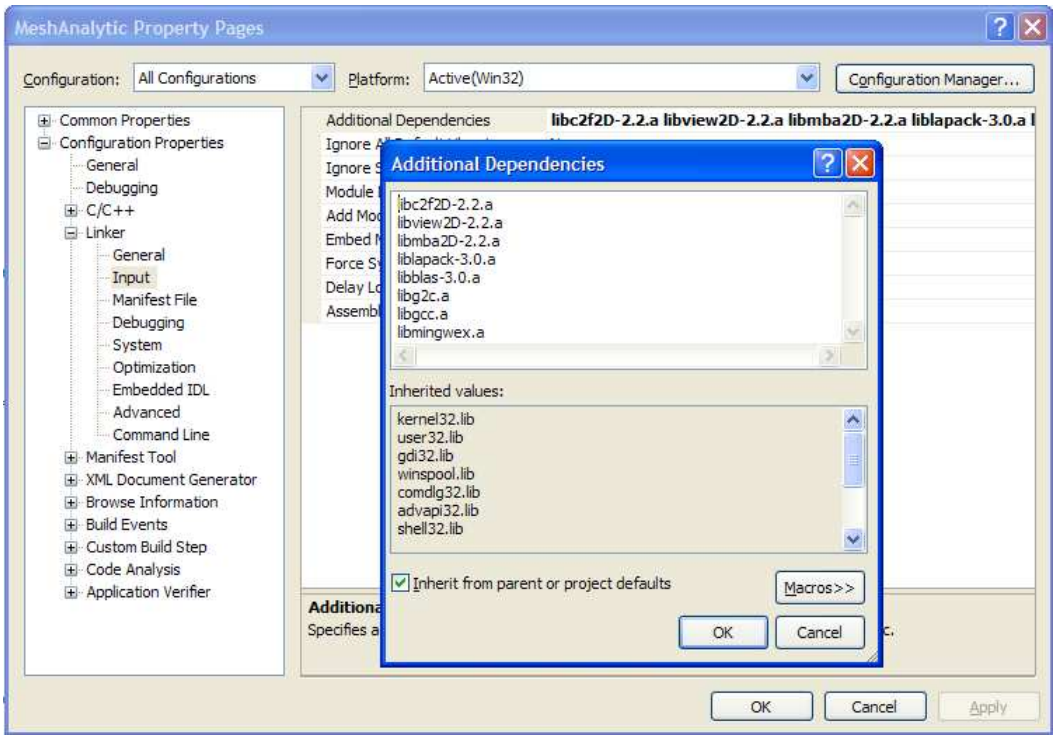
Here we'll compile MeshAnalytic Ani3D example (PackageC2F/main_analytic.c) in Visual C++.

1. Create an empty Win32 Console Application (don't forget to check "Empty project" check box in Win32 Application Wizard)

2. Add main_analytic.c from ani3D/src/Tutorials/PackageC2F into project

3. If needed fix paths to meshes in source code (lines 37, 89 and 94)

4. In *Project* → *MeshAnalytic Properties* → *Configuration Properties* → *Linker* → *Input* → *Additional Dependencies* list required libraries: libc2f3D-2.3.a libview3D-2.3.a libmba3D-2.3.a liblapack-3.1.a libblas-3.1.a libg2c.a libgcc.a libmingwex.a




Application is ready to run.

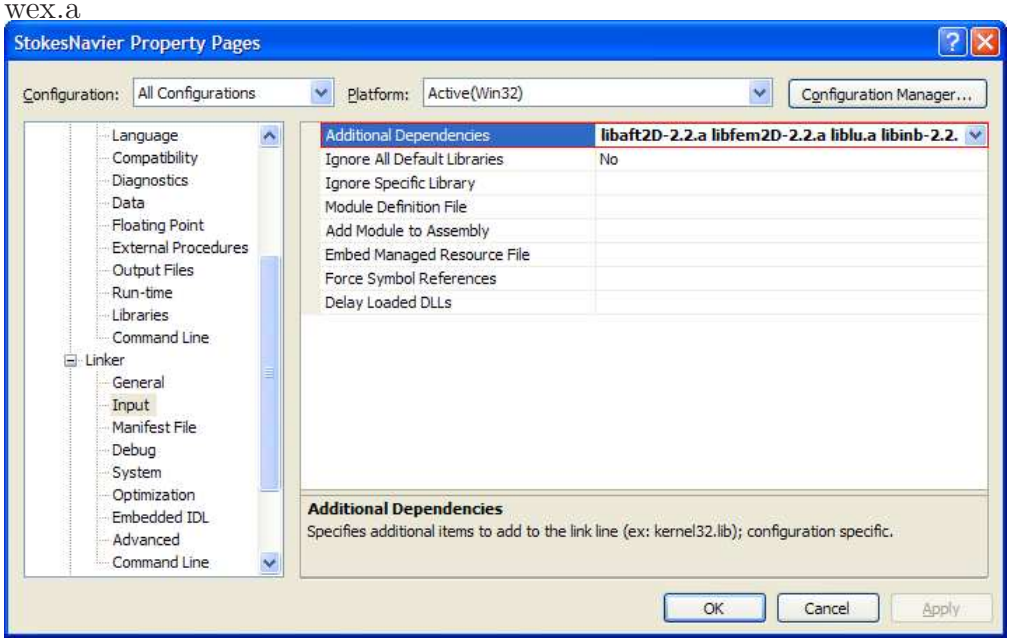
### 11.3.7 Visual Fortran step-by-step tutorial: StokesNavier

Let's demonstrate usage of Ani3D in Visual Fortran. Here we'll compile StokesNavier Ani3D example (MultiPackage/StokesNavier) in Visual Fortran.

1. Create an empty console project

2. Add main.f and forlibfem.f from ani3D/src/Tutorials/MultiPackage/StokesNavier into project

3. In *Project* → *MeshAnalytic Properties* → *Configuration Properties* → *Linker* → *Input* → *Additional Dependencies* list required libraries: libfem3D-2.3.a liblu.a libinb-2.3.a libmba3D-2.3.a libview3D-2.3.a liblapack-3.1.a libblas-3.1.a libg2c.a libgcc.a libmingwex.a




4. Try compiling the project. You should get 3 "unresolved externals" errors. Fix them by adding underscores to subroutine names as described in "Naming conventions" section

5. Tutorial now compiles and works correctly